MODELLING MEMORY AS CONDITIONAL PHENOMENA FOR A NEW THEORY OF COMPUTING

Weerakoon Arachchilage Chinthanie Weerakoon

(118031T)

Degree of Doctor of Philosophy

Department of Computational Mathematics

University of Moratuwa
Sri Lanka

June 2020

MODELLING MEMORY AS CONDITIONAL PHENOMENA FOR A NEW THEORY OF COMPUTING

Weerakoon Arachchilage Chinthanie Weerakoon

(118031T)

Thesis submitted in partial fulfillment of the requirements for the degree Doctor of Philosophy

Department of Computational Mathematics

University of Moratuwa
Sri Lanka

June 2020

Declaration

I declare that this is my own work and this thesis does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief, it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to University of Moratuwa the non-exclusive right to reproduce and distribute my thesis, in whole or part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books)

Signature: Date:

The above candidate has carried out the research for the PhD thesis under my supervision.

Signature of the supervisor: Date:

Signature of the supervisor: Date:

Dedicated To

My Loving

Mother, Father,

Husband, Son Vikum & Daughter Imandi

Acknowledgement

Through the difficult journey towards making this research work a reality, many individuals have helped me. I take this opportunity sincerely appreciate them all.

It is with a deep sense of gratitude that I acknowledge the guidance and encouragement gave me by my supervisor, Prof. Asoka Karunananda, who has allowed me to work in this research and supported me during the last few years with his patience, kindness and the knowledge while allowing me a room to work in my own approach. It was really hopeful that he was believing my work. Without his guidance this research won’t be realistic. Further, Prof. Asoka always encouraged and taught me how to attach to the research work through all the difficulties. Moreover, professor provoked my long forgotten bond on my religion and always advised to correct the way how I present.

I would like to offer my greatest gratitude to my co-supervisor Prof. N. G. J. Dias for introducing Prof. Asoka Karunananda and arranging an opportunity to talk to him. Furthermore, his patience, kind advices, guidance given during all the time was magnificent. Prof. Dias facilitated me with necessary facts with the past experience. Sometimes, he foresaw the difficulties that could arise, and always advised me to protect my dignity and to correct my faults. Further, prof. Dias commented on the work as quick as possible.

Special thanks goes to Dr. L. S. K. Udugama, Dr. (Mrs.) Uditha Rathnayake, Prof. T. S. G. Peiris, and all the members who were in my bi-annual progress panels for giving me the constructive comments on my research work.

I can’t forget all the members of the Department of Computational Mathematics, University of Moratuwa. Exceptional thanks goes to Ms. Dilini Kaluwansa, Dr. (Mrs.) Subha Fernando, and Dr. (Mrs.) Thushari Silva. Their encouragements and the friendliness made me so comfort.

I am really grateful to the CEO, CodeGen International, Dr. Harsha Subhasinghe, with a single request he generously approved funds from CodeGen to publish my first SCOPUS indexed conference article. My supervisor, Prof. Asoka Karunananda made this connection through Mr. Viraj Dayarathne, a kind hearted person that we could rarely find.

I would like to thankfully remind all the academic and non-academic members of University of Kelaniya those who have supported me during the work. Special thanks goes to the former vice-chancellor Prof. Sunanda Madduma Bandara, the vice-chancellor Prof. D. M. Semasinghe, the dean, Faculty of Science, and the head of the department of Statistics & Computer Science Dr. (Mrs.) D. D. M. Jayasundara for granting me the study leave to complete my degree. In addition to that, Dr. Dhammika Weerasinghe, and Mr. Buddhika Godakuru in ICT center, University of Kelaniya, allowed me to execute bulk programs on the servers there. Moreover, Mr. Buddhika Godakuru was a good technical commenter on my programs.

Again, I would like to mention Dr. (Mrs.) D. D. M. Jayasundara and all the current members of my department for giving me all the encouragements and the freedom for my work even with their busy schedules and difficulties. Further, Dr. (Mrs.) Carmel Wijegunasekara and all the former members of our department gave me a great backing.

I am heartily thankful to Dr. (Mrs.) Mihirini Wagaarachchi, my research companion, she was not a mere colleague, she was someone, which I could always talk to, her advices, reassurances, and kindness always made me strong. And also, I like to thank my colleagues, Mr. Buddhitha Hettige and Mrs. Hansika Gunasekara, their generous supports were also excellent. All the time, I had blessings and well wishes from many other relatives, and friends. Further, I would like to remind the kind and generous individuals those who got to know through SLAAI. They encouraged me to complete this as soon as possible. I like to take this opportunity to thank them all.

I would like to extend my greatest gratitude to my husband Amal for his support given me in clarifying things where I am not clear about. I can talk to him, he listens, bare all the difficulties and my moods. His patience and kindness showed towards me as being my loving husband, protecting and taking care of me. Further, I also grateful to my husband's father and mother. My little son Vikum, as his father bare all my absences without a single complaint. I know he loves me a lot. Then, my little baby Imandi, she always seeks my presence and warm touch. She comes with a finger in her mouth and pat me smoothly, when I am in the computer. My son's silent patience and my daughter's heartfelt requests encouraged me to finish the work as soon as possible. Finally, I would like to give my heartfelt thanks to my ever loving parents for their dedication, patience, guidance, protection, encouragement and all the conveniences given me, all the time, being behind myself, my husband, and my babies. They released me from everything. The words are not enough to express my gratitude towards them. This period was full of hospitalizations and surgeries. As the only child of my parents, I could have managed those and looked after my family. Sometimes, I failed and was lost. But, I could rise again. I always feel that I am gifted because of them.

Their patience, munificence and excellent supports were admirable.

Abstract

Computation in Von-Neumann architecture was quite different from the computation in the human mind, which processes in association with the brain by improving quality, accuracy and speed over the generations of execution of instructions. It was argued that this difference has been primarily caused by the separation of memory from processor, which results in delay in processing in the Von-Neumann architecture. Therefore, to improve computational efficiency on Von-Neumann architecture, various hardware and software level improvements have been introduced. In this sense, many researches were done in order to produce hardware level solutions, but there are limited researches to produce software level solutions. As such researches into develop new computing models at software level has been a research challenge. Our research has also discovered that despite the neuroscience of brain has inspired various computing models, behavior of mind has not been exploited to build models for computation.

As inspired by a theory of mind from an Eastern philosophy, Theravada Buddhism, we postulate the memory as a result of processing, and the memory and processing are not separated. The mind as a processor executes a conditional flow of thoughts pertaining to five-sense doors or the mind itself. The processing mechanism in the mind results an evolving memory. This thesis presents a novel **S**ix-State Processing Model (SSPM), which implements the processing in the mind and causing an evolving memory to improve processing speed of the computer. The six-state of SSPM encompasses New, Ready, Running, Blocked, Sleep, and Terminate, where the states Sleep and Terminate are new and are variations of the Exit in five-state model. Further, the SSPM has a set of newly defined transitions Ready-Ready, Sleep-Ready, Running-Ready, Running-Sleep, and Ready-Terminate that are associated with the concepts exploited from the Causal Relations of Buddhist Theory of Mind. Due to these states and transitions, the new model SSPM exhibits three distinct features, namely, internal and external processes, continuous processing, and a smaller tactics memory. Altogether, the SSPM works as a mind-like computer.

The evaluation of the SSPM has been conducted both in empirical level and the theoretical level. The empirical level testing was carried out separately for several computing programs that have been customized by the SSPM, where a Fraction Calculator (FC), a Quadratic Equation Solver (QES), a Sorting Program, and a Simulated Process Scheduler (PS) were among the programs. Further, the customized programs were named as SSPM-FC, SSPM-QES, SSPM-Sorting, and SSPM-PS. In fact, SSPM-FC considered a set of operations that included Plus, Minus, Multiplication, and Division, while the SSPM-Sorting had two categories such as SSPM-S-Insertion and SSPM-S-Equal. Furthermore, SSPM-FC (with Plus, Minus, and Multiplication), SSPM-QES, SSPM-Sorting (SSPM-S-Insertion, and SSPM-S-Equal), and SSPM-PS were tested separately under several testing scenarios to check their ability to gain improvements in the subsequent program execution cycles. Hence, it could prove the ability of the SSPM-system to improve the computing efficiency of the system in consecutive execution cycles of the system. Next, with SSPM-FC, SSPM-QES, and SSPM-Sorting, it could be demonstrated that how the smaller tactics memory is improved over the time. In addition to that, the speedups gained by the SSPM-S-Insertion and the SSPM-S-Equal with compared to the original Quicksort were compared with the speedups gained by the quicksort programs implemented with some other computing approaches. There, the SSPM-S-Equal case showed speedup with compared to the original quicksort for all the tested lists (number of elements were varying from approximately 2 to 4M). However, the SSPM-S-Insertion had limiting conditions in showing the better performance. So then, the smaller tactics memory with the SSPM-Sorting could be organized and the appropriate mechanism could be selected as per the requirements over program execution cycles improving the computing power of the system. Finally, to evaluate the model in the theoretical level, SSPM has been simulated with a Turing Machine. Afterwards, checking the satisfiability, it has been proved the NP-completeness of SSPM. Hence, its computability and the real-world applicability has been theoretically proved. Overall, it has been able to prove that the solutions can be provided faster over subsequent execution cycles by modelling memory as conditional phenomena and leads to a new theory of computing.

Keywords: Evolving Tactics Memory, Six-State Processing Model, Continuous Processing, NP Complete, Special Compiler, 24-Causal Relations

Table of Contents

CHAPTER 01......1

INTRODUCTION......1

1.1 Prolegomena 1

1.2 Aims and Objectives...... 2

1.3 Background and Motivation 2

1.4 Inquisitiveness in Brief 4

1.5 Memory and Processing in Current Computing Models 5

1.6 The Proposed Computing Model...... 7

1.7 Testing and Evaluation 11

1.8 Resource Requirements 12

1.9 Contribution to the Field of Computer Science 12

1.10 Organization of the Thesis 12

1.11 Summary 13

CHAPTER 02......15

REVIEW OF MEMORY AND PROCESSING MODELS......15

2.1 Introduction...... 15

2.2 Hardware Improvements...... 15

2.3 Improvements for Memory and Processing in Software Level 18

2.3.1 Processing Models in Operating Systems 18

2.3.1.1 Two-State Model 19

2.3.1.2 Three-State Model 19

2.3.1.3 Five-State Model...... 19

2.3.1.4 Seven-State Model...... 20

2.3.1.5 Processing Model in Linux OS...... 20

2.3.1.6 Kernel Thread Model of Windows OS 21

2.3.1.7 Kernel Thread Model of Solaris 21

2.3.2 Incremental Computing...... 23

2.3.2.1 Self-Adjusting Computation...... 24

2.3.2.2 Imperative Self-Adjusting Computing 25

2.3.2.3 Incoop: MapReduce for Incremental Computations 25

2.3.2.4 ADAPTON – Compassable, Demand-Driven Incremental Computation 26

2.3.2.5 iThreads: A Threading Library for Parallel Incremental Computation.... 26
2.3.3 Multi-agent Systems and Evolutionary Computing.................................... 27
2.3.4 Neural Computing.. 28
2.3.5 Program Tuning with Adaptability 30
2.4 Quantum Computing.. 34
2.5 Summary... 35
CHAPTER 03...37
THEORETICAL FOUNDATION FOR THE NOVEL COMPUTING MODEL...37
3.1 Introduction.. 37
3.2 Buddhist Theory of Mind 38
3.2.1 Thought-process 38
3.2.2 Explanation for the Human Memory from BTM 42
3.2.3 Twenty-four Causal Relations in BTM 43
3.2.4 Exploiting Twenty-four Causal Relations in Explaining the Thought-process ……………………………………………………………………..47
3.3 Real-World Inspirations.. 48
3.4 Attempts in Computer Modelling of Human Mind Based on BTM................ 50
3.5 Other Approaches on Mind and Memory 52
3.6 Human Memory Models.. 53
3.7 Problems in implementable Theories of Mind 54
3.8 Summary... 55
CHAPTER 04...56
NOVAL APPROACH TO A COMPUTING MODEL..56
4.1 Introduction.. 56
4.2 Hypothesis 56
4.3 Input.. 58
4.4 Output 60
4.5 Propose the Computing Model 60
4.6 Features of the New Model.. 62
4.7 Users 64
4.8 Exploiting Twenty-four CRs in Modelling the SSPM.................................... 64

4.9 Implementation 71
4.9.1 Fraction Calculator (SSPM-FC) 72
4.9.1.1 Why use the technologies 72
4.9.1.2 The Implementation Process 73
4.9.2 Quadratic Equation Solver (SSPM-QES) 77
4.9.3 Sorting Program (SSPM-Sorting) 77
4.9.4 SSPM-PS 78
4.10 Experimental Mechanism of SSPM 79
4.11 Simulation of SSPM in a Turing Machine 82
4.12 Summary 82
CHAPTER 05 83
HOW THE SYSTEM WORKS 83
5.1 Introduction 83
5.2 Fraction Calculator (SSPM-FC) 83
5.3 Quadratic Equation Solver (SSPM-QES) 96
5.4 Sorting Program (SSPM-Sorting) 97
5.5 Simulated Processes Scheduler (SSPM-PS) 101
5.6 Summary 103
CHAPTER 06 104
TESTING AND EVALUATION 104
6.1 Introduction 104
6.2 Experimental Mechanism of SSPM-FC. 104
6.3 SSPM-FC - Testing Scenario 1 105
6.3.1 Experimental Setup 105
6.3.2 Choice of Expressions and The Responses 105
6.3.3 Testing Scenario 1.1: Addition (Plus Operator): 106
6.3.4 Testing Scenario 1.2: Subtraction (Minus Operator) 112
Step 1: 112
6.3.5 Testing Scenario 1.3: Multiplication (Multiplication Operator) 118
Step 1: 118
6.4 SSPM-FC - Testing Scenario 2: 124
6.4.1 Experimental Setup 125

6.4.2 Choice of Inputs and The Responses 125
6.4.3 Relevant Statistical Tests 128
6.4.4 The Scenario 128
Step 1: 128
6.5 Summarizing Results of the Experiments on SSPM-FC 134
6.6 SSPM-QES Testing Scenario 135
6.7 Experimental Mechanism on the Implementation of Sorting Program 139
6.8 SSPM Sorting - Testing Scenario 1: 139
6.8.1 Experimental Setup 139
6.8.2 Choice of Expressions and The Responses 140
6.8.3 Testing Scenario 1.1: InsCalcModule 140
6.8.4 Testing Scenario 1.2: Similar List to Sort (EquCalcModule) 144
6.9 Trade-Offs: SSPM Sorting Vs Original Quicksort-Testing Scenario 1 147
6.10 SSPM Sorting - Testing Scenario 2 153
6.10.1 SSPM Sorting - Testing Scenario 2.1 153
6.10.2 SSPM Sorting - Testing Scenario 2.2 156
6.10.3 SSPM Sorting - Testing Scenario 2.3 158
6.11 Formal Verification 161
6.11.1 Why the Turing Machine has been Used? 161
6.11.2 Nondeterministic Turing Machine (NTM) 162
6.11.3 Configurations of NTM 165
6.11.4 Satisfiability of NTM 168
6.11.5 Results of Formal Verification 174
6.11.6 Time Complexity 174
6.12 Summary 175
CHAPTER 07 176
CONCLUSION AND FUTURE WORK 176
7.1 Introduction 176
7.2 Modelling Memory as Conditional Phenomena for a New Theory of Computing 176
7.2.1 Critical study about various models for computing 176
7.2.2 In depth study about the Buddhist Theory of Mind 177

7.2.3 Propose a new computing model where the memory is a result of continuous processing 177

7.2.4 Customizing Programs with SSPM 180

7.2.5 Evaluate the proposed model 180

7.3 Limitation 182

7.4 Further Work 182

7.5 Summary 184

References 185

APPENDIX A 194

SELECTED CODE SEGMENTS 194

A.1 Process Switcher 194

A.2 Operation Organizer 195

A.3 Input-Content Analyser of SSPM-FC 199

A.4 Small Compiler 203

A.5 Write Engine 204

A.6 Terminating Point 206

A.7 Two Methods for Multiplication 207

APPENDIX B 208

DATA SETS 208

B.1 SSPM-FC: Plus Operator 208

B.2 SSPM-FC: Minus Operator 209

B.3 SSPM-FC: Multiplication Operator 210

B.4 SSPM-QES – Positive Discriminant 211

B.5 SSPM-Sorting 212

APPENDIX C 213

PUBLICATION 213

List of Figures

Figure. 1.1: Where the proposed model is placed…………………………….... 09
Figure. 3.1: Eye-door thought-process (three stars denotes the sub-moments in each thought-moment) at the present of a very great object …………………. 40
Figure. 3.2: Mind door thought-process (three stars denotes the sub-moments in each thought-moment) at the present of a clear object…………………….. 41
Figure. 3.3: Memory is a continuous thought process...................................... 42
Figure. 4.1: High Level Block diagram for the proposed computing model …. 57
Figure. 4.2: Input Patterns (IP) (a) Patterns (b) Example for each pattern…...... 59
Figure. 4.3: Six-state Continuous Processing Model (SSPM)………………… 61
Figure. 4.4: Conditionally evolving smaller tactics memory………………….. 63
Figure. 4.5: High level Flow chart for the proposed model…………………… 74
Figure. 4.6: How the modules have been connected………………………….. 75
Figure. 4.7: Algorithm for Fraction Calculator (SSPM-FC). (a) Fraction Calculator as a whole, (b) Input-Content Analyser, (c) Process Switcher, (d) Operation Handler, (e) Write Engine, (f) Small Compiler……………………. 76
Figure. 4.8: Algorithm for Simulated Process Scheduler…………………….. 79
Figure. 4.9: Sample size determination using MedCalc for testing scenario 1.1 SSPM-FC……………………………………………………………….. 81
Figure. 5.1: Smaller tactics memory starts with the entries relevant to the instructions that are inserted into the system……………………………….. 85
Figure. 5.2: Identify the Input (a) Different Op, Same Exp, Same Op, (b) New Op……………………………………………………………………… 86
Figure. 5.3: Current Fractional Expression Contains a New Operator..………. 87
Figure. 5.4: (a) Insert the Relevant Instruction set for the New Op (b) Create and Save the Library……………………………………………………… 88
Figure. 5.5: Update the Smaller tactics memory with an entry for the instruction set of the New Op……………………….……………………… 89
Figure. 5.6: For the frequently executing plus operator, the module is created. (a) The respective entry is updated to a 5-tuple in the smaller tactics memory, (b) File of the particular module is created and stored……….………........ 91
Figure. 5.7: Update the relevant entry in the smaller tactics memory due to deleting the inefficient method(a) How the states are changing, (b) How the smaller tactics memory and the program update ………………..………… 93
Figure. 5.8: User interface with all the other details………………………….. 94
Figure. 5.9: User interface for Quadratic Equation Solver when a class has been created for positive discriminant module (updated record in the smaller tactics memory), during an external process…………………..……………….. 96
Figure. 5.10: User interface for SSPM-Sorting, before introduce delete to the system………………………………………………………………………….. 98
Figure. 6.1: Time values taken for computing fractional expressions with Addition in SSPM-FC- before and after the change …..………………………. 106

Figure. 6.2: Probability Plot for addition (Difference) (a) Probability Plot (b)Values………………………………………………………………… 108
Figure. 6.3: Outlier Plot Addition (a) Outlier Plot (b) Values for difference... 109
Figure. 6.4: Power Curve for Paired t Test with size 100 (Addition)………… 110
Figure. 6.5: Time values collected before and after organizing the smaller tactics memory of the FC with subtraction…………………………………….. 113
Figure. 6.6: Probability plot of difference for Subtraction…………………….. 114
Figure. 6.7: Outlier Plot of Subtraction (a) Outlier Plot (b) Values for difference…………………………………………………………... 115
Figure. 6.8: Power Curve for Paired t Test with size 100 (Subtraction) ……… 116
Figure. 6.9: Time values collected before and after organizing the smaller tactics memory of the FC with Multiplication operator. (it is possible to refer the complete samples in the appendix)………………………………... 119
Figure. 6.10: The probability plot of differences (Multiplication)…………….. 120
Figure. 6.11: Outlier Plot of Multiplication (a) Outlier Plot (b) Values for difference ……………………………………………………………… 121
Figure. 6.12. Power Curve for Paired t Test with size 100 (Multiplication)…... 122
Figure. 6.13: Total time values in nanoseconds for best multiplication algorithms with selection process vs inefficient multiplication algorithm (MulCalc2())………………………………………………………………… 129
Figure. 6.14: The probability plot of differences (Total Time Values)……….. 130
Figure. 6.15: Outlier Plot of Difference (a) Outlier Plot (b) Values (Total Time values)………………………………………………………………… 131
Figure. 6.16: Power Curve for Paired t Test with size 20 (Total Time Values).. 132
Figure. 6.17: Time values recorded with the quadratic equations with positive discriminant before and after organizing the smaller tactics memory of the QES……………………………………………………………………….. 135
Figure. 6.18: Time values recorded for sorting lists (InsCalcModule) before and after organizing the smaller tactics memory of the sorting program……... 141
Figure. 6.19: Time values recorded for sorting lists (Same List) before and after organizing the smaller tactics memory of the Sorting program………… 145
Figure. 6.20: Comparison when (a) one new element was available (b) two new elements were available (c) Five new elements were available…………. 149
Figure. 6.21: Percentage of new Elements in each list, above which showed better performance on SSPM compared to the original……………………….. 150
Figure. 6.22: How speedup of SSPM insertion varies depending on the percentage of new elements within a list………………………………………. 153
Figure. 6.23: (a) Graph showing speed up ratio by using parallel quicksort. (b) Speedup of the SSPM Sorting when one new element available (c) Speedup of the SSPM Sorting when all are new elements (d) Speedup of the SSPM Sorting when half of the demands are new elements (e) Speedup of the SSPM Sorting when all elements are equal…………………………………………… 156

Figure. 6.24: Speedup ratio, when 2M elements are there in the list (Server)… 159
Figure. 6.25: Speedup ratio, when 2M elements are there in the list (Laptop)... 159
Figure. 6.26: Transition Diagram…………………………………………… 162
Figure. A.1: Source Code Segment for Process Switcher……………………... 194
Figure. A.2: (a) Operation Organizer in SSPM-FC with respect to "Same Op", "Same Exp", "Different Op", and "New Op" (b) Part of the Operation Organizer in SSPM-Sorting………………………………………………. 198
Figure. A.3: Input-Content Analyser of SSPM-FC……………………………. 202
Figure. A.4: Smaller Compiler………………………………………………… 203
Figure. A.5: Write Engine…………………………………………………….. 205
Figure. A.6: Terminating Point………………………………………………… 206
Figure. A.7: Two Multiplication Methods; MulCalc1(), and MulCalc2()…….. 207

List of Tables

Table 2.1: Advantages and disadvantages of Processing models in OS……. 22
Table 2.2: Advantages and disadvantages of computing models……………. 32
Table 4.1: CRs of BTM in Designing the Actions of the proposed Computing Model………………………………………………………… 66
Table 4.2: Exploiting CRs of BTM in deriving the transitions……………… 68
Table 6.1: Process execution with FCFS basis…………………………..... 101
Table 6.2: Process execution with SJF basis………………………………… 102
Table 6.3: Results after Selection…………………………………..…... 102
Table 6.4: Values From Grubbs' Test…………………………………..…. 109
Table 6.5: Values From Power Test for Paired-T Test …………………….. 110
Table 6.6: Values for The Paired-T Test (Addition) ……………………... 111
Table 6.7: Values From Grubbs' Test……………………………………..... 115
Table 6.8: Values from Power Test for Paired-T Test with sample size……. 116
Table 6.9: Values for The Paired-T Test (Subtraction)…………………....... 117
Table 6.10: Values From Grubb's Test……..……………………………….. 121
Table 6.11: Values from Power Test for Paired-T Test with sample size…... 123
Table 6.12: Values for The Paired-T Test (Multiplication)…..…………….. 123
Table 6.13: Selection Process Vs Inefficient Algorithm……………………. 126
Table 6.14. Sample Total Values for Selection Process and Inefficient Algorithm (M2)……………………………………………………........ 127
Table 6.15: Values from Grubbs' Test………………………………………. 131
Table 6.16: Values from Power Test for Paired-T Test with sample size....... 133
Table 6.17: Values for The Paired-T Test (Total Time Values)…………….. 133
Table 6.18: Summary of Results (FC)……………………………………… 134
Table 6.19: Values for The Paired-T Test QES…….……………………….. 137
Table 6.20: Values for The Paired-T Test (InsCalcModule)…………….…... 143
Table 6.21: Values for The Paired-T Test (EquSortModule) ……………….. 146
Table 6.22: Comparison Tables (a) Average run times for different thresholds and number of elements for parallel QS, (b) Relevantly tested SSPM, sorting list results with original QS, when there are 1, half and all new, all equal elements than/to previous list in SSPM….………….……... 154
Table 6.23: Quicksort with Self-adjusting computing Vs SSPM sorting. 157
Table 6.24: Quicksort with ADAPTON with incremental computing Vs SSPM sorting……………………………………………………………… 158
Table 6.25: Comparisons of SSPM sorting with DTL and GAA sorting.……………………………………………………................ 160
Table 6.26: Transition Table ……………………………………………….. 164
Table 6.27: Configurations of the NTM (p(n)=number of moves)………….. 166
Table 7.1: Comparison with Existing Processing Models…………………...... 178
Table 7.2: Transition-wise comparison of the proposed model with existing OS processing models……………………………………………………….. 179

List of Abbreviations

ADAPTON	- Compassable, Demand Driven Incremental Computing
BT	- Buddhist Theory
BTM	- Buddhist Theory of Mind
CDDG	- Concurrent Dynamic Dependency Graph
CAS	- Column Address Signal
CL	- CAS Latency
CBRAM	- Conductive Bridge Random Access Memory
CPU	- Central Processing Unit
CR	- Causal Relation
CUDA	- Compute Unified Device Architecture
DDG	- Dynamic Dependency Graph
DRAM	- Dynamic Random Access Memory
EC	- Evolutionary Computing
EPROM	- Erasable Programmable Read Only Memory
FC	- Fraction Calculator
FCFS	- First Come First Serve
GA	- Genetic Algorithm
GPGPU	- General-Purpose computing on Graphics Processing Unit
GPU	- Graphics Processing Units
I/O	- Input/ Output
IC	- Incremental Computing
IDE	- Integrated Development Environment
IPC	- Inter Process Communications
ITS	- Throughput Target Scheme
LSTM	- Long Short Term Memory
LTM	- Long Term Memory
MAS	- Multi Agent Systems
MPI	- Misses Per Instruction

NMR	- Nuclear Magnetic Resonance
NTM	- Non deterministic Turing Machine
OS	- Operating System
PCM	- Phase Change Memory
PS	- Process Scheduler
QES	- Quadratic Equation Solver
RAM	- Random Access Memory
ROM	- Read Only Memory
RRAM	- Resistive Random Access Memory
SAC	- Self-Adjusting Computing
SAIL	- Self-Adjusting Imperative Language
SQUID	- Superconducting Quantum Interference Device
SSPM	- Six-State Continuous Processing Model
STM	- Short Term Memory
STT-MRAM	- Spin-Transfer-Torque-Magnetic Random Access Memory
TM	- Turing Machine
VNA	- Von Neumann Architecture
WEIS	- Instruction weighted speedup Targeted Scheme

CHAPTER 01
INTRODUCTION

1.1 Prolegomena

The processing power of a computer has been immensely relying on its software and hardware architectures. Yet, many well-established computers are based on the Von-Neumann Architecture (VNA), where the memory is separated from processing. Further, the interest and the requirement on researching into new approaches for powerful computing have been more prevalent in the field. In this background, it was noticed that with compared to the VNA, the human mind displays a different model of computation according to the Buddhist Theory of Mind (BTM). In fact, the most cited natural computing model is human mind. In contrast to VNA, the sophisticated human mind becomes competent in providing quality and fast solutions over subsequent processing cycles. Moreover, the major cause behind this difference is how the memory is formed in the human mind. According to the BTM it is a continuous process. Thus, the work appears in this thesis was started with discussing the hardware level improvements that have been introduced to enhance the computing power. Next, it was critically reviewed a set of important memory and processing models and some other relevant factors that were used to improve computing in the software level. Finally, the new computing model was introduced imitating the continuous processing model in the human mind based on BTM to enhance the computational efficiency of the computer.

This chapter precisely describes the research that introduce the new computing model, in which the memory is modelled as a sequel of a continuous process. The new model was named as Six-state Continuous Processing Model (SSPM). First, this discusses the aims and objectives of the work, while the subsection 1.3 explains about the background and motivation. Then, the inquisitiveness that provided the birth for this research work is presented in the subsection 1.4. Whereas, the subsection 1.5 briefly analyses memory-processing models of current computing models towards the purpose, the subsection 1.6 states the proposed model with the new approach, which exploited BTM. The subsection

1.7 briefly explains the testing and evaluation process of this research. The subsection 1.8, mentions the resource requirements and subsection 1.9 presents the structure of the thesis. Next, the significance of the research work is briefly reported in the subsection 1.10. Finally, the last section summarizes the chapter one.

1.2 Aims and Objectives

The aim of this research is to introduce the new computing model, in which the memory is modeled as conditional phenomena for a new theory of computing that improves the computational power of the computer, with the inspirations received from how the human mind works according to the BTM and the observed nature of the human mind.

In order to achieve above aim, it was set below mentioned major objectives for the research:

- Critical study of various memory and processing models.
- Analytical study about the BTM.
- Propose a new computing model.
- Simulate the proposed model for computing.
- Evaluate the proposed computing model.

The next section briefly discusses about the contextual and the motivating factors behind this research in reaching towards the above objectives, further achieving the aim.

1.3 Background and Motivation

In VNA, the memory and the processing have been separated [1] based on the Charles Babbage's historic mill and store concept with Loom's weaving [2]. Therefore, the communication between the memory and the processor was established through communication busses [3]. However, the delays exist in this communication curtail [1] the processing power of the VNA-based computer due to the discrepancy between the fast processors and the slow memories [4]. Some refer this issue as memory wall [5]. A lot of attempts has been made in order to hit this memory wall [6] and enhance the power of computation. Specifically, the processing units were placed near the memory cells, and cache hit rate [5] was improved by enhancing the data locality [4]. In fact, it has been a live research challenge to enhance the performance of the computer. In doing so, various hardware and software level developments have been introduced. The most of these hardware developments have been based on the memory-processor separation [7].

For example, in hardware level, the processing power was improved by introducing various processing units such as multi-cores [3], Graphics Processing Unit (GPU) [8] and GPGPU [9] with increased processing speeds, memory management and concurrent processing capabilities. Further, the chipmakers have made-up chips with larger density [1] to improve processor speed. In addition to that, various memories such as RAM, Phase Change Memory (PCM) [10], metal oxide memory RRAM [11], DRAM, ROM, Caches [12], and Registers [13] were invented with diverse capacities and introduced into different levels of the memory hierarchy. Certainly, with the introduction of the nanotechnology, the hardware developments will go beyond the Moore's predictions [14].

However, the ability in fully utilizing the introduced processing powers with memory capacities was limited by separation between the processor and the memory due to the data and instruction routing mechanisms [4] and overlaid software architectures. This inadequacy led the researchers to find alternative computing models to make the use of the processing speed acquired by the under-laid hardware architecture completely. Having these motivations behind, the next section declares curiosity, the seed that have sprouted this research. Here, it is more appropriate to name the next subsection as "inquisitiveness in brief" rather than "problem in brief," since this curiosity has provided a considerable motivating force.

1.4 Inquisitiveness in Brief

As introduced, finding a new computing model to improve the processing power of the computer was a great research challenge. When it comes to an alternative approach for computing than of the models where the memory has been separated from processing, the challenge would have been rather significant. This study revealed that, the most of the existing software solutions were mainly focused on providing solutions for the real world problems. These problems are arising from the natural systems with large number of entities that are connected to each other and operated in distributed or parallel manner in the environments that are changing dynamically. However, providing quality solutions more efficiently over subsequent generations of system executions were considered minimally. Those would be rather the modelling of real world systems and the focus was sometimes bit deviated from enhancing the computational efficiency.

It is an important fact that, the most cited best computing model in the nature, the human mind [15], has been working in a way that is different from the VNA. According to the BTM, the memory is not a unit distinct from processing [16]. It has been an evolving result of conditional phenomena [17] and compiles as a smaller tactics memory [18] through continuous processing. In fact, VNA shows no change, though the same program has been executed for multiple times. As opposed to the VNA, the human mind as observed in the real-world has been capable in providing solutions for the same problem or executing the same program with improved efficiency through the cognitive process with the knowledge and the skills gained [16]. Although, it could find the evidences of computer modelling of human mind based on BTM in [19], [20], [15], and [21], it has been difficult to find any evidence on a computing model, which imitates the human mind based on BTM to improve computational efficiency. This has happened to be an omission within the field of computing, because as mentioned earlier, the human mind is the best computer in the natural surroundings. Therefore, the research reported in this thesis is a result of the effort in dealing with introducing a new computing model that exhibits the aforementioned features of the human mind by exploiting the BTM.

The next section briefly discussed the memory and the processing in the current computing models based on VNA, but were set with different features.

1.5 Memory and Processing in Current Computing Models

When reviewing the software level memory management and processing, it has been divided the review into two sub categories such as memory management and processing models in Operating Systems (OSs), and memory-processing in other computing models such as Incremental Computing.

In OSs, from the two-state model to the seven-state model, the processing models were steadily developed. The two-state system had the states ‘not running’ and ‘running’. Then the two state model was updated with the blocked and ready queues to keep the blocked and ready processes separately [22] replacing the ‘non-running’ state ‘blocked’ and ‘ready’ states. With these states the Three-State model was introduced [23]. Afterwards, for the purpose of management of data, ‘exit’ and ‘new’ states were introduced. The ‘exit’ state handles information 'after the fulfillment of the execution' and the ‘new’ state handles the information 'before stacking the recently characterized processes into the memory'. Such a way, the Five-state model was presented [22]. Next, it was introduced the Seven-state model to reduce idling of the processor.

Further, Parallel Computing [24], Neural Computing [25], Multi-Agent Systems [26], Quantum Computing [27], Evolutionary Computing [28], and Incremental Computing [29] can be highlighted as the major computing models, which have been introduced to enhance the processing capabilities of the computer in different ways. Many of these computing models have been introduced on the improved VNA and imitated natural models [30]. For example, Multi-Agent systems have copied the behavior of natural swarms [26], while Evolutionary Computing (EC) model imitated the natural selection. Specially the Genetic Algorithm [31] is largely applicable in optimization problems, search problems, and provide solutions by altering associations [33] counting on biologically inspired operators. Furthermore, the neural computing has been inspired from the human brain [33]. Therefore, it is evident that imitating natural models [34] is a good trend in designing new computing models as the natural models [35] surprisingly equipped with algorithms, operators and other mechanisms to accurately and efficiently solve natural complex problems [34].

Moreover, this research has narrowed down its literature review to analyze the memory and processing in computing models such as Incremental Computing, Genetic Programming, and Multi-Agent systems. Multi Agent Systems (MAS) has been involved in problem solving by sending messages among a group of agents [26] having inspirations from the behavior of natural complex organizations [34] such as ant colonies, fish schools, and bee colonies. MAS offered a novel model for distributed and parallel computing on VNA and can yield emergent solutions [36]. However, when certain applications of MAS are dealing with large groups of agents working together in the same stage, the efficiency improvement in such system cannot be expected [34]. In fact, the concepts such as logical agents [37], long short term memories and reinforcement learning [38], were quite impressive as many of those had the insight from the human mind according to the theories introduced in Bartlett’s Remembering [39], constructive memory [40], and Atkinson-Shiffrin [41] and the Baddeley [42] models with a western philosophical view. Meanwhile, the foundation of the evolutionary computing was laid by the Genetic Algorithms (GA), having inspirations from the Darwinian theory of evolution [43]. There are many aspects in the field of computing that are benefited from GA. For example, for CPU scheduling GA has been applied [44] in order to maximize CPU throughput or utilization [45] or optimize the waiting time [46]. Further, over generations of executions, the GA can produce better quality solutions, although the GA consumed memory and CPU in a considerable level. The Incremental computing was an approach in modelling systems with the incremental and dynamic slight changes in input data [47]. There, the memory management [48] was done through the graphs and memorization [49]. Self-adjusting computing [50] was one of the branches in incremental computing. Further, there were different adaptive algorithms that have been applied in order to speed up the computing [51]. In some cases, it has also been used different program transformation [52] techniques to enable adaptability and achieve speeding up. Further, it has been discussed memory and processing in parallel computing and neural computing also.

With this background, the next section describes the proposed computing model that is inspired from the BTM and other related real-world examples to enhance the computing power of the computer.

1.6 The Proposed Computing Model

This research was conducted to discover a new computing model to enhance the processing power of the computer, where the characteristics of the human mind was incarnated into a new processing model exploiting the BTM. According to the BTM, the human mind undergoes a continuous flow of thoughts [53]. The continuity of this processing is maintained by several factors such as the inputs received through physical five sense doors (external inputs), the inputs internally generate in the mind door (internal inputs), and a set of causal relations from twenty-four causal relations [54] explained in BTM. All the time, the internal inputs are generated in relation to and are affected by the prior external or internal inputs. In addition to that, the repeated processing on the same set of inputs, improve the speed, quality and the accuracy of processing [55] as the processing is not separated from the memory. The memory is a result of continuous processing that arise as per the conditions. Further, starting from an initial setup, the smaller tactics memory has gradually improved and organized through this continuous processing or practice. The knowledge and instruction entities entered in as any sort of inputs or instructions, are labeled, in the way, which one can identify, describe, relate or retrieve back the knowledge entities and the results of relevant computations. Moreover, a set of tactics such as pattern identification, classification, and prioritization has been used.

The set of tactics have been derived from a set of fifteen causal relations, namely, Object ($\bar{A}rammana$), Root ($H\bar{e}tu$), Co-Nascence ($Sahaj\bar{a}ta$), Association ($Sampayuktha$), Mutuality ($A\tilde{n}\tilde{n}ama\tilde{n}\tilde{n}a$), Pre-Dominance ($Adhipati$), Presence ($Atthi$), Support ($Nissaya$), Pre-Nascence ($Pur\bar{e}j\bar{a}ta$), Proximity ($Anantara$), Karma ($Karma$), Repetition ($\bar{A}s\bar{e}vana$), Disappearance ($Vigata$), Post-Nascence ($Pacchaj\bar{a}ta$), and Karma-Result ($Karma - Vipaka$) from twenty-four causal relations in BTM [16]. This was the set that has explained the process in the human mind better. This has been explained in detailed in the chapter four.

An inspirational example that has displayed the nature of the human mind is discussed next. Let’s think about the two cases, where a student, and a senior professor who are preparing for and do their presentations. When, the student does the presentation, in most of the cases, he needs external aids such as PowerPoint slides to drive through his own knowledgebase. Through series of refinements, he can well organize his slides using set of tactics and improve his own ability to do the presentation accessing his knowledgebase. In the case of a senior professor, he has such a well-organized smaller tactics memory, which enables him to clearly conduct his presentation accessing his larger knowledgebase. This ability and the smaller tactics memory has been improved throughout the years. All such skilled workers do in the same way. Therefore, one can believe the existence of a smaller tactics memory in the human mind. This smaller tactics memory gradually updates through continuous processing, and allows access to the large knowledgebase, is a part of processing, and improves the processing back. Again, it is obvious that this smaller tactics memory has been different from the smaller memories of the current computer such as caches or registers [18]. Through this continuous processing, human can improve the processing power, accuracy and the quality of the work they do. Then, this would be a new approach for computing to improve computational efficiency.

This processing model can improve the processing power, quality, and accuracy of the computation done by the computer with the support of an evolving smaller tactics memory, which is a result of continuous processing.

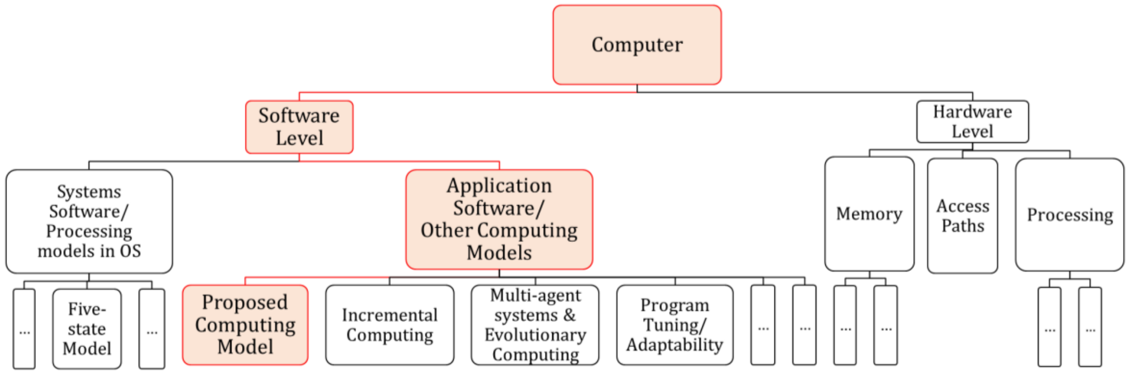

Computer
Software Level
Hardware Level
Systems Software/ Processing models in OS
Application Software/ Other Computing Models
Memory
Access Paths
Processing
Five-state Model
Proposed Computing Model
Incremental Computing
Multi-agent systems & Evolutionary Computing
Program Tuning/ Adaptability

Figure. 1.1: Where the proposed model is placed

In software level, it is an evident fact that the efficiency of processing hugely affected by the actions and the corresponding states in the process flow. After critically studying memory and processing models as mentioned earlier, this research has introduced SSPM [16] to the software level as seen in the Figure. 1.1 to produce the conditionally evolving smaller tactics memory. The actions that are the constituent of the transitions of the new processing model were formed by utilizing the above mentioned fifteen causal relations.

Then, “New,” “Ready,” “Running,” “Blocked,” “Sleep,” and “Terminate” are the processing states of SSPM [16]. Here, the ‘Terminate’ is a new state and an operation reaches this state if there is neither more modifications apply on that operation nor related inputs come in. Then, instead of ‘Exit’ state in the five-state processing model, here is the ‘Sleep’ state. After finishing a particular execution, an operation reaches the ‘Sleep’ state, but it is not removed out from the system, and used for the organizing the system further. In addition to that, the ‘Ready’ state of the new model is more complex than the ‘Ready’ state of five-state model as it involves in organizing. This organizing is an internal process, which includes classification, prioritization, and deletion. Through this, the system and the smaller tactics memory gains improvements. Accordingly, the New to

Ready is a modified transition, while the Ready to Ready, Sleep to Ready, Running to Sleep, and Ready to Terminate are new transitions. The six-state continuous processing model (SSPM) has three characteristics, which can be used to distinguish it from other existing computing models. Those are the internal and external processes, continuous processing, and conditionally evolving memory.

This model with the above characteristics and the actions have been incorporated into a Fraction Calculator (SSPM-FC) [16], quicksort algorithm (SSPM Sorting – particularly SSPM Insertion), quadratic equation solver (SSPM-QES), and in a simulated process scheduling program (SSPM-PS). However, it has been done a great work on FC as it has matched better with these conditions and circumstances of the proposed model than the other systems. SSPM Sorting allowed to compare the model with existing computing models. This SSPM-FC solves equations of fractions with the operators +, -, *, and /. How the three characteristics have been implanted in the SSPM-FC, is briefly mentioned below.

Internal and External Processes - External Processes are initiated due to the user inputs (fractional equations inserted by the user). Next, the internal processes are initiated due to the system generated fractional equations in the absence of user inputs (external inputs). Meanwhile, modification-related actions (tactics) that are applied to the system would belong to the internal process category.

Continuous Processing - As a result of the internal and external process, there is no lapse between two processes. Hence, it establishes the continuity of the system.

Conditionally Evolving Smaller Tactics Memory - For the simulation purpose a text file has been used as the smaller tactics memory, through which can call instructions as per the queries. Due to the above mentioned inputs, and relevant internal and external processes including modification processes, this smaller tactics memory get improved up to a certain level over program execution cycles.

It is an important fact to remind that the entire processing model in the above mentioned programs are managed through a smaller tactics memory with a set of tactics. Finally, all these were tested with many examples and evaluated. The next section has briefly mentioned the respective testing and evaluation scenarios that has been taken place.

1.7 Testing and Evaluation

The SSPM-FC, SSPM Insertion, and SSPM-QES, in which the new model is incorporated, has been executed for many rounds with sets of inputs. Meanwhile, the time taken for the computation of each input has been recorded in nanoseconds. All the cases were tested to check whether an improvement has gained by creating modules for frequent operations over program execution cycles. The time values were collected for the execution of each equation in the same set of equation before and after do the modification. Then, the paired samples of time values were statistically analyzed with the paired-t-test after checking the samples for the applicability of the test in the samples.

Finally, with the SSPM-FC, SSPM-QES, and SSPM-S-Equal, it could prove that the system gain improvement over generations of program executions by generating modules for frequent operations. There was a second scenario for the SSPM-FC that was to test the ability to enhance efficiency by deleting inefficient or unnecessary items. The testing process was conducted similar to the first scenario. Here also, it could prove that the system gains improvement over the original case. However, with regard to the SSPM-S-Insertion case, there were some limiting factors that affects performance such as number of elements in the lists and standard deviation. Further, both the SSPM-Sorting cases were compared with other quicksort programs developed with different computing models such as incremental computing, self- adjusting computing, genetic algorithmic approach, parallel computing, and dynamically tuned library for sorting. Respective results are reported in the chapter six. Moreover, the model has been theoretically simulated in a Non-deterministic Turing machine (NTM). Afterwards, a Boolean expression $E_{T,w}$ has been derived from the transitions and proved the satisfiability and NP-Completeness using Cook's theorem [56]. Therefore, it could prove the appropriateness for the real-world in theoretical level.

1.8 Resource Requirements

In implementing the proposed computing model in a SSPM-FC, SSPM-Sorting, and SSPM-QES, NetBeans 8.1 has been used with Java 1.7, MySQL 5.1, Notepad++. Further, the statistical analyses have been conducted using Minitab 17 and Minitab 18, with the support of EXCEL. In addition to those, the model has also been incorporated in a simulated process scheduling system using Turbo C. Moreover, JFLAP has been used to draw the transition diagram of the model simulated in a Turing Machine.

The SSPM sorting was tested in both Intel(R) Xeon(R) CPU ES-2623 V3 @ 3.00 GHz with Turbo Boost up to 2.0GHz with 16GB cache size, 64B cache line size and 4 registers in SUSE Linux (Server), and Intel(R) Core i7-8550U 1.8GHz (2GB VRAM, 8 GB DDR3 L Memory, and 1000GB Hard Disk) with Turbo Boost up to 4.0GHz with 4608MB cache size, 64B cache line size and 8 registers in Windows 10 operating system (Laptop). Moreover, initially, all the cases were developed, experimented and statistically analysed using the above mentioned laptop.

1.9 Contribution to the Field of Computer Science

The model SSPM has the ability to improve the speed and the quality of program execution in subsequent program execution cycles due to the conditionally evolving smaller tactics memory through continuous processing. Therefore, SSPM is most applicable to enhance the processing speed and quality of continuously processing systems such as the systems which require continuous developments or monitoring. Further, the model can be implemented in both the hardware and software levels.

Ultimately, the SSPM where the memory is modelled as conditional phenomena, will make a paradigm shift in the field of computer science as it provides an alternative approach for computing than the VNA.

1.10 Organization of the Thesis

Meanwhile, first chapter has given the introduction for the thesis; the rest of this thesis is structured as follows. The chapter two critically describes the existing computing models that have been applied to the current computer to enhance performance in computing. Then, the chapter three discusses the theoretical framework provided by the Buddhist Theory of Mind, and other real-world inspirations for the work appeared in this thesis. Further, it mentions some existing memory models and the philosophical approach of those. Next, the chapter four presents the novel approach towards introducing the new computing model, which imitate the human mind to enhance the computational efficiency. Later it has discussed how the Buddhist Theory of Mind has been exploited in modelling new continuous processing model, while the implementation details of the proposed computing model in different prototypes has been mentioned at the end of this chapter. The chapter five explains how the systems work. The next chapter reports the testing and evaluation process, and respective results of the empirical evaluations and the formal verification. Finally, the last chapter concludes the work, discussing the results of formal verification and the experiments, applicability of the proposed model in the real-world, limitations and its future.

1.11 Summary

This chapter presented the aims and objectives of this research that has introduced a continuous processing model for computing. Then, it has explained the background, the motivation, and the based curiosity for this work, emphasizing the significance of the work. Further, this has concisely reviewed the current computing models, while briefly explaining the proposed computing model, implementation, testing and evaluation with the philosophical, theoretical and real-world inspirations for the model. The resource requirements were also mentioned in the latter part of the chapter. Finally, it has presented the order of the chapters of the thesis. The chapter two examines the existing computing models in hardware level and software level in deep, but in a more focused manner.

CHAPTER 02
REVIEW OF MEMORY AND PROCESSING MODELS

2.1 Introduction

The previous chapter briefly described the entire research work reported in this thesis. It initially started with mentioning the aims and objectives of the research. Next, it has briefly discussed the background and the motivation with the utmost curiosity. Then, relevant software techniques of memory and processing in current computing models has been discussed. After that, the proposed model SSPM was described with the new approach, conceptual aids, other inspirations, testing, evaluation, and resources requirements in short. This chapter review different approaches for the memory and processing those were applied in solving natural problems efficiently and improving performance of computing. As introduced in the last chapter, these approaches were two fold. One is hardware level and the other is software level. In hardware level, improvements were introduced separately to the memory and the processing. Further, the connection between the memory and processing was also improved using different technologies. The memory and processing models in the incremental computing, self-adjusting computing, parallel computing, multi agent technology, evolutionary computing, neural computing, quantum computing, and dynamically tuning with algorithmic choice in VNA are discussed under the software level approaches together with the processing models in operating systems.

2.2 Hardware Improvements

The hardware level improved in different ways to expand the power of processing in the computer. Those improvements are based on the memory and processor separation [3]. In the Memory side, Static Random Access Memory (SRAM), and Dynamic Random Access Memory (DRAM) carried a huge burden in enhancing the computational performance as the memory lead the ways to enhance computing power [57]. Among others, the spin-transfer-torque magnetic random access memory (STT-MRAM) [58], PCM [10], Conductive Bridge Random Access Memory (CBRAM), and RRAM contributes for a substantial performance improvement in the computing [57]. Further, to reduce the memory-processor gap, different fast memories such as caches (L1, L2, L3, [59] and L4 [60]) and registers were introduced and those act as a temporary storage in between the processor and the RAM, keeping the frequently or recently requested data to enhance the memory access. Further, internal caches and registers have been located on the processor chip, possibly making the quick access from the processor itself. Further, the communication busses were introduced with increased bandwidth [1], [12] and shortened path lengths [1] to increase data transferring ability. Moreover, researchers tried to introduce diverse communication patterns [61] as well. In one way, the computing performance was enhanced by applying different heat spreading or cooling techniques on RAM [62]. In another way, RAMs were introduced by lowering the Column Address Signal (CAS) Latency (CL) [62], where the CL referred to number of clock cycles taken to access the information stored in a single column [63]. However, in some high performance RAMs like DDR4, the CL is higher than DDR3 RAMs due to the high storage density of the DDR4 RAM [63]. Another mechanism in enhancing the RAM performance was to maximize the sub-word operations and minimize the data transfer between registers and memory [7]. Meanwhile, processing in memories [64] and three dimensional (3D) memories [57] has become a trend in enhancing the computational efficiency [4]. Those can reduce the unnecessary data transferring between the memory and the processor [64]. Further, the near-DRAM acceleration architecture [65] is such a new attempt to reduce data movement by bringing the processing near to the memory with 3D stacking. Despite this closer integration, the separation between the processing and memory is still remains [58]. With the time, researchers started to think in fading this separation line between the processing and memory [4] imitating the human brain. So far, the researchers have done their best to reduce the distance between the processor and the memory to enhance the performance of computing. Hence, it was apparent that there exists a separation between the memory and the processing, and it is different from how the human execute instructions in his mind.

Some other, hardware level improvement is the use of parallel computing. To accelerate processing in the computer, multiple cores were introduced [66]. Further, it has been introduced dedicated programmable logic chip, which is called as the Graphical Processing Unit (GPU) [67] to work with images, videos and animations efficiently. Generalizing the GPU, it has been introduced general-purpose computing on graphical processing unit (GPGPU) [68] to improve performance of not only computing graphics, but also the entire computing system [67]. In multi-threaded architectures, the bandwidth has become a severe bottleneck [9] even in GPGPU, and co-scheduled applications significantly reduce the overall performance [8]. Further, the strength of the memory was suggested to be determined by last-level cache misses per instruction [69]. However, the performance of computing was not only dependent on the misses per instruction (MPI), but also depend on the bandwidth achieved by the memory [8]. However, some researchers proposed in using only one memory with a one level cache [70] for the GPUs. An interesting application-aware memory scheduler with high performance for GPU has been introduced in [8]. In doing so, they have introduced two memory scheduling approaches, namely, Instruction weighted speedup Targeted Scheme (WEIS), and Throughput Target Scheme (ITS). In the first case, applications with lower MPIs are prioritized. For that, they have recorded the number of instructions executed. In the second case, the applications with lower bandwidths were prioritized. Here, the requests were served picking up the oldest requests from the highest priority applications. Then, improving the GPU architecture further, the CUDA architecture has been introduced allowing the speed up of the tasks with severe computing loads. Parallel computing is a technique that can achieve performance improvement through parallel processing. With the parallel computing, it was possible to execute an application on many processors [24] speeding up the processing. This parallelism can be achieved in different levels such as data, instruction, process, and thread [71]. In addition to that, the threads or parallel algorithms have been used to enhance the performance of programs [72]. For example, Quick sort algorithm was improved such a way. However, major focus of this research was not under the parallel computing concern.

In every year processing speed is improved according to the Moore's law [14]. Uninterruptedly, these hardware developments will cross the boundaries of the Moore's estimate through the move of nanotechnology into the production of computers. However, the memory arrangements and access mechanisms are still considerably slower than the processor [71]. Further, the distinction between the memory and processing is clearly visible. This is one of the major obstacle in gaining the performance in the computer.

Then, the other obstacle in gaining the performance of the computer is insufficient software level improvements to cope with underline architecture and their own separated memory and processing models. The coming section has been allocated to discuss on memory and processing techniques and respective modifications of some different computing models.

2.3 Improvements for Memory and Processing in Software Level

The researchers in different fields of computing tried to address memory-processing issues and improve computational efficiency in the inherent ways. The study of these different approaches provided a sound technological knowledge and different aspects to be concerned about, when researching for a newer approach for computing. Therefore, under this section it has been discussed about different computing models such as processing models in operating systems, incremental computing, multi-agent technology, evolutionary computing with adaptations, and neural computing.

2.3.1 Processing Models in Operating Systems

As an aid for designing a processing model and understand actions and states, operating systems' (OSs') processing models has also been studied. Further, those models provide an idea of the clear-cut difference and the particular relations between the actions and their respective states. Moreover, the knowledge of the evolution of the OSs' can be applicable not only to introduce newer processing models to OSs, but also it provides a greater insight into introducing processing models for different contexts or for entire computing system.

2.3.1.1 Two-State Model

This model consists of two processing states as Running and Not-Running and are determined such a way that the process is being executed by the processor or not. Further, there is a single first-in-first-out queue to hold processes, which are not running. This queuing discipline is good if all the processes in the queue are ready to execute[22]. However, sometimes the queue consists of both the ready and the blocked processes. Then the dispatcher, which switches the processor from one process to another, must find not only the first-in process, but also the process, which is ready, from the queue. Thus, two queues such as ready and blocked were introduced to the processing model by splitting the above queue to solve this issue, hence introducing the three-state model.

2.3.1.2 Three-State Model

In the three-state model, the processes in the ready queue are in Ready state, the processes in the blocked queue are in Blocked state and the executing processes are on the Running state. As such, there exist three states [23]. Those states are the most important states and are available in each and every model which are discussed hereafter. However, in this model, there was no mechanism to store information related to the states after end of the execution of processes and before load the newly defined process into the memory (activated processes). Next, the researchers introduced five-state model including these missing details.

2.3.1.3 Five-State Model

This model has New, Ready, Running, Blocked and Exit states [22]. Here, they have introduced two new states New and Exit to the above three-state model. A process is in New state if it is not added to the group of executable processes and not loaded into the main memory, whereas a process can be in Exit state if it is released from the above group due to a termination for a particular reason.

These two states are useful in managing the processes, as with the New state, it is possible to limit the processes available in the main memory avoiding the main memory limitations, and as the details of released processes which are in Exit state, are unnecessary, then those can be removed freeing the memory.

However, this model produce processor idling. Therefore, as a solution Seven-state model has been introduced.

2.3.1.4 Seven-State Model

Let us consider the scenario, where it was required to load the process to be executed next with the highest priority. But, if the main memory was completely filled by loading blocked processes and no virtual memory exist, then, until occur the waiting event, the processor will be idling. Due to this idling, The Input/Output (I/O) activities were highly affected and delays occurred in the memory access. To overcome this issue, the two Suspend states [22] such as; Ready/Suspend and Blocked/Suspend were introduced and the model was named as Seven-state model. With this model it was able to keep highest priority processes in the main memory and send lower priority processes into a secondary storage. This swapping slightly enhanced the performance of the system, as disk I/O is normally quicker than the rest of the I/O. However, the I/O activities and the delays exist in memory access are still required to be solved.

2.3.1.5 Processing Model in Linux OS

Since, the Linux operating systems also has a virtual memory [73], no suspended states were used. Although, this consists of the three principal states, the Blocked state is splitting into two states such as Uninterruptible, and Interruptible [22] depending on different conditions. Further, there is another state called Stopped, similar to the Blocked states, however, can be resume only by a positive response received from a different process. Finally, this model has another state with the name Zombie [22], which is similar to the Exit state, but keep the task structure in the table of processes. The child processes are in Zombie state if those died before its parent process dies. Otherwise, the parent processes cannot be survived. Once the parent process dies, the stored task structures of relevant Zombie child processes are also deleted. Such a way the Linux OS consists of five major processing states.

2.3.1.6 Kernel Thread Model of Windows OS

The states of the thread model of Windows OS is categorized into two categories such as Runnable and Not runnable, where ‘Ready’, ‘Standby’ and ‘Running’ are Runnable states and ‘Transition’, ‘Waiting’ and ‘Terminated’ are Not Runnable states. Here, the two special states are ‘Transition’ and ‘Standby’. In fact, the processes in ‘waiting’ state in traditional processing model has been divided into ‘Waiting’ and ‘Transition’ states. Further, a thread is in ‘Transition’ state, if the thread is ready to run itself, but is lack of the resources such as space [22]. In addition to that, the traditional ‘Ready’ state is separated into ‘Ready’ and ‘Standby’ states, where the thread to be executed next with the highest priority is in the ‘Standby’ [22] state. Therefore, this model has six states.

2.3.1.7 Kernel Thread Model of Solaris

In this Model, the terms used for the states are slightly different from the traditional terms. For example, instead of ‘Ready’ state, this has ‘Run’ state, then for ‘Running’, this has ‘OnProc’, and for ‘Waiting’ or ‘Blocked’ this has ‘Sleep’. Further, ‘Stop’ is also a ‘Blocked’ state, where the threads, for example, which are stopped for debugging, can be in ‘Stop’ state. However, the ‘Zombie’ state of this is similar to the ‘Zombie’ state of Linux. Additionally, this model has a different state called ‘Free’, in which the threads stay if they have finished execution, released the allocated resources and ready to be removed [22].

In previously discussed models, the processes in the states that are related to the ready/ blocked and running are activated and stored in the memory. Rest of the processes are stored in the secondary storage. Further, it could be summed up that the memory management ability has been evolved from one OSs' model to its next improved state. These memory management capabilities can be used to manage and support processing. The following Table 2.1 discusses the advantages and disadvantages of the processing models of operating systems.

Table 2.1: Advantages and disadvantages of Processing models in OS.

Processing Model	**Advantages**	**Disadvantages**
Two-state [22]	States are determined such a way that the process is being executed by the processor or not.	Overhead for dispatcher to check not only the first-in process, but also the ready and blocked processes.
Three-state [23]	Separate queues to store blocked and ready processes.	No mechanism to store information after the end of processes and before activate the processes
Five-state [22]	New states New and Exit to store information before activate and after ending the process limit the processes available in the main memory	Produce processor idling: Delays occur in input/ output activities and memory access
Seven-state [22]	Keep highest priority processes in main memory and send lower priority processes into a secondary storage. Disk I/O is comparatively quicker.	Delays exist in memory access are still needed to be solved.

Linux, Windows Thread Model, Solaris Kernel Thread Model	Process and memory management capabilities were improved	Neither the processing model nor its access mechanism nor functionality was improved

However, it was visible that neither the processing model nor its access mechanism nor functionality was improved during the operating systems' orchestration of hardware, software, and program execution.

Next section discusses on an improvement gaining computing model. That is Incremental Computing.

2.3.2 Incremental Computing

Through, the incremental computing, the efficiency of computation can be enhanced, when the computation is recursive or iterative and acts on the inputs, which differs from each other only to a certain extent. In fact, the authors of [52] has categorized the attempts on incremental computing into three categories, which includes incremental algorithms, incremental execution framework, and incremental-program derivation approaches. Under the above third category, the work appear in [74] has introduced an all-inclusive and formal transformational method, which is equally applicable in all the languages. Further, a prototype system that is called as CACHET [75] has been designed and implemented. It was a program transformations system and was semi-automatic. Here, the incremental program was derived through previously stored result of the program, cached intermediate results and other auxiliary details. With compared to the other program transformation systems, CACHET allowed to use all the functional facilities related to program execution easily.

The incremental computing allowed in efficiently executing programs [29] by only updating the portions of the outcome that are affected by the altered inputs [29], [76].

Moreover, an Incremental Compiler can be considered as an example. Let us think it compiles a certain program and produce an object file. Later, it compiles the same program after adding few more lines to the program. Then, the incremental compiler will execute only on the newly added lines and will update the object file accordingly.

In fact, the incremental computing requires to handle larger number of intermediate results and to cache them, which was quite difficult. To mitigate this a technique called cache-and-prune [77] was introduced, where the cache achieved this statically through program transformations.

There were different branches of incremental computing, such as Self-Adjusting Computing, Imperative Self-Adjusting Computing, Incoop, ADAPTON, and ithreads. Each of the incremental computing mechanism is separately discussed below.

2.3.2.1 Self-Adjusting Computation

Self–adjusting computing (SAC) is an incremental computing technique, in which the computation adapts to the external change of the input data [78]. This technique has been proposed, so to update output incrementally [50] through dynamic dependency graph (DDG) and memorization [49]. Further, these two approaches have been used to identify function calls that are related to the changed input and required to re-execute through change propagation. If the reference modifiers are write-once, function returns no value, and function access non-local computations, then the DDG can be created by tracking memory operations and function calls. However, in SAC data reside in the memory can be high as the memory contains the execution traces. Therefore, the SAC is an overhead for garbage collector GC and for counters. In eliminating this problem, a memory management mechanism [48] for SAC was introduced. This memory management technique has three states such as Free, Live, and Dead. The user can allocate memory using SAC memory allocator and SAC automatically free memory. Finally, the SAC with this memory management technique was compared with other SACs, which resulted that due to this memory management, 10% in timing improvement and 75% of space usage reduction.

2.3.2.2 Imperative Self-Adjusting Computing

Imperative self-adjusting computing is the generalized form of self-adjusting computing, and support for imperative programming [79]. In contrast to self-adjusting computing, this allows to write modifications to modifiable reference for multiple times, therefore no restriction for the writing into the memory. This research has introduced a new consistent language called Self-Adjusting Imperative Language (SAIL). Through this, it was able to trace all the dependences, while reading the content of a modifiable and the expressions bound to particular contents. However, with compared to the purely functional languages, the SAIL is simpler. Without any difficulty, it can write to memory. Further, the logical relations of this carry out location bijections, hence managing the unpredictable effects of memory assignments. It was possible to improve the efficiency of change propagation in the traditional self-adjusting computing.

Next section discusses about another approach of self-adjusting computing, that is Incoop.

2.3.2.3 Incoop: MapReduce for Incremental Computations

Incoop [80] was an all-purpose MapReduce framework for incremental computing and it was a well-tuned, output-reuse, efficient technique that was applicable to respond to slight input changes and update the output automatically. By dropping the chunking of job and doing constant partitioning of inputs Incoop could incrementally enhance the efficiency having the insight from the [50] and CEAL [81]. To further enhance efficiency, the Incoop uses memorization aware scheduling technique, which could control the data passing across the system as the memorization server had an in-memory-key-board store. In addition to, that it could use locality of previously stored results. Moreover, the chunking can be reduced by dividing jobs into smaller jobs without excessively accessing data.

2.3.2.4 ADAPTON – Compassable, Demand-Driven Incremental Computation

The traditional incremental computing suffers from several issues such as, IC is incapable in performing well, when the interactions with the program are unpredictable or when the out-layer tries to orchestrate the inner computation based on dynamic information. In fact, this inner layer does the computations, the results of which, are changed later. Then, the outer layer is the external observers. The ADAPTON [29] was the solution for the above mentioned issues, where it re-computes only the traversals affected by the demands on the outputs triggered by the outer layer. Specifically, it is a library that enables this special incremental computing and uses a demanded computation graph to track changes and perform lazy evaluation. Further, this work has compared the results obtained by executing several sorting algorithms through lazy and eager evaluation, calculating the speedups and measuring the memory usage. Best cases were selected upon the highest speedup and less memory consumption, where the lazy evaluation was the best with ADAPTON in most of the tested cases.

2.3.2.5 iThreads: A Threading Library for Parallel Incremental Computation

This is another application of self-adjusting computing where concurrent dynamic dependency graph (CADDG) has been used to store data and relevant computational dependencies [82]. The computation is divided into sub-computation units and those are reused or recomputed. Further, the change propagation is applied through CDDG and parallel incremental computing. This approach supports shared-memory and multi-threaded programming. In addition to that, iThreads use Release Consistency [83] memory model as the memory model hugely affect the performance. Although, it is an old memory model, this allows asynchronous granularity of memory component at given points on sub-computations reducing the restrictions that can be imposed on memory access with more accuracy and liveness.

2.3.3 Multi-agent Systems and Evolutionary Computing

Multi agents provide the modularity with autonomy. And also, an intelligent software agent serve the environment by achieving the desired objectives [26]. There are two major approaches such as agent-centric and organization-centric, which have their own pros and cons [84]. Moreover, Multi-agent Systems (MASs) can be used in expanding the existing software paradigms [85]. In MAS, sometimes, with compared to the benefits and emergent solutions that one can obtain from MAS, the inefficiencies could be more weighted, as the system has to handle enormous number of agents work together [85]. However, there is an interesting member in the MAS concept. That is the logical agent, which has more similar features to the human. All the time, this exercises with the memory [37], experience [86], and the adaptation or learning. There are different approaches for the memory management in Artificial Intelligence. When, introducing theories for designing memory of logical agents, most of the researches were based on the STM, LTM [41], and working memory [42]. In [87], and [88], the memory is considered as a “reasoning process”, than a mere location in the system. They suggested that the LTM stores the external information, results of agent’s own elaborations of perceptions, experience, and previous internal processing. Specifically, an agent’s memory is strengthening a notion or a process of learning [87]. This proposes a framework for handling the agents’ memory, where the memory is produced, customized, and maintained rendering the current situation. To represent the situation, they have used present events. An agent can agree to change due to external changes as well as its own preferences about how to proceed.

Major concept behind in evolutionary computing is natural evolution [28], which is based on Darwinian evolutionary system (EC) [43]. There, a natural selection occurs in a population with restricted resources, where a competition exist in acquiring resources [28] (the fittest one will survive). Recently, researchers have been applied genetic algorithm (GA) approaches in different fields [32]. For instance, GAs have been applied in OS to optimize the waiting time [46] or enhancing the throughput or CPU utilization

[45]. Further, EC also have been applied in optimizing the programs by adapting best algorithms based on the environmental conditions [89]. In addition to that, those have used ‘crossover’ or ‘mutation’ methods. Moreover, it could find an interesting component that was considered in particle swarm optimization, it is the velocity component [90]. It records the memory of previous particle movement, the performance of each particle is determined and used in determining the best particle and the velocity of the particle for the next move. This computation totally relies on the previous knowledge of the particle. Another case found where the evolutionary computing has been used to improve the life span of the memory devises through several bit detection and correction [91].

Such a way, it could find many applications of evolutionary computing [92] and MAS in solving problems efficiently. Further, it could find a combined approach of Evolutionary Computing and MAS in order to improve the state of a MAS, considering the stability and fitness of agents [93]. In that scenario, the probability of being in a such state can be varied with time and other conditions and therefore solely the fitness value determination was not enough.

However, it was difficult to find cases where these were used in enhancing the power of computing. For example, MAS was used to create evolving knowledgebase [94] to share among team members, though it is not contributing to enhance computational efficiency of the computing system. As well as the MASs with large number of agents, EC also requires more resources, both the memory and processing and those are based on the VNA where the memory is separated from processing.

2.3.4 Neural Computing

If a neural architecture has to work in a dynamically changing environment, then it requires an STM that attach to the corresponding past event and a coordinator to predict or classify those past event [95]. Further, they did a characterization considering the form of the content and their adaptability, and this characterization was done according to the depth and the resolution. Here, the depth is how far the memories back in the past. Then, the resolution is the amount of stored information related to an element of an input sequence. There were some memory models that have been applied in this context. The first one was tapped delay line memory with low depth and high resolution (initially, the memory was implemented through mercury delay lines [96] in VNA). In which, the series of delay lines used to set a buffer to store most recent *n* inputs. Furthermore, this model was extended with non-uniform sampling of past events. The second memory model was, exponential trace memory with high depth and low resolution. Here, the strength of an input dropped down, but not the content. However, there could occur information loss or difficulty in extracting information. The next memory model was Gamma memory. This is a continuum of memory from high resolution-low depth to low resolution-high depth, formed with Gaussian kernel. Such a way, different memory models have been introduced with respect to the most of the kernels, such as exponential and Gaussian. The content of the memory should not necessary to be exact input sequence. The input could be transformed and stored. Further, by adjusting the memory parameters, an adaptive memory can be obtained. Moreover, this adaptive memory was capable of selecting the most relevant portion of input in making predictions. However, there were limitations in predictions were occurred. To improve the accuracy of the neural computing, neural architectures were expanded with many hidden layers, introducing the deep neural networks. When doing so more resources such as memory was required. One approach to overcome this was dynamically allocating GPU memory to train deep neural nets [97]. Further, a processing in memory concept has also been used in this regard to address the memory wall issue arise during neural computing with the use of metal-oxide resistive RAM [98]. Moreover, to reduce deep memory problems in neural computing gated recurrent memory unit with memory block [99] has been introduced further solving the problems between the memory, processing and data management. This write and read data to/ from the specific memory block in a way similar to the neural Turing machine [100] that consists of neural controller and a memory bank. Further, the reading and writing accomplish on a specific part of the memory while ignoring the sparse rest of the memory.

The models such as parallel computing, Neural Computing and Agents are largely relying on distributed processing, since the scientists believed that the performance could be enhanced with such a decentralization.

2.3.5 Program Tuning with Adaptability

With this concept, it could automate the optimization of computer programs so to support in adapting to different environments and to different requirements [101]. This is called as autotuning. Further the choice of the best algorithm depends on the system load, input, and hardware during the runtime [102]. Under this category there was a technique to determine the best algorithm to suit to the inputs [103]. This method tackles the input grouping, and classifier constructions with feature extraction in a different way using two level clustering [103]. With this it could gain, a 3 times speed up more than single configuration for all the inputs. Further, this speedup compared to the single level input analysis was 34 times. In fact, most of the autotuners used in single projects were difficult to use in an another project, as those were running with domain-specific information and requires specific searching techniques. However, an OpenTuner [104] was allowed to build domain specific, but multi-objective autotuners. There was a compiler with extended language [105], which provides a fully automatic autotuning during compile-time and install-time to accomplishing optimizations for variable accuracy algorithms by achieving the target accuracy. Through autotuning, it could apply optimizations for memory hierarchies [101]. Further, it should be noted that, when executing different algorithms, those have different memory access mechanisms for each algorithm. The locality of this memory access also affect the cache utilization as well as the performance. Dynamically tuned sorting library [51] has the adaptability to the sorting techniques such as quicksort and merge-sort depending on the input size, hardware architecture, and standard deviation and the distribution of the input. They have concentrated on the standard deviation, because it matters with the cache line sizes. Furthermore, there is another approach which find the best algorithm through a competition by dividing processing resources among algorithms and executing them in parallel [106], and it is depend on the dynamically changing loads. This model was equipped with an autotuning setup and particular splitting techniques. In most of such tuning cases, the tuning process was done in black-box manner. In contrast, the white-box tuning process [107] has been introduced in order to conduct in stepwise manner independently. Nevertheless, the intermediate results are aggregated together. Further, the results that produce inefficient computation can be eliminated during the process. Moreover, there could find different concerns on applying autotuning to high performance computing [108]. There, they have considered to apply autotuning when the context changes. For example, if the software is installed in a different architecture or the nature of the input [109] is changed as discussed in [51]. For different inputs, system can call libraries accordingly [103].

However, when rising the requirement of the automatic work, it generates overhead on the system resources.
The Table 2.2 shows the advantages and disadvantages of the computing models.

Table 2.2: Advantages and disadvantages of Computing models

Processing Model	**Advantages**	**Disadvantages**

Incremental computing [52] (Update the Portion of output affected by altered input)	Efficiency of computation can be enhanced. Language Independent	Handles larger number of intermediate results and requires to cache them. Larger number of function calls for a small change. Difficult to cope with broader changes in input Difficult to feed an output of one algorithm to the another. Lower performance since the interactions with the program are unpredictable Domain specific.
Self-adjusting Computing [78] (Memory Management)	Timing improvement and space usage reduction.	Function calls are re-used: But, a recall occur the other calls between the current call and the recalled function call are deleted.
Incoop (Tuned well) [80]	Well-tuned, output is reusable	Applicable to respond to slight changes in input data
ADAPTON (Uses dependency graphs, track the changes and use lazy evaluations) [29]	Can perform well, though the out-layer tries to orchestrate the inner computation based on dynamic information.	Not perform well for eager total order evaluation.
iThreads (Parallel Computation: sub-computation units are re-used) [82]	Supports shared-memory and multi-threaded programming, asynchronous granularity of memory component at given points on sub-computations	Space and performance overhead. Current insertions and deletions of changed data produce displacement of unchanged data.
Imperative Self-Adjusting Computing (Generalised form of self-Adjusting computing) [79]	Write modifications to modifiable references for multiple times. Therefore, no restrictions on writing. The language introduced by this is simple, but able to trace all the dependencies. Improve the efficiency of change propagation	Efficiency comes only when the number of reads and writes of the modifiable are constant. In general, this incurs a logarithmic time overhead. Since no restrictions on writes, it can produce cyclic data structures – therefore, difficult to prove consistency.

Multi-agent Systems (Modularity and autonomy) [26]	Framework for handling the agents' memory, where the memory is produced, customized, and maintained rendering the current situation. Agree to change due to external changes as well as its own preferences about how to proceed	As the system has to handle large number of agents work together it requires more resources and cause for inefficiencies.
Evolutionary Computing (optimize the solution based on the knowledge on the object considered for the computation) [28]	Can yield solutions better than the current solution. Therefore has an evolving knowledge base.	As it is necessary to work with pools of larger data sets, this requires more resources. Improving the computational efficiency is not a main concern.
Neural Computing (Use network of neuron to solve problems) [97]	Learn by themselves. Networks can detect fault neurons and missing information and produce output.	Suffer from memory wall issues. Heavily rely on the underlined hardware architecture. Needs more processing power.
Program Tuning with Adaptability (Automate the optimization of computer programs to support in adapting to environment and requirement changes) [101]	Ability to select best algorithms based on the system load, input and hardware during the runtime. Improve the speed of processing.	Domain specific.
Dynamically Tuned Sorting Library [51] **and Sorting with Genetic algorithmic Approach** [89]	Select sorting technique depending on the input size, standard deviation, distribution of the input data and hardware architecture. Inefficient computation can be eliminated during the	Domain specific.

(Auto tuning Techniques)	process.	
Six-state Continuous Processing Model (Modelled Memory as Conditional Phenomena)	Improve computational efficiency over subsequent program execution cycles, organize memory, remove overhead on memory, improving the access to the knowledge base. prevents processor idling, use the idling time to improve the system, through continuous processing.	

2.4 Quantum Computing

Quantum devices are ultimately designed to be micro- or nano- and their functions are accomplished by fine-tuning the laser pulses. In quantum computing, the alternative to the bit is qubit. Further, A qubit is formed by the spin states denoted as 0 and 1 [110] that are called as spin up and spin down respectively. In addition to that, the transitions between the states are accomplished through the equilibrium of the atomic configuration. The Nuclear Magnetic Resonance (NMR) was the first scheme that was used by the researchers to illustrate the notion of quantum computing, and it is the most successful schema so far. However, the NMR is difficult to be extended. The Superconducting Quantum Interference Device (SQUID) is an updated level of NMR. But, on the SQUID regard, the systematic awareness of the authors seems to be very limited.

The researchers are designing distributed computing [27] systems that use clusters consisting of the strength of many processors to fill the demand entirely. Adding more processors linearly increases the cluster's computing capacity. For every 18 months, the computing power is doubling as per the Moore's predictions. The doubling of the strength of the capability for each new chip generation signifies [27] that nearly half as many atoms are utilized per bit of data. However, this movement reaches the limit of one atom per bit of data one day. Apart from the nanotechnology, the quantum computing also provides the potential to go beyond. Since, the structure built up from qubits, the quantum computers are powerful than the traditional computer. For example, the quantum computers factorize larger numbers efficiently.

A few limitations [111] were displayed by the quantum computers. The first limitation is the speeding up of the quantum computing was achieved only for the cases that have fewer answers or even a unique solution. The second limitation is the quantum computers may derive outputs that are fixed-points of a unitary operator. Stated in another way, the Eigen states of unitary or Hermitian operator comes with Eigen value 1.

Still, not all these software solutions are capable of fully utilizing the underlying hardware improvements. Even though, some of the above-mentioned software solutions display some features of human mind such as some features displayed in logical agents, those have the fundamental difference from the human mind where the memory is not detached from processing in the mind. Further, those do not have a smaller compiler like tactics memory, which grows and organizes over the time due to continuous processing, and enhances processing as seen in the human mind.

2.5 Summary

This chapter has discussed the researches based on improving the memory and processing in the computer, those provided the technical inspiration for the work reported in this thesis. This analysis was done, having a particular emphasis on the memory. In hardware level, the most of the developments were disjoint with regard to the memory and processing separation. Further, different connection mechanism developments between the processor and the memory were also discussed. Then, other than solving real world problems, some software level models were introduced in trying to utilize the underline hardware as much as possible. For examples, by reducing cache misses improving the data locality, chunking the data so as to match with cache lines, and reducing the data routing within the system. Next section, describes the philosophical approach towards proposing new computing model.

CHAPTER 03
THEORETICAL FOUNDATION FOR THE NOVEL COMPUTING MODEL

3.1 Introduction

The chapter two reviewed the technologies and computing models that have been introduced to improve the power of computing. In most of the models, the memory and the processing were treated separately and as raised by Gero [87] the word ‘memory’ is used to denote a distinct place in the computer that stores data or instructions. In fact, the human mind displays a different way of computing than VNA and it can provide accurate solutions fast in a good quality as introduced in the chapter one. As per the arising conditions, processes are continuously executed in the mind, and the human memory has also been a result of this continuous processing. Further, The Buddhist Theory (BT) has explained everything as conditional phenomena. In the same way, the BTM has also explained how thoughts conditionally arise in the human mind and has described the process in the human mind as conditional phenomena. Yet, BT has introduced every effect has a root cause. Further, the relationship between the effect and the cause has been beautifully explained in BT with a set of concepts called 24-causal relations. These causal relations can be used to explain any phenomena, which includes the conditional phenomena that forms the human memory. Therefore, this chapter has explained the BTM and other inspirations, which has laid the foundation for the theoretical framework of this research and has provided an insight to propose a new computing model.

Furthermore, it could find some evidences for the computer modelling of human mind based on BTM. Moreover, some real-world scenarios have been interpreted showing the evolving and continuous nature of human mind. Finally, this has mentioned some other mind theories and has discussed the problems in implementable theories of mind.

3.2 Buddhist Theory of Mind

Buddhism considers the behavior as a result of relations. In last thirty years, new hope on the applicability of Buddhism [112] into cognitive science has been utterly examined by a rising number of established researchers such as Richard Davidson, James Austin, Christopher de Charms, Jeremy Hayward, Daniel Goleman, Francisco Varela, Alan Wallace, and Eleanor Rosch. The Buddhism brings extra value from the scientific perspective, and from the virtuous perspective it gives realistic sincerity. The Buddhist theory of mind (BTM) provides an all-inclusive theory on human mind that goes through many intellectual jobs such as reasoning, remembering, and thinking [112].

According to BTM, human mind consists of a continuous thought-process. All *Viññāna, Citta, Mana, Cēta, Nāma, Cittuppāda,* are considered as similar terms [113] for 'thought' in *Abhidhamma* ("the Buddhist analysis of mind and mental processes" [55]), the distinguished (*Abhi*) teaching (*dhamma*) of load Buddha [55]. Further, from the view of *Abhidhamma*, there is no difference between the consciousness and the mind [113]. Moreover, the consciousness occurs in the cognitive process (*Cittavīthi*) except on the cases rebirth, death and life-continuum (*bhavanga*), where those rebirth, death, and life-continuum are process-freed [55]. In some references like [113], [53], [114], and [115], this cognitive process is termed as the thought-process. Consequently, this research refers the formation of the thought-process, which constitutes the mind, exploiting twenty-four causal relations based on *Abhidhamma* studies.

3.2.1 Thought-process

Thought-process is a conditional flow of thoughts (thought moments) arise due to five-sense-inputs (external objects - *bahiddhā ārammana*) and mind-input (internal objects –*ajjhattika ārammana*) [53] correspondingly forming two thought processes; five-sense-door-thought process and mind-door-thought process. A thought-moment consists of three sub-moments. They are genesis (*uppāda*), static (*thiti*) and cessation (*bhanga*) [113]. Further, thought moments arise together with mental factors (c*estika*) and they exemplify the characteristics of thoughts. Once the process got a considerable input, the thoughts are generating conditionally one after the other. An output from one thought becomes the input to the next thought. If there is no specific input, automatically the thought process is turned back to a neutral thought process which consists of a thought called *Bhavanga*. This neutral process keeps the continuation of the thought process. Next section has described the inputs for the thought process.

3.2.1.1 Input – Object ($\bar{a}rammana$)

The five-sense-inputs are the objects coming into five physical sense doors, namely, eye-door, nose-door, ear-door, body-door (skin), and tongue-door. Hereafter, these are termed as external inputs. Then, the mind-inputs are the objects coming into mind door and hereafter termed as internal inputs.

Further, depending on the nature of the objects, the presentation of the objects is six fold. The internal inputs can be ‘clear’ or ‘obscure’, and the external inputs can be ‘very great’, ‘great’, ‘slight’, or ‘very slight’ depending on the visibility or audibility, or so on [55]. Such a way, six presentational categories of objects can be named.

In this research, for the simplicity, it considers only the clear internal inputs for mind door thought-process and very great external inputs for five sense door thought process. The next section has discussed the thought process arise due to the external input.

3.2.1.2 Five Sense Door Thought-process

Here, let us consider the eye-door process at the presence of a very great external input. The other physical sense door processes are similar to this process. Figure. 3.1 depicts how the relevant thought-moments arise in the eye-door thought-process, once a sense-door got touched with an object. This consists of 17 thought-moments.

→	1	2	3	4	5	6	7	8	9	...	15	16	17	→
***	***	***	***	***	***	***	***	***	***	...	***	***	***	***
Stream of Life-continuum	Past Life-continuum	Vibrational Life-continuum	Arrest Life-continuum	Five door adverting	Eye Consciousness	Receiving	Investigating	Determining	<---Javana--->			Registration	Registration	Stream of Life-continuum

Figure. 3.1: Eye-door thought-process

(three stars denotes the sub-moments in each thought-moment) at the presence of a very great object

The section 3.2.1.3 has discussed the thought process arise due to the internal input, while the section 3.2.1.4 has explained the consequent mind door processes which occur immediately after a five-sense door thought process.

3.2.1.3 Mind-Door Thought-process

An object sensed in the sense-door thought process previously, is become the input in this process. But at this moment, the sense-door thought process is not alive and this input object occurs due to the sanna, the mental factor. Figure. 3.2 shows the mind door process at the presence of a clear object.

→	1	2	3	4	5	...	11	12	13	14	15	16	17	→
***	***	***	***	***	***	...	***	***	***	***	***	***	***	***
Stream of Life-continuum	Past Life-continuum	Vibrational Life-continuum	Arrest Life-continuum	Mind door adverting	<--Javana---->			Registration	Registration	Life-continuum	Life-continuum	Life-continuum	Life-continuum	Stream Life-continuum

Figure. 3.2: Mind door thought-process (three stars denotes the sub-moments in each thought-moment) at the presence of a clear object

3.2.1.4 Consequent Mind Door Process

After receiving an object though five sense door, as mentioned earlier, object enters into a five sense thought avenue and starts five sense door thought process. However, it has been difficult to identify the particular input unless it has been accompanied by four consequent mind door processes.

The consequent mind door thought processes are;

1. Take the earlier received object ($tad\bar{a}tuvattaka\ manodv\bar{a}ra\ v\bar{\imath}thi$).
2. Combine the different parts of the object to form the complete object ($sam\bar{u}haggahana\ v\bar{\imath}thi$).
3. Understand the meaning of the object (features of the object) ($atthaggahana\ v\bar{\imath}thi$).
4. Understand the name of the object ($n\bar{a}maggahana\ v\bar{\imath}thi$).

All these four thought-processes occur at the mind door. At the end of this continuous processing, input is identified. However, once an object is received, the object can be identified only if one has built up a related perception ($sa\tilde{n}\tilde{n}a$) and related mental factors earlier. The next section explains how the memory has been formed according to the BTM.

3.2.2 Explanation for the Human Memory from BTM

'Memory', the term which is used today, is neither explained as a distinct component of the consciousness nor a mental factor in $Abhidhamma\ Studies$. According to that, memory is rather a complex process [114]. It starts with perception ($sañña$) mental factor by labeling (consequent identification) the incoming objects. Then, it conditionally evolves with the thought ($viññāna$) and wisdom ($paññā$), slowly establishing the knowledge [54]. In [113] $sañña$, $viññāna$, and $paññā$ are compared with a beautiful example of an identification of a rupee coin. A toddler gets an idea of the coin regarding its shape, colour, and hardness (that is similar to $sañña$). Then, an ordinary man knows these external characteristics, as well as the use of the coin (this similar to $viññāna$). Finally, think about a chemist who knows all about the coin including how the coin is made and its chemical composition (this is like $paññā$).

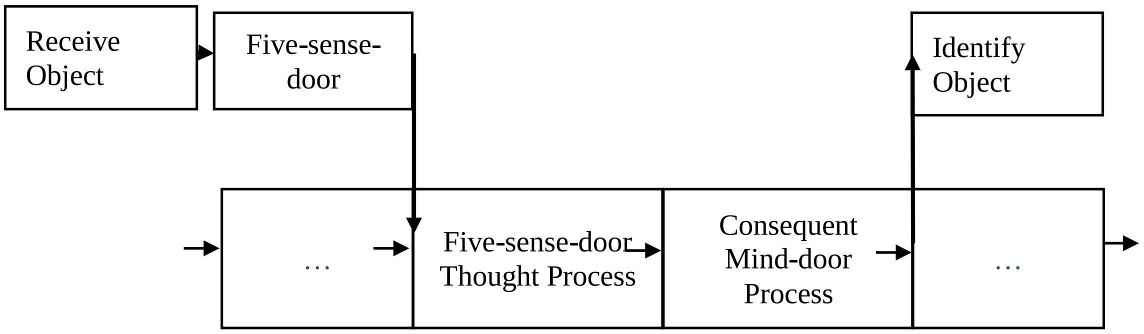


Figure. 3.3: Memory is a continuous thought process

With the five sense door process and consequent mind door processes as seen in Figure 3.3, one can remember the past things (received objects, related features, and actions) and make judgments over present objects through perception ($sañña$). Further, Some have misunderstood the mindfulness ($sati$) in BTM as memory or it is mentioned as the closest connotation for the memory. In fact, the mindfulness is the concentration on the present and not relate to past [55]. However, it contributes to the identification of the object in the memory process by attaching to the current object. Overall, memory is a conditionally evolving process.

The next subsection discusses the most important set of concepts that has been utilized in explaining the thought process that forms the human memory.

3.2.3 Twenty-four Causal Relations in BTM

The best references for the twenty-four causal relations could be found in [55], [116], [117], [118] and [119]. The twenty-four causal relations, which can exist in between causes and their effects [54], has been introduced in $Paṭṭāna$ of $Abhidhamma$. Further, those provides a thorough description on the different ways in which they interconnect the mind and matter phenomena [55].

Now, consider the 24-causal relations in detail:

1. **Root** (What is the Reason/ What is the cause) ($H\bar{e}tu\ Pacchaya$)
   - By means of which, an effect come to be
   - By which an effect is established
   - A serviceable or supportive factor
   - States are entirely dependent on the simultaneity & presence of their respective roots.

2. **Object** (The input) ($\bar{A}rammana\ Pacchaya$)
   - Something which forms the condition for process (Memory as conditional phenomena)
   - Things on which the subject delights in or hangs upon are objects.

3. **Predominance** (Priority) (Rise of one thing, into which the others bend towards) (*Adhipathi Pacchaya*): As;
   - Concentrated Intension/ Wish (chanda)
   - Energy/ Effort (viriya)

- Consciousness/ Thought (citta)
- Investigation/ Reasoning (vimamsa)

4. & 5. Proximity & Contiguity (immediacy) (*Anantara and Samanantara Pacchaya*)

Any state of consciousness/ mental phenomena conditions the immediately following state in the process of consciousness. (One thought moment perishes immediately giving birth to another)

E.g. Visual Process:

Once the mind received an object, eye-consciousness occurs and then, instantly starts investigating the object forming the mind consciousness.

<u>Proximity</u> (No interventions) – Conditioning state ceases to immediately arise the conditioned state. (No Interval)

<u>Contiguity</u> – Conditioning state ceases to immediately arise the conditioned state with fixed order. (arise suitably)

6. Co-nascence (Simultaneous Arising) (*Sahajāta Pacchaya*)

A phenomenon, one forms a condition such a way that, simultaneously with its arising, also the other thing must arise.

E.g. Feeling, Perception, Mental Formations and Consciousness (4 mental groups)

7. Condition by way of mutuality (*Aññamañña Pacchaya*)

States may be causally related (support) one another for the emergence by the way of mutuality.

8. Support (*Nissaya Pacchaya*)

- A phenomenon which aids another phenomenon by the way of foundation or base.
- Causal relation of dependence

9. **Decisive-support Condition (Inducement)** (*Nissaya Pacchaya*)

- A prominent supportive condition
- A strong inducement or cogent reason.

By the way of object, proximity, natural decisive support.

10. Pre-nascence (*Purējāta Pacchaya*)

Previously arisen something which forms a base for another thing which is arising later on.

11. Post-nascence (support & strengthen the preceding one) (*Pacchajāta Pacchaya*)

A necessary condition for the preservation of the existing thing

12. Repetition (increased power & efficiency) (*Āsēvana Pacchaya*)

- Repeated practice, as a rule, leads to proficiency.
- By repetition one can acquire certain amount of skill in any particular thing. (Students become proficient by repeated studies in lesson)

13. Karma (co-nascent/ asynchronous) (*Karma Pacchaya*)

Purposely produce something (generating condition) physically or mentally.

14. Karma-Result (*Vipaka Pacchaya***)**

Mental states of resultant types of consciousness are causally related to coexistent mental states & material phenomena by way of effect.

15. Nutriment (*Ahara Pacchaya***)**

Mental contacts or impressions causally related to feelings, volitions or moral and immoral actions to rebirth consciousness and rebirth consciousness to mind and matter. (as material foods sustains the physical body)

16. Faculty (*Indriya Pacchaya***)**

Conditioning state relates to conditioned state conditioned states by exercising control in a particular department or function.

17. Jhana (*Jhana Pacchaya***)**

Causally related one another and other concomitants (happens at the same time) byway of close perception and contemplation. (carefully think for long time)

18. Path (function as a means for reaching a particular destination) (*Magga Pacchaya*)

A way or road to achieve something, where the road consists of several path factors which are causally related.

19. Association (*Smapyuktha Pacchaya*)

Certain mental states with distinct characteristics arise together, perish together, have one identical object & one identical base, they are causally related.

20. Dissociation (*Vippayuktha Pacchaya***)**

Helpful to each other being dissimilar
(different object, different base)

21. Presence (*Atthi Pacchaya***)**

Exist in the present, condition for other phenomena. (Being pre-nascent or co-nascent)

22. Absence (in ceasing the current gives the opportunity to another to arise immediately next) (*Natthi Pacchaya*)

Disappearance of the predecessor, the successor appears. Such is the causal relation by way of absence.

23. Disappearance (*Vigatha Pacchaya*)

Own disappearance, gives the opportunity to the next mental state to arise.

24. Non-Disappearance (*Avigatha Pacchaya*)

Similar to the Presence

3.2.4 Exploiting Twenty-four Causal Relations in Explaining the Thought-process

The explanation for the thought-process using an extracted set from twenty-four CRs of BTM can be found in [55], [113], and [53]. Further, [53] provides a simple clarification for the start and the continuation of thought-process. In the thought-process, a thought is arising immediate after ceasing the previous thought with the help of the several of twenty-four CRs. The reference [53] divides these relations into four groups ignoring slight differences. First one is object relation. Without an object no thought-moment arise and attributes of the object determines the nature of the thought-process. Then, the second is actions ($Kamma$), where the resultant thought-processes of past-actions are produced, and come into inquire and investigate. Thirdly, the Presence relation can be accounted. In there, certain thoughts are appearing only in the presence of another thought. The final one is decisive support relation, which discusses the dependence.

In between these, one can notice many further relations in the mind process [53]. For example, ceasing thought-process conditions the incoming thought-process by the way of proximity, contiguity, absence, and disappearance relations. Another is, previous Karmic impulsive moment or impulsion ($Javanas$) conditions for succeeding impulsion ($Javanas$) by the way of repetition relation. Moreover, co-nascent thought and other mental factors are conditioned for each other by the way of association [55] and mutuality. In addition to these, $Kamma\,Vip\bar{a}ka$ relation produce the results of the $kamma$ (actions), asynchronously or at the same time.

Similarly, continuous memory process, which consists of several thought processes also can be explained exploiting 24-CRs.

3.3 Real-World Inspirations

As mentioned earlier the human mind has an evolving memory that has been a result of continuous processing over large knowledgebase according to the conditions arising [18]. The human mind has been become matured over continuous practice. Further, we have observed the nature of human mind and its memory in several real-world scenarios [19] and had a great insight for the research. By its very own nature and it is a well-known fact that the things we process more are established in the memory very well and do again faster. Because of this, the pupils who are preparing for the exams, take short notes through which they can access the large knowledgebase on a particular subject. Furthermore, they do tutorials and identify question and answering patterns. Continuously practicing this process over revision cycles, they improve the ability to answer the question papers as well as improving the short notes by adding and removing entries. Meanwhile, they identify the relationships among knowledge entities and question patterns as well. However, this short note never becomes the large knowledge base itself. Some of the tuition masters teaches the techniques to successfully develop a such a well-organized memory in the student by conducting revision classes. Through this memory the student can drive through and operate on the large knowledgebase in order to derive answers in the examination. Although the knowledgebase is rich, if student does not have such a driving memory, the student cannot answer the questions in the exam. Furthermore, this idea has been explicitly explained with another real-world ordinary example. Let us consider a student who is preparing for a presentation on a particular topic with a presentation program (slide show). Initially, the student refers the large notes on the topic and add major points to the slides, which of those can be used to refer the entire knowledgebase. In the cycles of preparation process, the student can do refinements to the slide show by a set of tactics such as adding new points, deleting unnecessary points, making links between existing points, and classifying and prioritizing the points as necessarily. In this scenario, the slide show and the set of tactics can be seen as a smaller tactics memory, which is gradually updated through continuous preparation. However, it is not eventually filled up with entire knowledge base, but will fill up with the label like points, which we can use to access the large knowledgebase. Finally, after the preparation process and the day of the presentation, the student will be able to have a well-organized slide show for his presentation. Here, the large notes on the topic and the queries raised on the topic can be considered as the input to the continuous processing and until new knowledge arrives, the refinement cycles should be carried out. The oral presentation can be mentioned as an external output, and the changes made and the slide show can be considered as an internal output of the process.

Moreover, this improving short note, preparing the slide show and the respective scenarios are similar to a well-organized and always refining smaller tactics memory of a veteran professor with a set of tactics, who does not need notes or presentation slides when doing the lectures [17]. Further, he is capable in answering questions in a shorter period than his students, it is not because of any secret, but it is because of this smaller tactics memory which is organized over the past period of time. Another example is the medical student vs veteran doctor, where the medical student has gradually developed a well experienced specialist doctor. How is this happened? This scenario is a good example to highlight the smaller compiler like human memory, which is gradually progressed. This behaviour of the human mind has also well-displayed in all the other skill workers such as carpenters, masons, and chefs. It is possible to draw a familiar example from the home kitchen. Now, let’s think about a young lady, who cooks in the kitchen for the first time with a book of recipes. Most of the time, in the first day, even-after spending a lot of time, at the end of her venture, the kitchen and the all could be a complete havoc. However, day by day she works in an organized manner by organizing her memory after identifying ingredients, processing and cooking patterns with the knowledge received from the recipes and the kitchen environment. The cooking is involved with many sub tasks such as washing, peeling, cutting vegetables, seasoning with spices, scraping coconuts, and making coconut milk, where the similar or related things can be done together. For example, washing, peeling, cutting all the vegetables are done together, and all curries are seasoned together. Automatically, the related tasks in the cooking can be clustered and accomplished together. These task clusters can be completed one after the other, accepting the output of one cluster processing to the next. Further, preceding job should be accomplished before start the next job. Such a way she uses tactics. The young leady become competent in cooking with her well-organized smaller tactics memory with the tactics.

The continuous improvement of the human memory is taken place with the knowledge, experience and the organization through the experience, knowledge and organization through continuous processing with scrutinizing, identifying patterns, classifying, prioritizing, retrieving, and other such tactics. In fact, its performance depends on the input, health condition and the continuous processing. The next requirement was the identification of necessary tactics, where the tactics are derived from an extracted set of above mentioned twenty-four causal relations. Indeed, this concept has philosophical aids from Buddhist Theory of Mind. It is an evident fact that human beings have exceptional capabilities in executing instructions using the above mentioned small tactics memory, which is different from the current smaller memories in the computer. In fact, in developing hardware or software solutions in order to enhance performance of computers, this concept has not been exploited yet. However, it could find a few efforts on computer modelling of human mind based on BTM.

3.4 Attempts in Computer Modelling of Human Mind Based on BTM

Despite of the difficulty in finding computing models, which were introduced to improve computational efficiency having inspired from the human mind, it was found few efforts based on BTM in modelling the human mind. First, the thought process was modelled with a transition matrix [19]. In this paper, the main focus was on the determining consciousness (Figure. 3.1) of five sense door thought process. Further, the major concerns were the Markovian nature of a thought process, influence of the immediate thought process, and the response of the process with respect to the object. However, object or the nature of the object was ignored. Finally, as a future work, this work suggests to include the influence of the previous thought process for this modelling process. In another research, two axioms were derived [20], where the first axiom based on the concept that every thought process arises with respect to an input and the second one is the mind probabilistically behave depending on the emotions. Finally, a theoretical basis for the Artificial Neural Networks was set by these two axioms focusing on the number of neurons and the weights (The work stated in[20] is initially started as an effort on modelling the process in the human mind). Both of these models examines a special potential between thought process. It is a limiting state. Further, it is one of the factors that decides the inherent quality of a person and maintains one's continuity of thought process [19]. Moreover, this energetic condition of existence [19] facilitates subsequent thought process to be begun. Next, OntoBM [15] is an ontology of human mind, which models the thought process. This includes eighty-nine thoughts (Citta), fifty-two mental factors (Cetasikas), and other thought associated influences. However, the interpretation about the connection between the thoughts and the respective mental factors in this article is arguing, where thoughts are explained as constituent of mental factors. In fact, thoughts have the same base and the same object as of the mental factors, and the thoughts starts, exist, and end together with the respective mental factors [19]. An expert system has been developed [21] in order to find related mental states with respect to a thought moment. As similar to the work appeared in [19], this work also considers the immediate thought process.

Next, two subsections have been discussed about the mind theories, which are based on western philosophical view.

3.5 Other Approaches on Mind and Memory

Steven Pinker who is a cognitive scientist with a western philosophical background, has presented his view [120] on the human mind as a naturally selected computing system structure. As he raised, there exists a gap between human cognition and the abilities of probable computational systems. Even though, there was a conversation on “how the mind works”, it was failed to provide a proper answer on this regard.

Mary Litch [121] also produces a concept of computation, in the setting of computing model of mind that provides a connection between physical and psychological description. She says that if we talk about theory of computation related to philosophy of mind, we should talk about it in the setting of computing model of mind. Again, she claims that non-learning connectionist systems are consistent with the computing model of mind as simulated on digital computers and argues about the two connectionist approaches; digital and analog. She says connectionist systems as simulated on analog computers are inconsistent [121], because, these have collapsing continuous states and it leads to new analog devices in which those the computational processes are not permitted. She talks about two unanswerable questions: first one is how a physical thing possesses physical states and how it can be transits from one state to another. The other question is how we can identify exact physical implementation of mental states. To answer these questions, she said that, for the first one, provide an existing evidence of physical device that did this, and for the second one, provide an identity condition for equality of mental states with computational states. Further, through computational thinking it was expected to derive solutions by formulating a problem by finding a matching model [122]. Here, the solution is said to be represented by computational steps and the algorithms. Implementation of the mind with mathematical measures has led to transfer from cognitive process to connectionism [123]. This connectionism and Buddhism have common views as both consider behaviours are emerging from the relationships.

Again Mary Litch raised that, the physical real valued quantities should be digitized to model them [121] using discrete physical devices. This has been inspired by the connectionism. In modelling the mind, it is required to apply precise mappings between psychological, mathematical, and physical states without changing the functionality or respective properties. A similar idea was produced by Elodzislaw Duch [124], where he pointed out the requirement of enhanced sensitive language to clearly express mind states. Although, the cognitive science is capable in explaining the mind in a better way. Further, such a model should have the ability to exhibit better estimates on brain functions and the link with the respective mental states.

Overall, the modelling of mind is so complicated. Numerous clarifications were introduced from the researchers in the field those who had different philosophical perspectives. In addition to that, the cognitive scientists have introduced different memory models, where the engineers have tried to build computational models based on those memory models. Next section discusses about such other memory models.

3.6 Human Memory Models

Waugh & Norman introduced a human memory model, which was known as “the boxes in the head”. The model consisted of long term store and a short term store. Later, in 1968, the “Classic information-processing model of Memory” was introduced. It had an extra store to hold the memories at the moment, when the mind perceived an object. This extra store was called as the sensory memory. In both of the cases, the STM occurred due to the stimulus is passed to the LTM only if those maintained the practice over the STM.

Zhang [125] has improved the above second model, by defining the dynamic form of STM as working memory [126] and dividing the LTM into two parts such as procedural and declarative memory. Zhang declared that the declarative memory can store all the knowledge that can be articulated symbolically or consciously retrieved through a language by talking or writing. Further, according to him, the procedural memory can store the skillful tasks such as typing, rowing, and riding. In addition to that, he proposed that as in the second model, at that moment, when a sensor received an object, the information of the incoming object is stored in the respective sensory memory and then passed to the STM until the sleeping time. If the human being is in the sleep, the information stored in the STM is transferred into the LTM. Moreover, the classes of information stored in LTM are special information, physical laws, beliefs, values and social goals, motor skills, and perceptual skills.

Different concepts with a western philosophical view have been introduced as Bartlett's Remembering (basically, the mind stores traces of events and according to my understanding Bartlett has seen this behaviour as an organized mass behaviour)[39], constructive memory (memory involves in a construction process) [40], and Atkinson-Shiffrin (Introduced a framework, in which the short-term memory is explained as a working memory of the Baddeley [42] and is separated from long-term memory. Further, the information moves from STM to LTM, and vice versa. Finally, the most important thing in this regard is the focus on the control processes and techniques used to code and store information, more importantly the rehearsal) [41]. These models have been widely using in improving computing technologies.

The succeeding section has covered the problems in implementing the theories of mind.

3.7 Problems in implementable Theories of Mind

Pentti O. A. Haikonen [127] said although the philosophers introduced the theories of mind, they do not have any technical knowledge on how these theories can be implemented. Therefore, they couldn't provide good specifications or guidance for the engineers to implement models based on these theories. On the other hand, the engineers do not have a comprehensive understanding on the mind theories. Further, identification and understanding of the process of mind, consciousness, emotions, and inner speech, was quite difficult. Stated in another way, the engineers require an algorithmic or computing approach which included the necessary instructions using standard technical commends or terms. As a matter of fact, the engineers were not capable of producing the simulations covering the entire theory. Lastly, Hikonen concluded that although these problems can be solved using a low-level system approach and a high level algorithmic approach. However, these methods have their own inherent problems to be addressed.

3.8 Summary

This chapter discussed the thought-processes that has been the constituent of the human mind and the human memory according to the BTM. Furthermore, it has described the 24-causal relations, which can be used to explain any phenomena. Then, the next section exemplifies many real-world cases, which have showed the evolving nature of the human mind. Next it has mentioned some attempts in modelling the human mind in the computer based on BTM. Later, some other mind theories under the western philosophical view has been discussed. Finally, this has drawn the attention to the problems in the implementable theories of mind. Overall, the consequence of the continuous processing over a large knowledgebase would be the human memory. The theoretical basis for this concept, the major supposition of this research, has been discussed in this chapter. The next chapter covers the new approach towards the proposed model, exploiting theories in modelling process, and design and implementation of the proposed model.

CHAPTER 04
NOVAL APPROACH TO A COMPUTING MODEL

4.1 Introduction

It has been intuitive that discovering a novel computing model has been an outstanding research challenge. The chapters one and two have revealed that there are different approaches in doing so. However, as mentioned in Chapter 1, it has been difficult to find a computing model that is similar to the continuous processing model that is taken place in human mind according to the BTM. The most influential factor that has been found in the human mind was its ability to improve accuracy, quality and efficiency of instruction execution over subsequent cycles in mind.

This chapter first discusses the approach for SSPM, the hypothesis, input, output, and the derived model, i.e. the SSPM. Then, it has been discussed how the system has been derived from the BTM. Finally, it has been reported the implementation details of the proposed model.

4.2 Hypothesis

This research hypothesized that *the processing power of the computer can be enhanced with the support of a smaller tactics memory, which improves as a result of continuous processing*.

Receiving the inspirations from BTM and real-world as stated in chapter three, this chapter presents the new approach in introducing the new computing model SSPM. The SSPM consists of a smaller compiler like special smaller tactics memory with a set of tactics. The SSPM that forms the smaller tactics memory was designed similar to the process of maintaining the human memory. It was postulated that the presence of the smaller tactics memory in human mind. In fact, it is capable in exploring the whole main memory or the stored knowledgebase. Furthermore, the smaller tactic memory becomes larger within certain boundaries depending on the domain and updates and become organized over the time. This smaller tactics memory contains a set of strategies in executing larger programs in the main memory. Moreover, it was convinced that this smaller tactics memory can be developed as a software solution or as a hardware solution.

As per the arising conditions, an entry in the smaller tactics memory was triggered as similar to the continuous processing taking place in enhancing the human memory. In fact, the processing a set of instructions means nothing nonetheless a consequence of satisfying the conditions in the body of a procedure. This is a core concept that comply with the Buddhist philosophical thought that everything in the world can be explained as a result of conditional phenomena, where every effect has a cause. As explained earlier, since the 24-CRs can be used to model any phenomena, by defining the relationship between the effect and the cause, it was proposed to exploit a subset of CRs, in designing the smaller tactics memory.

The internal process of mind can be summarized into a block diagram and this diagram can be presented as the basic structure for the new theory of computing. The Figure. 4.1 depicts how the conditional phenomena are taken place at high level.

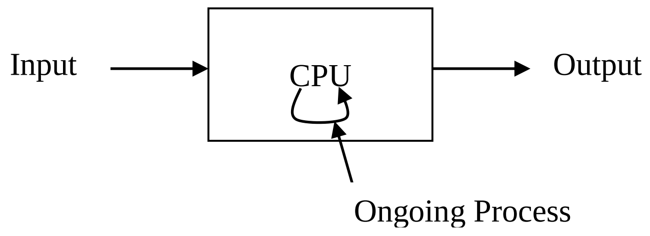


Figure. 4.1: High Level Block diagram for SSPM

The coming section has described the inputs of the proposed model.

4.3 Input

The SSPM system has two types of inputs, called Internal and External inputs, and are identical to the human mind's internal and external inputs. As per the inspirations received from the internal inputs accepted by the mind-door of the human mind, the ‘Internal Input’ of the SSPM was introduced. With a great probability, the next internal input was associated with the currently running process in the memory. Further, an external input of the computer could be a user input or another input coming from the external environment similarly. This is alike the input coming through the five-sense-doors in human mind [55] as mentioned in the chapter 3. Indeed, the internal inputs was generated by the system through the operations deposited in the smaller tactics memory, in the absence of the external inputs, and those were related to the current process with more possibility. The concomitant of this is the internal inputs in the human mind coming through the mind-door process.

Moreover, the external inputs can be categorized into four categories as follows:

1. input with new operation
2. input with an operation which is different from the most recent operation.
3. input is similar to the previous input.
4. Frequent input.

The internal inputs are three fold as follows:

1. input with an operation which is different from the most recent operation.
2. input is similar to the previous input.
3. Frequent input.

Note that, the first category in the external inputs is not available in the internal inputs.

Doing thorough analysis on the input data, it could identify some patterns or rules of the incoming data in addition to the above categories, and these were valid for any input. For example, these inputs can be lists or fractional expressions with any number of elements or questions/queries those correspond to particular sections of knowledgebase.

Suppose that X and Y are two sets. In fact, X be the previous input and Y be the current input, where $X = \{x_1, x_2, x_3, \dots, x_n\}$ and $Y = \{y_1, y_2, y_3, \dots, y_m\}$, n and m are any arbitrary positive integers. (These *x*s and *y*s could be letters or numbers or fractions of fractional expressions or knowledge entities in questions/ queries). The input patterns are related to the similar and different inputs above (the inputs with some relevancy to the most recent input) and the relevant examples can be seen in the Figure. 4.2.

(a)

$IP1$: $X \subset Y$

$IP2$: $X \nsubseteq Y \ and \ Y \nsubseteq X \ \Rightarrow \ X \cap Y \neq \phi \ or \ X \cap Y = \phi$ (two cases are there as shown in Figure. 4.2 (b))

$IP3$: $Y \subset X$

$IP4$: $X = Y \neq \phi$

(b)

$IP1$: $X \subset Y$

Let $X = [1,2,3]$ and $Y = [2,3,1,5,6]$, then $X \cap Y = [1,2,3] = X$

$IP2$: $X \nsubseteq Y, \ Y \nsubseteq X \ \Rightarrow \ X \cap Y \neq \phi \ or \ X \cap Y = \phi$

1.Let $X = [1,2,3,4]$ and $Y = [2,3,1,5,6]$, then $X \cap Y = [1,2,3] \neq \phi$

Or

2.Let $X = [1,2,3]$ and $Y = [4,5,6]$, then $X \cap Y = [] = \phi$

$IP3$: $Y \subset X$

Let $X = [1,2,3,4,5,6]$ and $Y = [2,3,1]$, then $X \cap Y = [1,2,3] \neq \phi$

$IP4$: $X = Y \neq \phi$

Let $X = [1,2,3]$ and $Y = [2,3,1]$, then $X \cap Y = X = Y = XUY = [1,2,3] \neq \phi$

Figure 4.2: Input Patterns (IP) (a) Patterns (b) Example for each pattern

After, processing the input through the proposed model, different outputs can be produced as follows.

4.4 Output

The outputs have also been two fold. First one has been the external output given after executing the program with external inputs. Then, the second output type is internal, but again has three categories. The first internal output has produced by the processing over the internal inputs. The second internal output has been the improved program itself. Then the last internal input has been the updated smaller tactics memory.

Next subsection explains the computing model that has been introduced imitating the continuous processing model, which made up the human mind.

4.5 Propose the Computing Model

This section proposed the six-state continuous processing model, and it is the core of this thesis. The model is abbreviated as SSPM. The SSPM system initially begins with an internal process, where the particular entry of operation was arbitrarily chosen out from the list of entries stored in the initial smaller tactics memory that were correspond to the operations in the specific domain. Further, the respective instructions of the operations saved in the knowledgebase can be invoked through these entries stored in smaller tactics memory. The system shifts to the internal mode, once an internal input is entered. The program can accept an input from the external environment, only when the present inner process sleeps. If the program is in the external mode and if there is no external input, the program can move back to the inner process. However, in the unavailability of external input, the inner process is proceeded with the actions linked to the latest external input. The system conducts ongoing processing in such a manner.

The model accomplishes a series of tasks, during the ongoing processing over generations. Particularly, it identifies the inputs and operations, adds library files for new operations, classifies appropriate operations with respective information and directives, prioritizes the operations relevantly, creates recurrently arising operating modules and deletes needless or wasteful modules and directives as well as the useless information. Such a way, the corresponding entries develop and organize the smaller tactics memory. In addition to that, these actions under the above mentioned two process categories can occur in a one stream. Moreover, the tasks, namely, deletion, classification, additions, and prioritization can be accumulated under the general term 'Organizing'. Consequently, the system is gaining improvements by iterating this organizing job. Depending on the process category, the results of each process can also be generated externally or internally.

As introduced in the first chapter, the newly presented computing model comprises of six states, specifically "New", "Ready", "Running", "Blocked", "Sleep", and "Terminate" as shown in Figure. 4.3.

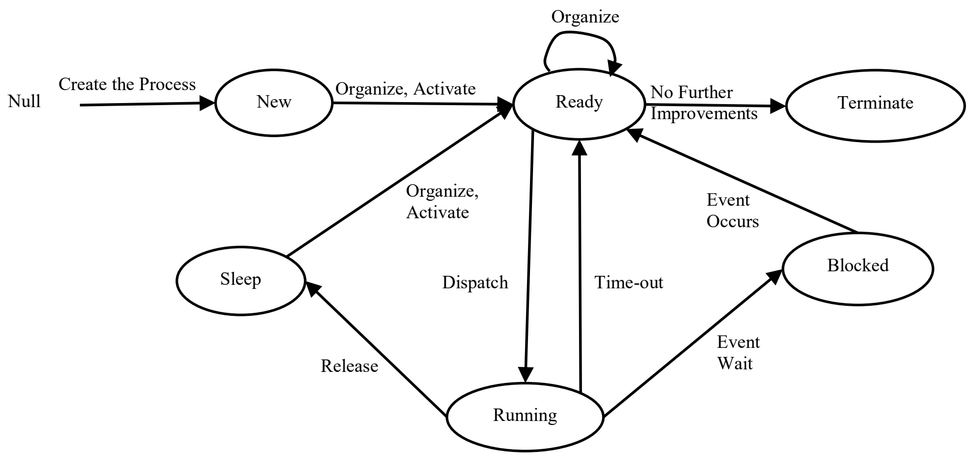


Figure. 4.3: Six-state Continuous Processing Model (SSPM)

At first, neither the recently generated processes were organized nor activated those processes were in the ‘New’ state. Once the processes were organized and activated, those were moved to the ‘Ready’ state. Then, a process, which was running on the processor was in the ‘Running’ state. A process was switched to the ‘Sleep’ state after finishing the execution enabling some other process to be initiated, executed or continued. Furthermore, if a process had to wait until a specific task to be completed, then the process was in ‘Blocked’ state. Finally, if a process was neither necessary to be modified nor requires any execution can be ended, and the state can be updated as ‘Terminate’. The states and movements between the states in the novel computing model are perceptibly illustrated in the Figure. 4.3. Introducing this model, it was expected to enhance the processing in the system and the memory process using a set of tactics maintaining the continuity. The coming section describes the exhibited features of the suggested model.

4.6 Features of the New Model

This model has several characteristics, which one can use to distinguish this model from others. Those are:

1. Internal and external processes

   The SSPM compromises of two types of processes: external and internal processes. These inner processes are aroused due to the system generated inputs, when the external inputs are absent, and are related to the recent external inputs those externally entered by the user. Then, the external processes are aroused due to external inputs.

   Internal Process – similar to the Mind-door thought process in the human mind (based on BTM)

   External Process – similar to the Five-sense-door thought process in the human mind (based on BTM).

2. Continuous Processing

   Combination of the above internal and external processes forms continuity of processing due to arising conditions.

3. Conditionally evolving smaller tactics memory

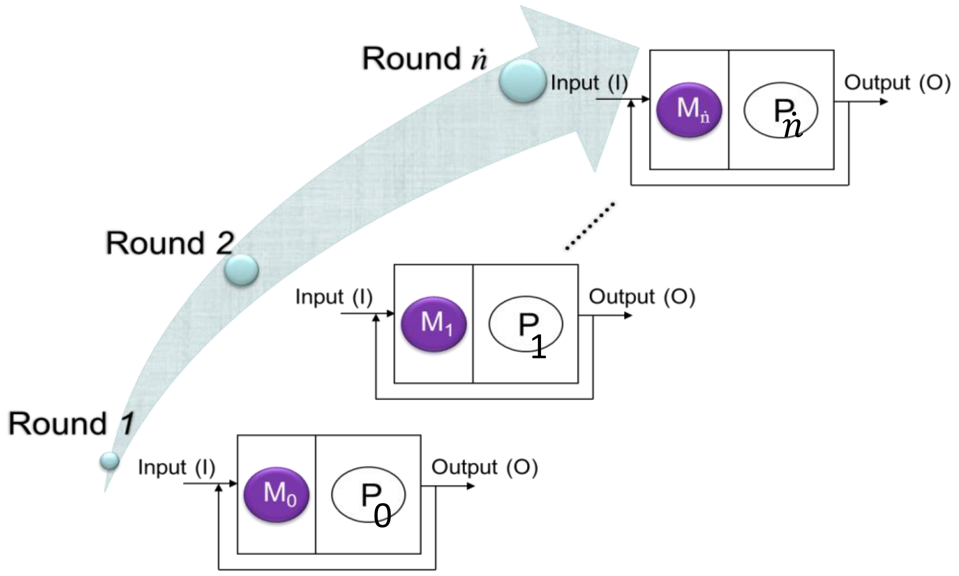


Figure. 4.4: Conditionally evolving smaller tactics memory

As shown in the Figure 4.4, the SSPM starts its execution with an initial smaller tactics memory, which allows to access the system's knowledgebase. Over the consecutive execution cycles this smaller tactics memory evolve with a set of tactics by being well organized. The set of tactics can be names as follows; (these are included in the transitions of the Figure.4.3)

a. Inputs and the relevant jobs are recognized.
b. For new jobs, the relevant library files are created.
c. The jobs together with the relevant data and instructions are classified.
d. Such jobs are prioritized for the execution
e. For frequent jobs, modules are created.
f. Unnecessary data, and unnecessary or inefficient instructions or modules are deleted.

In establishing the memory in the human mind, all the thought processes with consequent mind-door thought processes are involved (based on BTM). Similarly, this model undergoes a continuous process in order to improve the memory. Further, this is permanently stored in the computer (secondary storage), it is activated when the program starts its execution and remains activated throughout the program execution cycles. With this conditionally evolving smaller tactics memory, the system will gain improvements in its speed and quality in subsequent processing cycles. Due to all these features, it is clear that this smaller tactics memory is different from the smaller memories introduced to current computing.

Next section discusses possible users of the system.

4.7 Users

Due to the above highlighted features of the model, it is applicable to the systems which need continuous processing. Specially, the software or game developers can use this model in design and implementation phases, as the designs and the programs or games are to be refined in cycles. It is beneficial, rather than a random selection with a pool of programs in evolutionary computing. As a single program can be improved with the time over program execution cycles. In high level, air traffic control systems can be suggested as a good example for the applicability of this model, since such monitoring or controlling systems must be always live.

4.8 Exploiting Twenty-four CRs in Modelling the SSPM

As mentioned in the chapter three, it is possible to find better explanation for the human mind from the BTM and as explained in Chapter 1 this research exploits BTM in modelling the SSPM, which has imitated the human mind. Further, the section 3.2.4 has described how the thought-process and the human memory can be explained by exploiting a set of twenty-four causal relations (CR) in BTM. Stated in another way, BTM describes the mind as a continuous thought process which encompass of unceasing flow of thoughts pertaining to our actions. Further, memory has been defined as a result of processing on the body of existing knowledge. Moreover, this section has explained how the BTM has been exploited in forming the model of continuous processing that mimic the evolving nature of the human mind. Stated in another way, the SSPM that models a special kind of memory for computers, was explained by exploiting a set of fifteen CRs from 24-CRs of BTM, but looking from a technical perspective.

Since, the Buddhist theory of mind analytically explains how the human memory is form, in designing the continuous processing model, it has been exploited 24-CRs stated in Patthana Prakarana [54] in BTM as in [17], [18], [128], [16]. Further, this provides a different perspective to think about the functionality of the computer and its performance improvement.

According to the BT, every effect has a cause, and there exist a CR between the cause and effect. These are the relationships described in 24-CRs in BTM. Furthermore, the BT interprets that any phenomenon is conditionally arising. Correspondingly, the human memory is also defined as conditional phenomena that occurs or remains as per the arising conditions [89]. Moreover, how this arousal is taken place establishing the continuation of the process [17], [18], was defined by a set of CRs in BT.

Specifically, this set consists of 15 CRs out of 24-CRs in BTM [16] such as Root ($H\bar{e}tu$), Object ($\bar{A}rammana$), Presence ($Atthi$), Support ($Nissaya$), Post-Nascence ($Pur\bar{e}j\bar{a}ta$), Pre-dominance ($Adhipati$), Co-nascence ($Sahaj\bar{a}ta$), Association ($Sampayuktha$), Mutuality ($A\tilde{n}\tilde{n}ama\tilde{n}\tilde{n}a$), Repetition ($\bar{A}s\bar{e}vana$), Proximity ($Anantara$), Disappearance ($Vigata$), Post-nascence ($Pacchaj\bar{a}ta$), Karma ($Karma$), and Karma-Result ($Karma - Vipaka$) [16] were only necessary for the modelling of actions of the transitions of SSPM. The respective set of CRs together with their own duties in establishing the continuous processing are included in the Table 4.1.

Table 4.1: CRs of BTM in Designing the Actions of the proposed Computing Model

CR	Major action formed in the model
Object ($\bar{A}rammana$)	recognize the external or internal input
Root ($H\bar{e}tu$)	creation of a process for a given operation, can support the continuation of the system by being established
Co-Nascence ($Sahaj\bar{a}ta$)	produce the related things together
Association ($Sampayuktha$)	related things exist and get deleted together
Mutuality ($A\tilde{n}\tilde{n}ama\tilde{n}\tilde{n}a$)	related things help each other for the execution and to exist
Pre-Dominance ($Adhipati$)	the process with the highest priority dominates the system
Presence ($Atthi$)	make the process or space or time available
Support ($Nissaya$)	make the ground for execution
Pre-Nascence ($Pur\bar{e}j\bar{a}ta$)	activating new processes by loading them to the memory to execute them later
Proximity ($Anantara$)	make no interval between two processes and maintain the continuity
Karma ($Karma$)	execution of ready processes
Repetition ($\bar{A}s\bar{e}vana$)	execution over generations
Disappearance ($Vigata$)	delete the unnecessary or inefficient processes/ modules/instructions
Post-Nascence ($Pacchaj\bar{a}ta$)	Make the continuation of a blocked process by a newly occurred event.
Karma-Result ($Karma - Vipaka$)	producing the results of execution

This paragraph has been allocated to provide an answer to the question how the rest of the relations have become not necessary for this modelling. First it should be emphasized that, when explaining a particular scenario, it uses the most appropriate set from the 24 CRs [54]. The relations Contiguity, Absence, and Non-disappearance have been omitted, because those are similar to the relations Proximity, Disappearance, and Presence. Then, one can ask, if those are equal why they have introduced different terms. This is because of the slight difference of the angle we looking at them [55]. That is, whether we are looking at the relation from the conditioning state or conditioned state. Therefore, it has been exploited the idea of considering them as similar terms, Contiguity, Non-Disappearance, and Absence terms were omitted. Then, the relation Dissociation is involved between distinct phenomena such as mental phenomenon and material phenomenon. As this research considered the mental phenomena in the human mind, another distinct phenomenon like material phenomenon was not involved. Therefore, the Dissociation condition was avoided. Next, the Jhana and the Path relations were omitted as those are associated with close contemplation of the object and reaching towards the blissful destination the Nibbhana (or reaching towards the woeful destination) respectively. When it comes to Decisive support, it concerns the strong dependence. This has three categories such as object, proximity, and natural, which have been discussed less under Object, Proximity, and Pre-Nascence relations (without the strong dependence term). The ideas coming from these three relations were sufficient and it was not necessary to use Decisive Support for this modelling process. In fact, the Nutriment condition supports to grow. This also has categories such as material nutriment and mental nutriment, where the material nutriment is immaterial for this scenario as it is concerning about the nutrients in the edible foods. Further, the mental nutriment has three categories such as nutriment contact, mental volition, and consciousness. However, the volition and the consciousness are rather spiritual not necessary to be applied in this level. Since, these concepts have been used for modelling in the computer, even though the Object is not the best match for the nutriment contact, it covers the purpose of using nutriment contact in the level of computer for now. Therefore, also the Nutriment contact was omitted from this designing. The last omitted concept is Faculty condition, which considers about controlling a particular department. This also has three types. Those are Pre-Nascence Faculty, Material life Faculty, and Co-nascence faculty. Pre-Nascence Faculty involved in ‘separately’ controlling the sense consciousness aroused due to five sense doors. In this model, it was not concentrated on such a separation. The second faculty controls the material phenomena. The last faculty associated with the material phenomena and the co-nascent mental states. Consequently, the Faculty condition was avoided. At last, it can be determined that these nine CRs and their variations are irrelevant for this design process. However, this model can be extended with the Faculty and Nutriment conditions for the field of robotics.

The conditions have three responsibilities to accomplish, namely, maintaining, producing, and supporting. Correspondingly, 24-CR contributed in the SSPM by accomplishing following one or more responsibilities for the conditioning state-conditioned state transition [89].

Then, it was possible to use the actions stated in the above Table 4.1, in modelling the transitions of the SSPM as mentioned below. All the transitions in the proposed model act towards the Ready state has been refined and has introduced three new transitions, Ready-Terminate, Ready-Ready and Sleep-Ready. Further, the modified transitions were Running-Ready, and Blocked-Ready. Below Table 4.2 shows how the transitions are formed, specifically mentioning from which causal relation each of the action has been derived.

Table 4.2: Exploiting CRs of BTM in deriving the transitions

Exploiting Twenty Four Causal Relations in Proposed Processing Model		
State Transition	**Action(s)**	**Causal Relationship(s)**
1. Null → New This transition occurs with respect to creating a new process.	1. Create a new process a. New job is available. b. Create appropriate data structures. i. Process Control Block (PCB) ii. Linked Queues iii. Other expanded data structures c. Allocate space and initialize. (parameter passing) d. Set appropriate links e. Add new entity to primary process table f. Create mail box for Inter Process Communication.	By being established - Root ($H\bar{e}tu$) Receiving job as an input - Object ($\bar{A}rammana$), Presence ($Atthi$) Space, Mail box – Support ($Nissaya$), Space, Data - Presence ($Atthi$) Presence ($Atth$i) Presence ($Atthi$) New entity- Post-Nascence ($Pur\bar{e}j\bar{a}ta$)

2. New → Ready New to Ready is a modified transition by SSPM, occurs when a newly created process is organized and loaded into the memory. Next, the set of actions derived from the causal relations are listed in the next column.	1. Send the highest priority process cluster in the ready queue for processing. 2. Organize: a. Classify the process into a cluster with related data, program/s, and processes. (e.g. Process Image) – Identify whether the process is I/O or processor bound. b. Merge Relevant Clusters (Use procedures/ System calls/ parameters) c. Do necessary Prioritizations. d. Add to the Ready queue. 3. Activate by loading into the main memory.	Process Cluster-Pre-dominance ($Adhipati$) Space- Presence ($Atthi$) Co-nascence ($Sahaj\bar{a}ta$), Presence ($Atthi$), Presence ($Atthi$) Association ($Sampayuktha$) Mutuality ($A\tilde{n}\tilde{n}ama\tilde{n}\tilde{n}a$) Merge- Association ($Sampayuktha$), Mutuality ($A\tilde{n}\tilde{n}ama\tilde{n}\tilde{n}a$) Pre-dominance ($Adhipati$) Non-disappearance ($Avigata$), Pre-nascence ($Pur\bar{e}j\bar{a}ta$)
3. Ready → Ready This is a newly introduced transition by SSPM, it does organizing using the actions as aforementioned.	2. Re-organize: a. Delete unnecessary data, program/s and processes. b. Organize: i. Classify the process into a cluster with related data, program/s, and processes. (e.g.: Process Image) – Identify whether the process is I/O or processor bound. ii. Merge Relevant Clusters (Use procedures/ System calls/ parameters) iii. Do necessary Prioritizations. iv. Add to the Ready queue. 3. Activate by loading into the main memory.	Repetition ($\bar{A}s\bar{e}vana$) Delete-Absence of any condition Co-nascence ($Sahaj\bar{a}ta$), Presence ($Atthi$), Non-disappearance ($Avigata$)Presence ($Atthi$) , Association ($Sampayuktha$), Mutuality ($A\tilde{n}\tilde{n}ama\tilde{n}\tilde{n}a$) Merge- Association ($Sampayuktha$), Mutuality ($A\tilde{n}\tilde{n}ama\tilde{n}\tilde{n}a$) Pre-dominance ($Adhipati$) Non-disappearance ($Avigata$), Pre-nascence ($Pur\bar{e}j\bar{a}ta$)
4. Ready → Running This is an existing transition.	1. Move the currently executing cluster out. (Preemptive/Non-Preemptive) a. Expiring the allowed time duration. (Time-out) – Clock interrupt. b. Blocking due to another interrupt. – I/O interrupts, Operating system calls, Signals c. Entering a higher priority cluster into the ready queue. 2. Dispatch (short-time scheduling)	Disappearance ($Vigata$) Time-Disappearance ($Vigata$) Interrupts-Presence ($Atthi$) Pre-dominance ($Adhipati$), Presence ($Atthi$)

	a. Switching context –save the status of the old process in its PCB and load the status of the new process. b. Switching to user mode (to execute the user process) c. Jumping to proper location in the user program	Proximity ($Anantara$)
5. Running → Ready This is an existing transition.	1. Time-out: expires the allocated time limit. Major concerns: Availability of Space, Level of priority	Space, interrupt, process cluster- Presence ($Atthi$) Process cluster- Pre-dominance ($Adhipati$) Time, running process cluster- Disappearance ($Vigata$)
6. Running → Blocked This is an existing transition.	1. Interrupt the currently executing process. a. Request made by the currently executing process is not responded promptly. (service calls/ resources are not available) b. Initiate an action (child process) which is to be completed before continues. c. Delays in Inter Process Communications (IPC) – data/ message. (Interrupts: Program: as a result of instruction execution-ex: division by zero, Timer: to perform in regular basis I/O: to signal normal completion or error H/W failure)	Interrupt- Presence ($Atthi$), Running process- Disappearance ($Vigata$)
7. Blocked → Ready This is an existing transition.	1. Event occurs	Event- Presence ($Atthi$), Post-nascence ($Pacchaj\bar{a}ta$) Process- Pre-nascence ($Pur\bar{e}j\bar{a}ta$)
8. Running → Sleep This is an existing transition.	1. Complete the currently executing process 2. Release the processor.	Current process- Disappearance ($Vigata$) Processor, Next process- Presence ($Atthi$)
9. Sleep → Ready This is also a newly introduced by SSPM for the	1. Re-organize: a. Delete unnecessary data, program/s and processes. b. Organize:	Repetition ($\bar{A}s\bar{e}vana$) Delete- Absence of any condition

purpose of reorganizing and reusing executed processes.	i. Classify the process into a cluster with related data, program/s, and processes. (e.g. Process Image) – Identify whether the process is I/O or processor bound. ii. Merge Relevant Clusters (Use procedures/ System calls/ parameters) iii. Do necessary Prioritizations. iv. Add to the Ready queue. 2. Activate by loading into the main memory.	Co-nascence (*Sahajāta*), Presence (*Atthi*), Presence (*Atthi*), Association (*Sampayuktha*), Mutuality (*Aññamañña*) Merge- Association (*Sampayuktha*), Mutuality (*Aññamañña*) Pre-dominance (*Adhipati*) Non-disappearance (*Avigata*), Pre-nascence (*Purējāta*)
10. Ready → Terminate This is the last newly introduced transition which occurs due to absence any of the causal relations.	1. Absence of every cause or condition such as input, event, actions, all necessities for re-organizations, empty ready queues, empty blocked queues. (No further improvements or executions)	

Please note that Six-state continuous processing system is so-called due to the six states 'New,' 'Ready,' 'Running,' 'Blocked,' 'Sleep', and 'Terminate'. Then, there are ten transitions between these states. For example, the moving a process from 'New' state to 'Ready' state, which is denoted by 'New -> Ready', is a transition. Such away, there are six states and ten transitions.

Introducing the model such a way, the coming section illustrates the implementation of the proposed model.

4.9 Implementation

The earlier sections have discussed the new approach towards the new computing model and the model itself. Then, this section has reported the implementation or the customization details of the proposed computing model, the SSPM. For the demonstration, testing and evaluation purposes the SSPM customized a Fraction Calculator (SSPM-FC). Other than the SSPM-FC, the model also customized a Sorting Program (SSPM-Sorting), Quadratic Equation Solver (SSPM-QES) and a Simulated Process Scheduler (SSPM-PS). These programs were specially selected to be customized by the SSPM, because those can be executed through continuous processing over program execution cycles without making an overhead for processing. All the important source codes are stored in the Appendix A.

4.9.1 Fraction Calculator (SSPM-FC)

The SSPM-FC has been implemented with Java in Netbeans 8.1 integrated development environment (IDE). Further, MySQL has been used with Notepad++ to record time values. Furthermore, SSPM-FC has facilitated to calculate any fractional expression with +, -, *, and d (division). It can work well with simple expression as well as complex expressions such as expressions with brackets and whole numbers. Longer expressions also can be computed and it can continuously do its processing. It has been possible to find some examples as follows.

```
6/7*8/9
3/5-2 5/7d1/9*(2/3+12/17)
```

Then, the following section has described the reasons behind selecting particular technologies.

4.9.1.1 Why use the technologies

First, it discusses why those technologies have been selected for this implementation from a large basket of such other technologies. The following sub sections discusses why it has been selected the technologies Java, Netbeans and MySQL in the implementation process.

4.9.1.1.1 Java

First and foremost, Java has been free and has a large pool of open source libraries. Further, the Java has been an object oriented programming language, which has provided flexibility, modularity and extensibility. Moreover, it has supported for potential development tools, such as Netbeans and Eclipse.

4.9.1.1.2 Netbeans

The Netbeans also a free and open source integrated development environment. Further, the Netbeans IDE has provided user friendly, rapid user-interface development, and fast and smart code editing environment with efficient project management facility. In addition to that it supports for many plugins.

4.9.1.1.3 MySQL

This has been an open source database management system software, which allows adding, accessing, querying and managing content easily. Further, it has allowed processing quickly, with guaranteed reliability. Moreover, it has provided a connector for Netbeans. Next section explains how this SSPM-FC has been implemented.

4.9.1.2 The Implementation Process

During this process, it has been focused on how the features of the six-state continuous processing model can be implemented. As mentioned in earlier chapters, the model has three features such as internal and external processes, continuous processing and evolving tactics memory. All these have to be combined with the following set of tactics.

1. Recognize the incoming inputs and pertinent operations.
2. Create relevant header or library files for new operations.
3. Do necessary classifications with inputs and operations.
4. As necessary do job prioritizations for the execution.
5. Create modules for frequent operations.
6. Delete inefficient or unnecessary items.

The Figure. 4.5 has displayed the flow of the proposed model integrated Fraction Calculator. However, it should be noted that the system has been more complexed than this high level diagram and the attempts to improve the autonomy could have increased the complexity more.

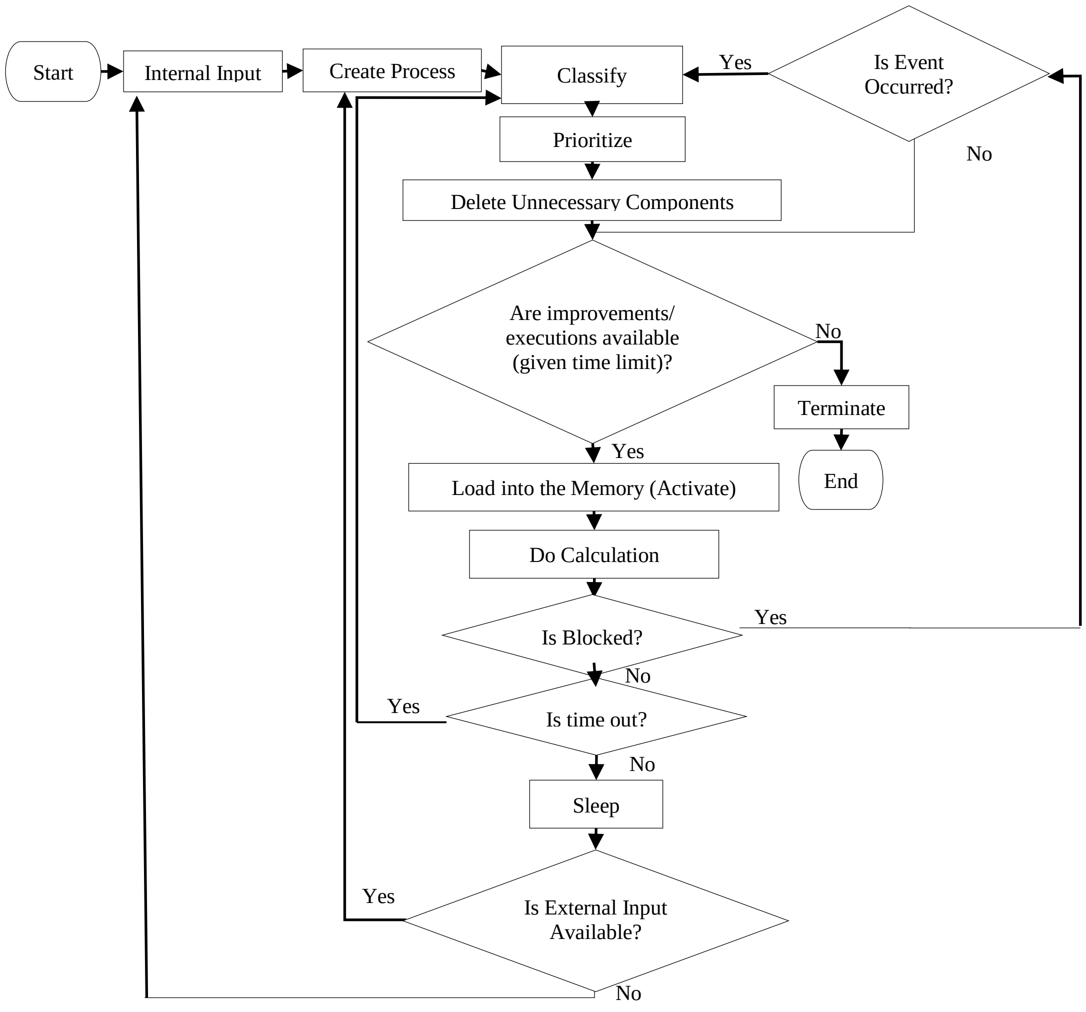


Figure. 4.5: High level flow chart for the proposed model

This model has been implemented under five major modules namely, Input Analyser, Process Switcher, Operation Organizer, Write Engine, and the Small Compiler. The diagram that has linked these modules together are displayed in the Figure. 4.6.

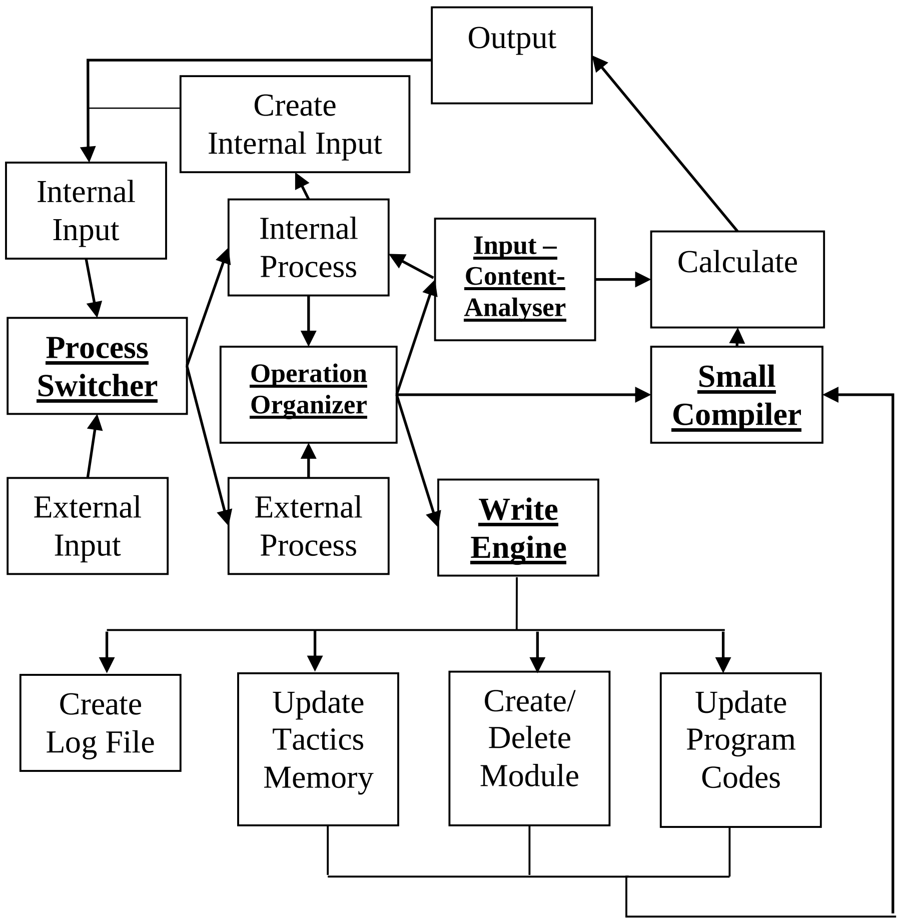


Figure. 4.6: How the modules have been connected

Further, the pseudo codes for the algorithm has been showed in the Figure. 4.7. (a), (b), (c), (d), (e), and (f).

(a)

```
Fraction Calculator
  While (true) do
      Input Expression(Internal/External);
      Call Input-Content Analyser;
      Call Process Switcher(Internal/External);
      Call Operation Organizer;
      Calculate;
      Display Output;
  End-while
End
```

(b)

```
Input-Content Analyser
   Get expression;
   While (not end of expression) do
       Split the expression;
       Find Braces;
       Find Operations;
       Find Fractional Operands;
   End-while
End
```

(c)

```
Process Switcher
   Get Input
   If (category of input = internal)
       Internal Process;
       Create Internal Input;
   Else If (category of input = external)
       External Process;
   End-If
End
```

(d)

```
Operation Handler
   Do
       If (ready)
           Execute process – Calculate;
           Display Results;
       else
           Call Write Engine;
           Call Compiler;
       End-If
   Until no execution or improvement
End
```

(e)

```
Write Engine
   Create/ Delete Module
   Update Codes
   Create Log
   Update Tactics Memory if necessary
End
```

(f)

```
Small Compiler
  Get new  changes (created or deleted modules)
  Compile;
      Set Class Path;
      Create Class Loaders;
      Load Classes;
End
```

Figure. 4.7: Algorithm for Fraction Calculator (SSPM-FC). (a) Fraction Calculator as a whole, (b) Input-Content Analyser, (c) Process Switcher, (d) Operation Handler, (e) Write Engine, (f) Small Compiler.

The coming section discusses the how the QES was customized according to the SSPM model.

4.9.2 Quadratic Equation Solver (SSPM-QES)

The SSPM-QES program development process was similar to how the SSPM was set in the FC, except its Input-Content Analyser, internal input creation and the calculation, where this QES was focused on solving quadratic equations.

4.9.3 Sorting Program (SSPM-Sorting)

The sorting program based on Quicksort customized by the SSPM. In fact, the Quicksort is highly efficient algorithm for sorting and is based on partitioning the data set into two subsets. In Quicksort, the original list of elements is divided into two sub lists, one of which holds values lower than a selected particular value, the pivot, depending on which the division is made and the other list holds values higher than the pivot value.

Further, it was concentrated on comparing this system with the quicksort programs customized by some other computing models such as parallel computing, ADAPTON, Self-Adjusting computing, Dynamically Tuned Library and Evolutionary computing. This development process was also similar to that of the above SSPM-FC, except its Input-Content Analyser, internal input creation and the calculation as it had work with lists of elements, with respect to the relevant computing techniques. However, it had to do comparisons with the lists of larger number of inputs, it has been used internal process case more. Specially, in the scenario, which was tried to compare the model with the Dynamically Tuned library and the Evolutionary computing, the standard deviation and the distribution of data were also mattered. Therefore, when creating the internal scenario, rather than creating lists with random numbers, it was required to write the code so to create normally distributed data with fixed standard deviation. In contrast to the FC and QES, The SSPM-Sorting program has been implemented so to support all the input patterns mentioned in the above section 4.3. Further, this had the techniques such as Insert for *IP1*, Equal for *IP4*, delete for *IP3*, and delete and insert or sort for *IP2*. However, with FC, it couldn't implement the *IP1*, and *IP3*, and with QES, only the Frequent Module creation and *IP4* were implemented.

Note:

1. $IP1\colon X \subset Y,\quad IP2\colon X \nsubseteq Y,\ Y \nsubseteq X \Rightarrow X \cap Y \neq \phi \ or\ X \cap Y = \phi,\quad IP3\colon Y \subset X,$ $IP4\colon X = Y \neq \phi$
2. In the section 4.3, it was supposed that *X* and *Y* were sets as it was easy to illustrate the idea in the identification of inputs assuming that no repeated elements are there. However, during the implementation process in Java, ArrayLists were used and repeated elements were handled accordingly, where the removal or insertion of repeated elements were explicitly handled as per the arising conditions also using the containAll(), retainAll(), and removeAll() methods.

4.9.4 SSPM-PS

This has been a simple example of a simulated process scheduler that has been implemented using Turbo C programming environment and displays the ability to evolve over processing cycles. The particular algorithm has been illustrated in the Figure. 4.8.

```
Simulated Process Scheduler
  While (True) do
      Input Processes, Scheduler Algorithms;
      Select FCFS;
      Do
          Execute Processes;
          Record Waiting Times;
          Record Burst Times;
          Calculate Turnaround Times;
          Calculate Average Waiting Time;
          Calculate Average Turnaround Time;
          Select Scheduler Algorithm;
      Until execution or improvement are available
  End-while
End
```

Figure. 4.8: Algorithm for Simulated Process Scheduler

Particularly, this was done for the simulation purpose. This was the very first example, which has been implemented to show the evolving ability in a program. Initially, implementations were started with the C Language. However, it has been moved from C language to the Java language for the straightforward implementation for rest of the programs.

4.10 Experimental Mechanism of SSPM

As stated above the hypothesis of this research is "*the processing power of the computer can be enhanced with the support of a smaller tactics memory, which improves as a result of continuous processing*." This has been used as the statistical hypothesis for the statistical analyses in each testing scenario, as follows.

H_0: There is no change among the means of time values collected before and after organizing the smaller tactics memory through continuous processing.

(i.e. the system gained no performance enhancement over generations of program executions)

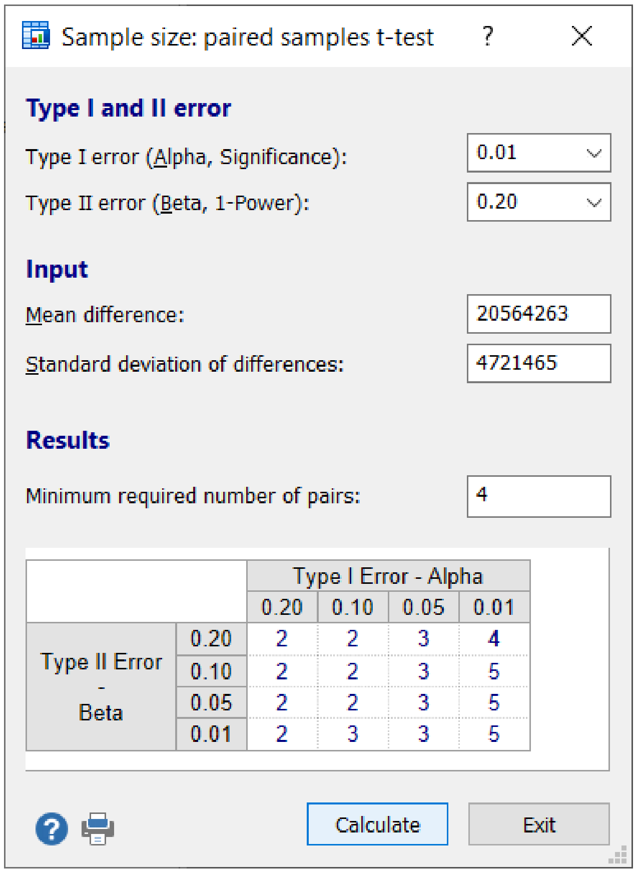
(H_0: $\mu_D = d_0$, d_0=0)

H$_1$: The mean value of the time values collected before organizing smaller tactics memory is greater than the mean value of the time values collected after organizing smaller tactics memory.

(i.e. the system gained a performance enhancement over generations of program executions)

(H_1: $\mu_D > d_0$, d_0=0)

Here, a set of arbitrarily selected inputs, was used for the execution. Then, the execution times were recorded for each input before and after organizing the smaller tactics memory as it was required to check whether the system has gained a considerable improvement due to the modification. Therefore, this kind of study is called as a crossover study [129]. Further, the recommended statistical tests for doing this kind of crossover analysis are the Sign test, Wilcoxon Signed Rank test, and Paired-t Test [130]. Nevertheless, the paired-t test is the most powerful test among these [130], [131], that test can be applied only under three conditions. These three conditions are: the data should be in continuous scale, the set of differences of paired time values should have a normal distribution [132], and the set of differences of paired time values should have no outliers [131]. In fact, the Wilcoxon Signed Rank test must be applied only if the samples seriously violates these three conditions [130], and the samples have a symmetric continuous distribution [133]. Further, if the samples satisfy none of these conditions, then the Sign test [132], [133] can be applied. However, The latter two non-parametric tests have lower power than the paired-t test and have been considered as alternatives [130] for the paired-t test. In addition to these, the power test has been applied using Minitab 17 for the sets of time values collected in each scenario to prove the suitability of applying the statistical test (paired-t test) with the given sample size. Moreover, to apply paired-t test, the sample size become immaterial, if the sample has a normal distribution [132]. This is also a good justification for applying paired-t test with different sample sizes for different mean differences. Therefore, the selected sample sizes do not harm the results of the statistical analysis. In some cases, such as testing scenario 2 in SSPM-FC and testing scenario of SSPM-QES, and testing scenario 1.1 of the SSPM-Sorting, the sets of time values reported here are the populations that could be recorded having minimum hardware and software disturbances such as heap size. Further, it could use MedCalc which is a statistical software to justify the selection of sample sizes in each case. For example, as shown in Figure 4.9, the required minimum sample size was 4 for the testing scenario 1.1 of SSPM-FC as it has a good power for applying paired-t test, when considering its mean difference. Here, the beta level was set as 0.02, i.e. 80% power and it is a higher value.



Sample size: paired samples t-test

Type I and II error

Type I error (Alpha, Significance):	0.01
Type II error (Beta, 1-Power):	0.20

Input

Mean difference:	20564263
Standard deviation of differences:	4721465

Results

Minimum required number of pairs:	4

		Type I Error - Alpha			
		0.20	0.10	0.05	0.01
Type II Error - Beta	0.20	2	2	3	4
	0.10	2	2	3	5
	0.05	2	2	3	5
	0.01	2	3	3	5

Figure 4.9: Sample size determination using MedCalc for testing scenario 1.1 of SSPM-FC

4.11 Simulation of SSPM in a Turing Machine

The SSPM has been simulated in a Turing Machine to prove the real-world applicability in theoretical level. A detailed description on this regard has been included in section 6.11.

4.12 Summary

This chapter has started with introducing the based hypothesis, which has been focused in improving the efficiency of computing. Next, it discussed about the inputs for the proposed processing model, while the next section has described the output of the processing model. Meanwhile, the section 4.5 presented the six-state continuous processing model, the section 4.6 has discussed its features, later discussing about the users of the system. Further, it has been discussed how a set of twenty-four CRs in BTM has been exploited in describing the thought-process, and how a set of fifteen CRs has been exploited in designing the six-state continuous processing model. Finally, this chapter has discussed about how the six-state continuous processing model (SSPM) has been implemented in SSPM-FC, SSPM-QES, SSPM-Sorting, and SSPM-PS. Further, this has mentioned SSPM was simulated with a Turing Machine. In addition to that, the reasons for using the respective technologies.

Then, the Chapter 5 will illustrate how the system works under the four examples, highlighting the features of the proposed system.

CHAPTER 05
HOW THE SYSTEM WORKS

5.1 Introduction

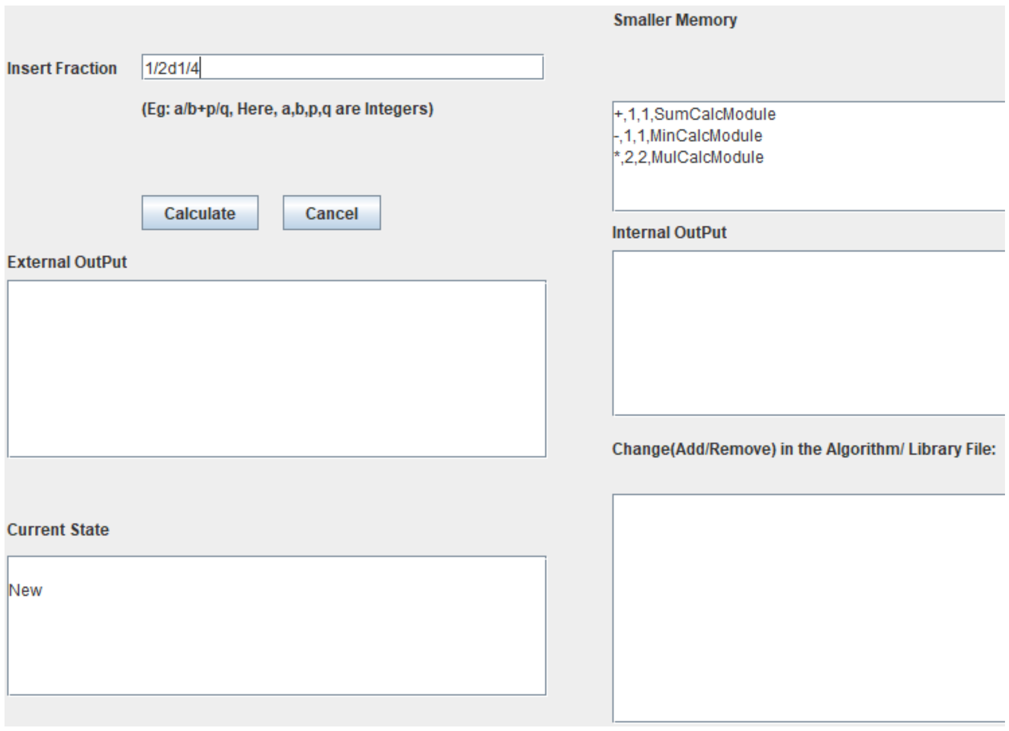
The fourth chapter presented the methodology of this research, which described the proposed model, how BTM has been exploited in modelling, design and customizing some existing programs with the Six-state continuous processing model, the SSPM. This chapter has explained how the SSPM model behave. Further, several working scenarios of the systems customized by SSPM have been described under four sections, such as SSPM-FC, SSPM-QES, SSPM-Sorting, and SSPM-PS in order to illustrate the major characteristics of the SSPM and the ability of SSPM and the smaller tactics memory to gain improvements over subsequent program execution cycles. In each case, the time taken for processing each of the input has been recorded for the evaluation purpose and it should be noted that a detailed report of the evaluation of the performance gained by each customized case after the modification is included in the chapter 06.

5.2 Fraction Calculator (SSPM-FC)

This section discusses how the SSPM-FC produced internal and external outputs, and how the smaller tactics memory as well as the processing were improved with the time. By customizing the FC with the SSPM, the SSPM-FC was introduced. The SSPM-FC computes the expressions of fractions and produce results. In fact, the fractions have three categories proper fractions (denominator is less than the numerator), improper fractions (denominator is greater than the numerator), and mixed fractions (consists of whole part as well). The input for this SSPM-FC consist of the fractions belonged to one of the above three categories. Further, there could be bracketed sub-fractional-expressions and the operators used in the system is multiplication, division, subtraction, and additions. Although, there were no limitation was imposed on the expression length. If the input is lengthy, two fractions were processed at a time with the aforementioned binary operations and produced a partial fractional output and finally produced the final output.

Like the human mind, there are two processes in the SSPM-FC such as *internal process and external process*. At the beginning, the SSPM-FC begins with an internal process resulting from an internal expression generated through the system by choosing an operation randomly from the current operations list and arbitrary fraction array. Then, the internal results are produced. The external mode of SSPM-FC was triggered due to the fractional expressions inserted externally by the user. The input has been an expression of two or more fractional operands operated on +, -, *, d (division). This can be done only when the internal process has been in 'Sleep' state. In this case the computation results can be seen by the users. The system returns to the internal process, in the absence of an external process. Here, the internal fractional expression is generated again through the operators that were included in the most recent external or internal expression. Such a way as per the arising conditions through internal and external processes the *continuity* is maintained.

In this system, initially, the *smaller tactics memory* has been filled with the entries related to the instructions that are already inserted to the system by the developers and starts with an internal input. As the system continuously does processing, the smaller tactics memory has been evolved due to the tactics apply on the system. Further, the smaller tactics memory contains a set of records, which correspond to the operators that are using. For instance, the entry (+,1,1,SumCalcModule) is the corresponding record of the plus operator as seen in Figure. 5.1.



1/2d1/4

(Eg: a/b+p/q, Here, a,b,p,q are Integers)

```
+,1,1,SumCalcModule
-,1,1,MinCalcModule
*,2,2,MulCalcModule
```

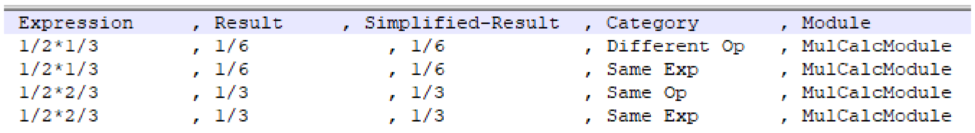
New

Figure. 5.1. Smaller tactics memory starts with the entries relevant to the instructions that are inserted into the system

The continuous processing of the SSPM-FC is done through the smaller tactics memory by using both the external and internal processes. Throughout these processing, a set of actions was taken place. The performance of the SSPM-FC was improved and the smaller tactics memory is updated over program execution cycles due to these actions. The set of actions of the SSPM are concisely explained as follows;

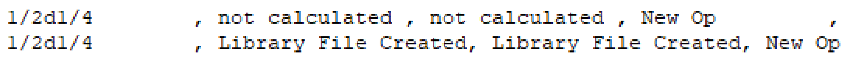
- The current input and the respective operators are identified:
  The input expression is analysed and then classified considering the fractions and the operators in the input as seen in the Figure. 5.2. (a) and (b):

(a)



```
Expression       , Result      , Simplified-Result  , Category         , Module
1/2*1/3          , 1/6            , 1/6              , Different Op     , MulCalcModule
1/2*1/3          , 1/6            , 1/6              , Same Exp         , MulCalcModule
1/2*2/3          , 1/3            , 1/3              , Same Op          , MulCalcModule
1/2*2/3          , 1/3            , 1/3              , Same Exp         , MulCalcModule
```

(b)



```
1/2d1/4        , not calculated , not calculated , New Op           ,
1/2d1/4        , Library File Created, Library File Created, New Op
```

Figure. 5.2. Identify the input (a) Different Operator, Same Expression, Same Operator, (b) New Operator

1. A fractional expression that includes a new operation: Input contains a currently unavailable operator in the smaller tactics memory or the knowledgebase/ instruction set. As seen in the Figure. 5.1 no entry for division in the smaller tactics memory as its instruction is not inserted into the system.

2. A fractional expression that includes a different operation: Here, the current operation is dissimilar to the most recently executed operation. (the fractions in the current expression can be similar or dissimilar to the fractions in the most recently executed expression)

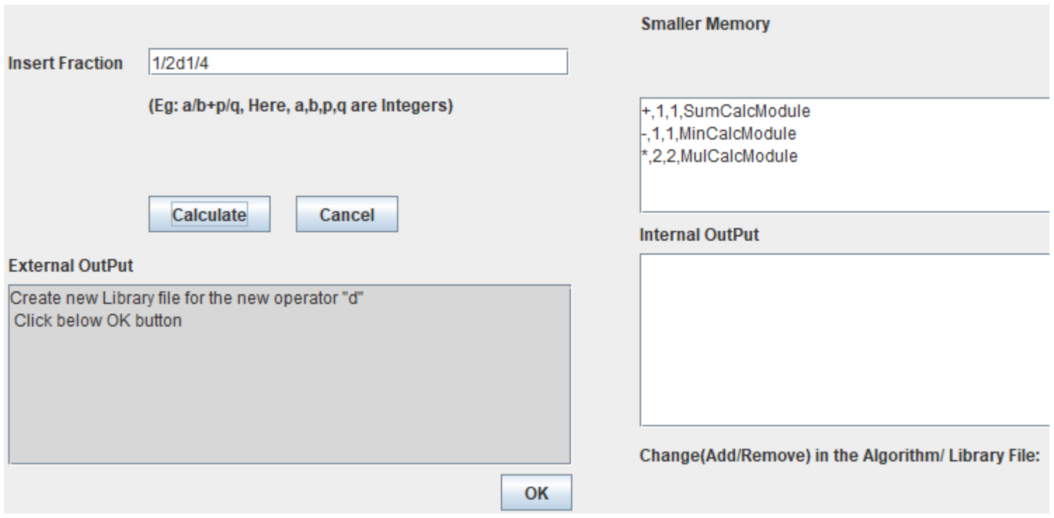
3. A fractional expression that includes the same operator: The operator in the current expression is alike the operator in the most recently processed input, but the set of fractions in the expressions are dissimilar.

4. Same fractional Expression: The most recently executed expression is exactly equal to the current expressions. (All the array of operators and the arrays of denominators and numerators in the input are similar in the order.)

All the operators and the fractions in an input can be separately listed and can be categorised according to the explanations given in the above section 4.3 in the Input regard.

- The libraries for novel operations are added.

  If current fractional expression contains a new operator (d) as seen in the Figure. 5.3, the particular instructions set that can perform the operation is added to the system through the given interface as a library file DivCalcModule.java as shown in the Figure. 5.4. (a) and (b). Routinely, a respective entry on the operator (+,1,2,DivCalcModule) is then recorded in the smaller tactics memory as displayed in the Figure. 5.5.



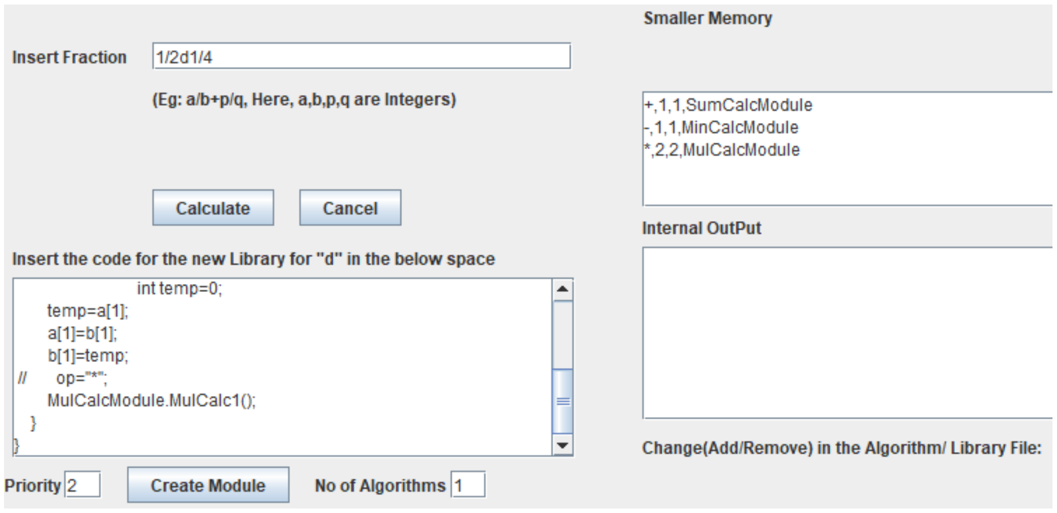
```
1/2d1/4
(Eg: a/b+p/q, Here, a,b,p,q are Integers)

+,1,1,SumCalcModule
-,1,1,MinCalcModule
*,2,2,MulCalcModule

Create new Library file for the new operator "d"
 Click below OK button
```

Figure. 5.3. Current Fractional Expression contains a New Operator

(a)



Insert Fraction: 1/2d1/4

(Eg: a/b+p/q, Here, a,b,p,q are Integers)

```
+,1,1,SumCalcModule
-,1,1,MinCalcModule
*,2,2,MulCalcModule
```

Insert the code for the new Library for "d" in the below space

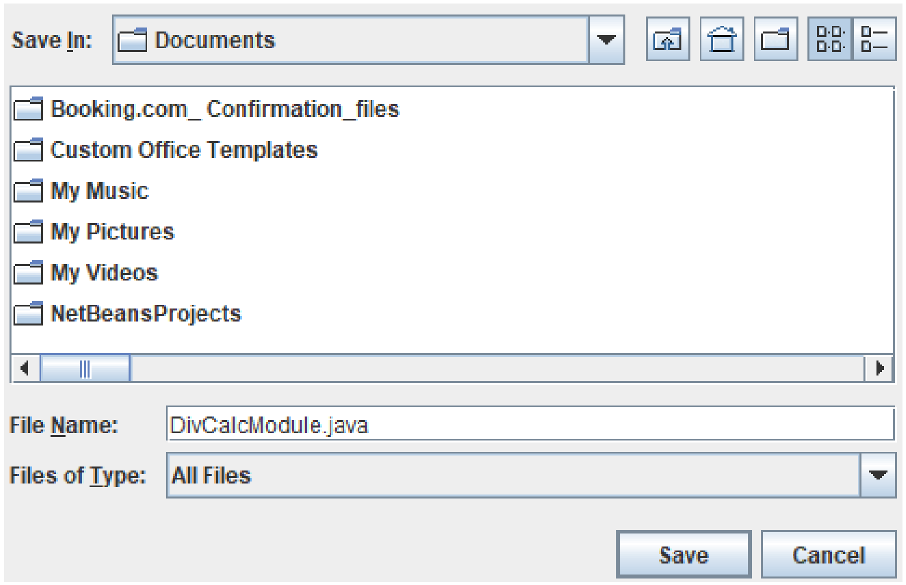
```
                int temp=0;
    temp=a[1];
    a[1]=b[1];
    b[1]=temp;
//     op="*";
    MulCalcModule.MulCalc1();
  }
}
```

Change(Add/Remove) in the Algorithm/ Library File:

Priority: 2

No of Algorithms: 1

(b)



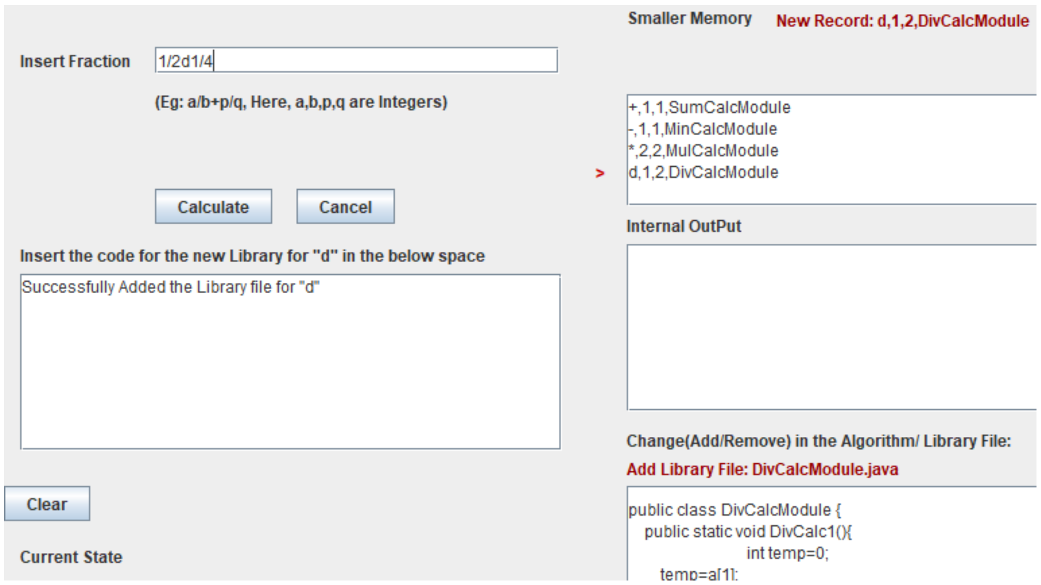
Save In: Documents

```
Booking.com_ Confirmation_files
Custom Office Templates
My Music
My Pictures
My Videos
NetBeansProjects
```

File Name: DivCalcModule.java

Files of Type: All Files

Figure. 5.4. (a) Insert the relevant instruction set for the new operator. (b) create and save the library.


Smaller Memory
New Record: d,1,2,DivCalcModule
Insert Fraction
(Eg: a/b+p/q, Here, a,b,p,q are Integers)
Calculate
Cancel
Internal OutPut
Insert the code for the new Library for "d" in the below space
Change(Add/Remove) in the Algorithm/ Library File:
Add Library File: DivCalcModule.java
Clear
Current State

1/2d1/4

```
+,1,1,SumCalcModule
-,1,1,MinCalcModule
*,2,2,MulCalcModule
d,1,2,DivCalcModule
```

Successfully Added the Library file for "d"

```
public class DivCalcModule {
   public static void DivCalc1(){
                  int temp=0;
     temp=a[1];
```

Figure. 5.5. Update the smaller tactics memory with an entry for the instruction set of the new operator

Another examples: if no entry was recorded in the smaller tactics memory for the Addition, when an expression with the plus operator has inserted, the SSPM-FC defines the current input as an input with a new operation. Next, the SSPM-FC gives the space to the user to add the set of instructions that can perform the fractional addition as a library file. Further, a respective entry for the plus operator is recorded in the smaller tactics memory with the priority, and the number of alternative methods that have been stored in the library file, which can perform the same addition operation. The particular entry in the smaller tactics memory is (+,1,1,SumCalcModule).

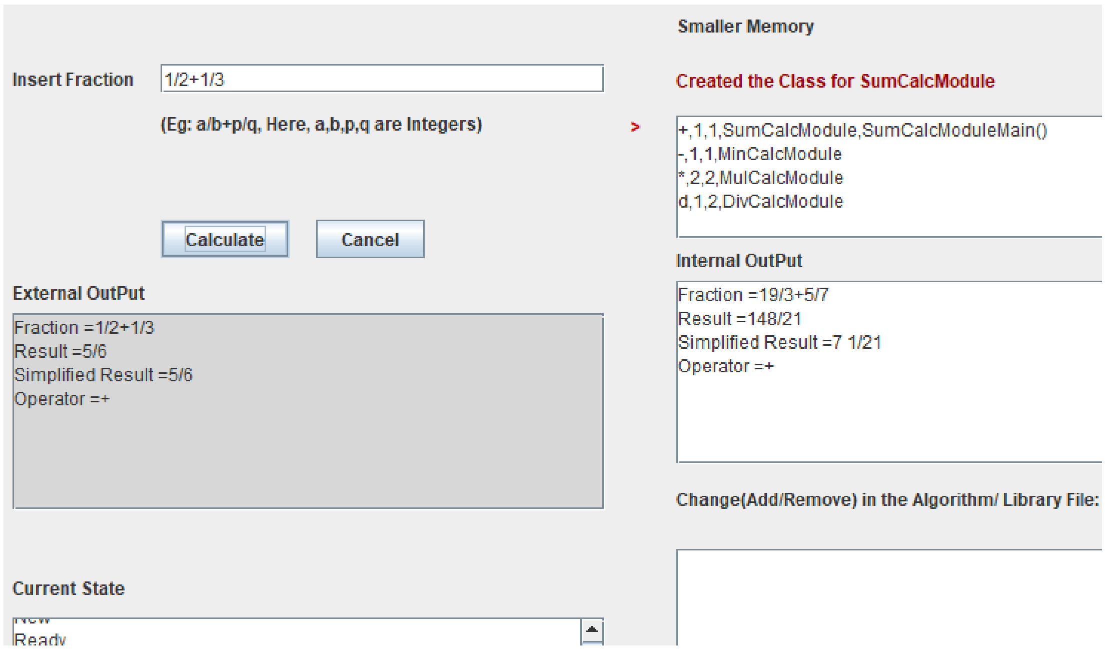
- The operations are classified together with the relevant instructions and data, and the classes were prioritized, for the execution.

  When, a fractional expression is inserted, the SSPM-FC dynamically calls relevant methods in the library files with respect to the operator/s in the fractional expression and generates a module. Then, SSPM-FC dynamically compiles it and compute the fractional expression to the allocated priorities.

- The operating modules are created for the frequently executing operations.
  The SSPM-FC initially begins with a specific file called “SubMain.java”. Depending on the operators in the inserted fractional expression, the content of this java file is modified dynamically by calling relevant methods in appropriate libraries. It should be noted that the separate modules for all the operations were not saved as their nature of the execution has not been recognized at the beginning. Throughout the SSPM-FC execution cycles, it recognizes the frequency of the invocation of each method in the library file and creates operating modules for the operations that are executed more frequently.

  Stated in another way, unless an operator is frequently applied, the system does not save respective operating modules, but creates them dynamically. if the same operator is computed throughout several program execution cycles, then the respective operator is marked as a frequent operator and its operating module is permanently stored as shown in Figure 5.6 (b). Furthermore, the 4-tuple entry of the particular operator in the smaller tactics memory is updated to a 5-tuple entry with the name of the module as shown in Figure. 5.6 (a). Therefore, gaining the straightforward access to the respective module through the smaller tactics memory.

(a)



Smaller Memory

Insert Fraction 1/2+1/3

(Eg: a/b+p/q, Here, a,b,p,q are Integers)

Created the Class for SumCalcModule

```
+,1,1,SumCalcModule,SumCalcModuleMain()
-,1,1,MinCalcModule
*,2,2,MulCalcModule
d,1,2,DivCalcModule
```

External OutPut

```
Fraction =1/2+1/3
Result =5/6
Simplified Result =5/6
Operator =+
```

Internal OutPut

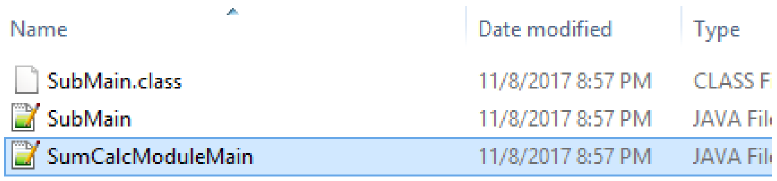
```
Fraction =19/3+5/7
Result =148/21
Simplified Result =7 1/21
Operator =+
```

Change(Add/Remove) in the Algorithm/ Library File:

Current State

```
New
Ready
```

(b)



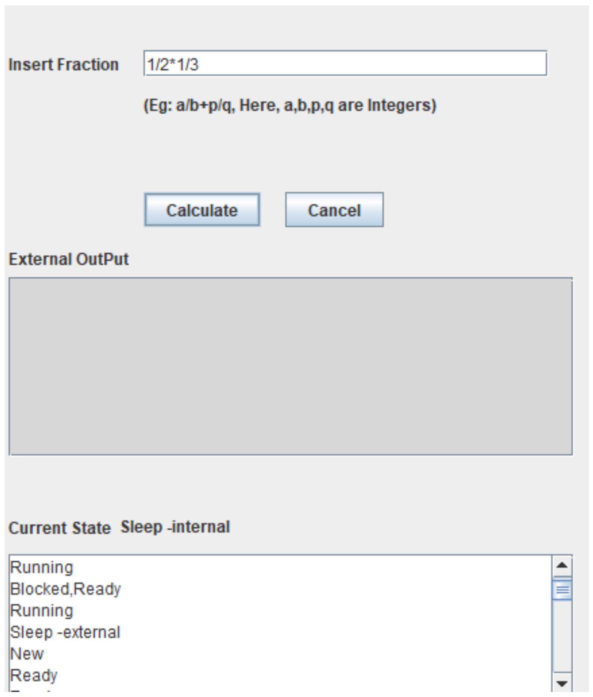
Name	Date modified	Type
SubMain.class	11/8/2017 8:57 PM	CLASS F
SubMain	11/8/2017 8:57 PM	JAVA Fil
SumCalcModuleMain	11/8/2017 8:57 PM	JAVA Fil

Figure. 5.6. For the frequently executing plus operator, the module is created. (a) The respective entry is updated to a 5-tuple in the smaller tactics memory, (b) File of the particular module is created and stored

- Irrelevant or inefficient instructions or modules, or irrelevant data are deleted.
  A single operation can have pool of different computing methods. In such a case, over program execution cycles, SSPM-FC finds the most efficient method and retain it eliminating the other inefficient computing methods in the pool.

  For this scenario, an example has been drawn from the multiplication computation. At the beginning, two alternative computing methods were there for the multiplication, namely, MulCalc2() and MulCalc1() (Appendix A.7). The SSPM-FC selects the most efficient method among the two methods, by analysing the recorded nanoseconds values in each processing cycle executing the inputs via both the computing methods. Then, the inefficient method is eliminated by the SSPM-FC, and respective entry in the smaller tactics memory is updated by reducing the number of alternative methods of multiplication operator from two to one.

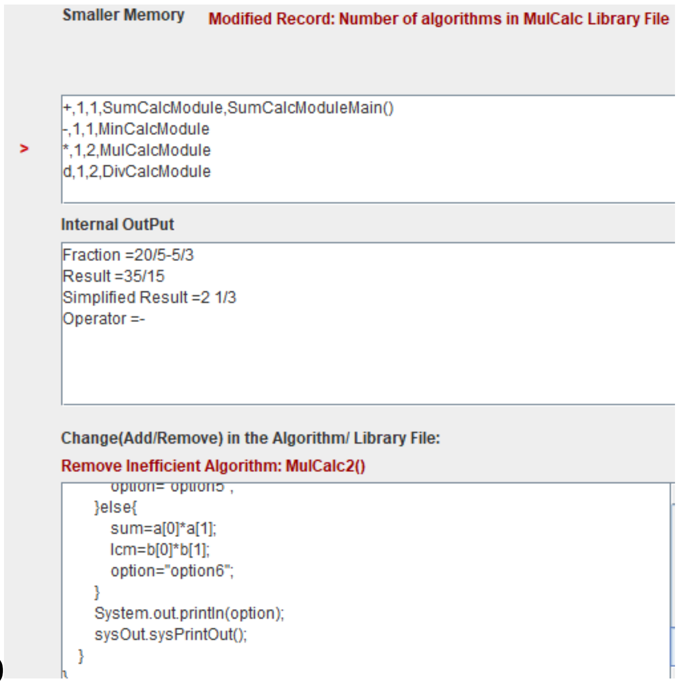
Moreover, the system can delete unnecessary instructions or modules that are no longer required for the processes in the SSPM-FC. However, the decision for this elimination is currently taken, when the respective instructions or modules have not been triggered within a given period of time. If so, the particular entry in the smaller tactics memory also get deleted automatically as showed in Figure. 5.7. Initially, for the Multiplication two methods (+,2,2,MulCalcModule) were there, then the number of methods in the entry of multiplication was reduced to one (+,1,2,MulCalcModule).

(a)


```
1/2*1/3
```

```
Current State  Sleep -internal
```

```
Running
Blocked,Ready
Running
Sleep -external
New
Ready
```

(b)

Smaller Memory
Modified Record: Number of algorithms in MulCalc Library File
Internal OutPut
Change(Add/Remove) in the Algorithm/ Library File:
Remove Inefficient Algorithm: MulCalc2()

```
+,1,1,SumCalcModule,SumCalcModuleMain()
-,1,1,MinCalcModule
*,1,2,MulCalcModule
d,1,2,DivCalcModule
```

```
Fraction =20/5-5/3
Result =35/15
Simplified Result =2 1/3
Operator =-
```

```
        option="option5";
    }else{
        sum=a[0]*a[1];
        lcm=b[0]*b[1];
        option="option6";
    }
    System.out.println(option);
    sysOut.sysPrintOut();
  }
```

Figure. 5.7. Update the relevant entry in the smaller tactics memory due to deleting the inefficient method (a) How the states are changing, (b) How the smaller tactics memory and the program update

Sample Scenario:

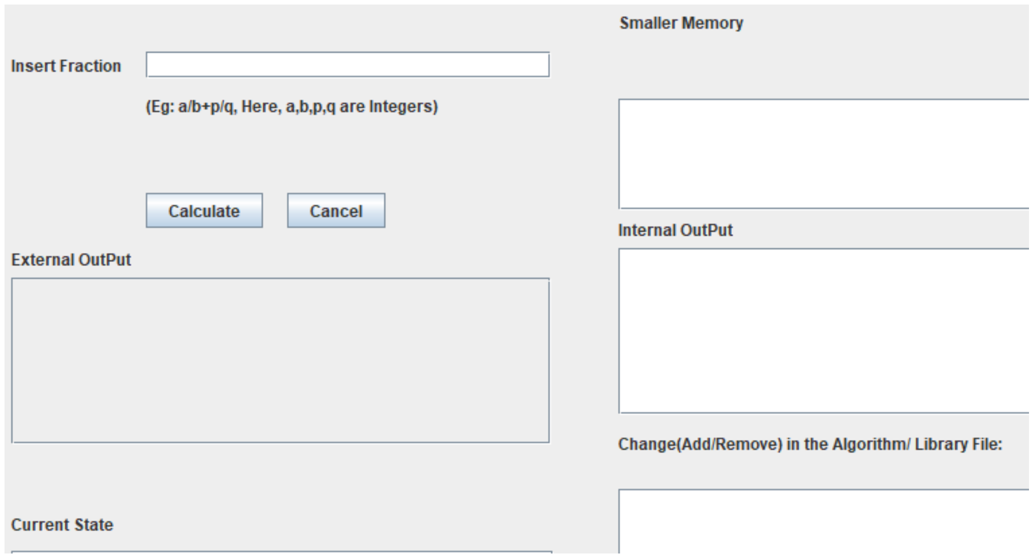

Smaller Memory
Insert Fraction
(Eg: a/b+p/q, Here, a,b,p,q are Integers)
Calculate
Cancel
Internal OutPut
External OutPut
Change(Add/Remove) in the Algorithm/ Library File:
Current State

Figure. 5.8: User interface with all the details

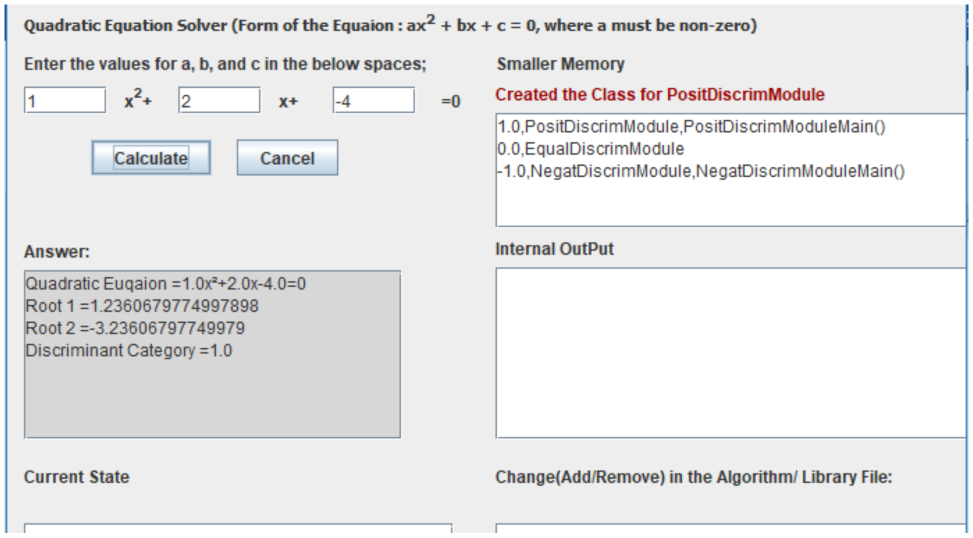
Once, the system starts to get expressions such as ½ + ½ as inputs to the system through the interface showed in the Figure. 5.8, first it identifies operation and creates the process, then doing classification and prioritization before do the calculation. However, initially, the system doesn't have the instructions on fractional addition or any other fractional operation unless a developer has inserted. In that case, the incoming operator is classified as a 'New' operator. Therefore, the knowledge related to execution of that operation should be inserted into the system, before continue the particular operation. Then, the smaller tactics memory starts to be filled up with relevant label like records with an internal process. For example, such a record could be like this: (Operator, Presidency, Number of algorithms (there could be more than one algorithm to accomplish a certain operation), Name of the library file that contains instructions for the particular operation). E.g. (+,1,1,SumCalcModule). Other than the 'New' category, there are four other categories such as 'Same Expression', 'Same Operator', 'Different Operator', and 'Frequent Operator'. If the inserted expression is classified into the 'Same Expression' class, the SSPM-FC does not process the expression, but the previous response is displayed in response to the current input without taking extra time unnecessarily. Then, until the operation is not frequent, the system dynamically creates a file calling the particular method importing the particular library (collecting related things together for processing) file without filling the memory unnecessary time. If a particular operation frequently inserts into the system, the file, which was created dynamically according to the incoming operation will permanently store in the system adding that label into the smaller tactics memory. E.g. (+,1,1,SumCalcModule,SumCalcModuleMain).

Meantime, if there more than one algorithm, the most efficient algorithm will be selected by removing the inefficient algorithm and updating the record in the smaller tactics memory. E.g. (+,2,1,MulCalcModule,MulCalcModuleMain). All these preparation processes are internally carried out. Eventually, the smaller tactics memory fills up with four such records for four operators. The modifications applied on the smaller tactics memory, and the answers produced for the internal inputs can be considered as the internal outputs, while the answers produced for the external inputs can be considered as the external outputs. Then, it has been apparent that the memory has not been overloaded and has been a result of processing. Furthermore, it could empirically verify that the modifications have been applied both on the smaller tactics memory and the system through continuous processing. Further, due to these modifications system could efficiently perform its processing.

The SSPM-FC displayed the behavior of the smaller tactics memory better, where it was well showed how its smaller tactics memory was updated according to the actions (tactics) applied on the system and the inputs. Although, each of the four program cases considered in this research have specific operations, the application of the tactics or actions was similar to the internal process illustrated with figures in the section 5.2 of SSPM-FC.
Next section has illustrated a testing scenario of the Quadratic Equation Solver with a similar interface as had in SSPM-FC.

5.3 Quadratic Equation Solver (SSPM-QES)

This has also been similar to the above SSPM-FC. In this case, the calculations have been dependent on the nature of the discriminant such as positive (1), equal (0), and negative (-1). Then, the classifications have been made accordingly. In Figure. 5.9, it can be seen the SSPM-QES user interface.


Quadratic Equation Solver (Form of the Equaion : ax² + bx + c = 0, where a must be non-zero)
Enter the values for a, b, and c in the below spaces;
1
x²+
2
x+
-4
=0
Calculate
Cancel
Answer:
Smaller Memory
Internal OutPut
Current State
Change(Add/Remove) in the Algorithm/ Library File:

Created the Class for PositDiscrimModule

```
1.0,PositDiscrimModule,PositDiscrimModuleMain()
0.0,EqualDiscrimModule
-1.0,NegatDiscrimModule,NegatDiscrimModuleMain()
```

```
Quadratic Euqaion =1.0x²+2.0x-4.0=0
Root 1 =1.2360679774997898
Root 2 =-3.23606797749979
Discriminant Category =1.0
```

Figure.5.9. User interface for Quadratic Equation Solver when a class has been created for positive discriminant module (updated record in the smaller tactics memory), during an external process

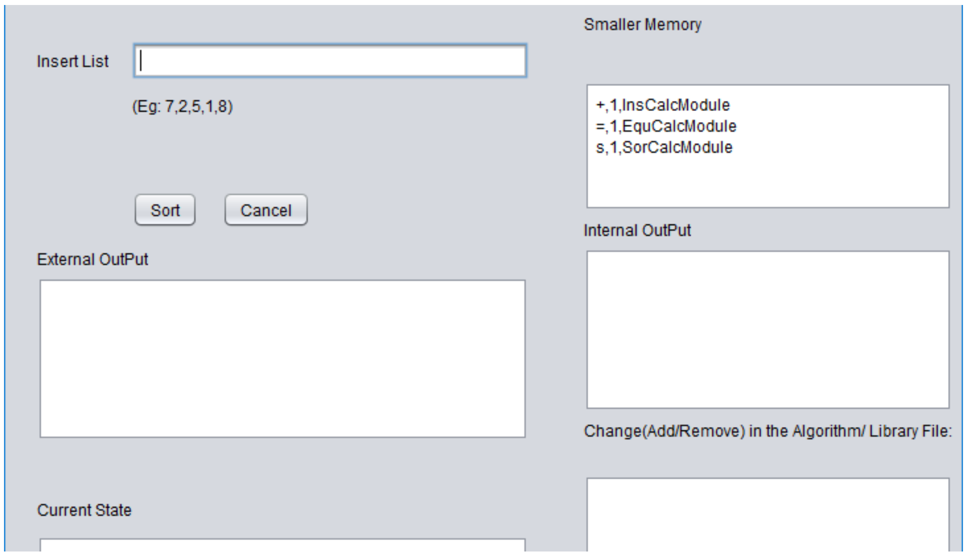
Sample Scenario:

Sample Input : $x^2 + 2x - 4 = 0$

Such a way, this has identified the equation and its discriminant, has created, and classified the process. However, initially, the system doesn't have the instructions regarding the particular discriminant. In that case, the incoming expression category is classified as a 'New'. Therefore, the knowledge related to execution of that operation should be inserted into the system, before continue the particular operation. Then, the smaller tactics memory starts to be filled up with relevant label like records with an internal process. For example, such a record could be like this: (Symbol for Discriminant, Name of the Module for the discriminant). E.g. (1,positDiscrimModule). Other than the ‘New’ category, there are four other categories such as ‘Same Expression’, ‘Same Discriminant’, ‘Different Discriminant’, and ‘Frequent Discriminant’. Here also, if the inserted equation is classified under the ‘Same Expression’ class, without processing the input further, the SSPM-QES displays the respective earlier response without unnecessarily taking extra time. Then, until the discriminant is not frequent, the system dynamically creates a file calling the particular method importing the particular library (collecting related things together for processing) file without filling the memory unnecessarily. If a particular operation frequently inserts into the system, the file, which was created dynamically according to the incoming operation will permanently store in the system adding that label into the smaller tactics memory. E.g. (1,positDiscrimModule, positDiscrimModuleMain).

5.4 Sorting Program (SSPM-Sorting)

This is also similar to the above two cases. In this customization regard the sorting on lists are reliant on the nature of the input pattern with compared to the most recent input and the respective computing technique to be selected: Insert, Equal, delete, and sort. Then, the classifications of the inputs and the relevant libraries or modules of instructions are made according to the computing technique. The Figure. 5.10 shows the SSPM-Sorting interface, before introduce the record ‘delete’ as a new technique into the smaller tactics memory.



```
+,1,InsCalcModule
=,1,EquCalcModule
s,1,SorCalcModule
```

Figure. 5.10. User interface for SSPM-Sorting, before introduce delete to the system.

In determining the particular computing technique, the identification of input was done also considering the *input pattern*s (*IP*s) mentioned in the section 4.3, where $IP1$: $X \subset Y$, $IP2$: $X \nsubseteq Y,\ Y \nsubseteq X\ \Rightarrow\ X \cap Y \neq \phi\ or\ X \cap Y = \phi$, $IP3$: $Y \subset X$, $IP4$: $X = Y \neq \phi$ (X is the previous list and the Y is the current list. When, analyzing the input in SSPM-Sorting input patterns were also used. This was something beyond the above two cases, where the SSPM-FC and SSPM-QES considered only the New Op, Different Op, Same Op, Same Exp, Frequent Op, and Unnecessary or inefficient module/method. These, *IPs* were also applicable in the SSPM-FC as well, and it was not worth to address at this level in SSPM-QES as it always solves quadratic equations with three or two components. However, when it comes to SSPM-Sorting it was a complex work to use the operators such a way, because the operators used within the program were > (less than) and < (greater than) to compare list elements, and Swap or Move to place the element is the appropriate position in the list. That was the major reason behind using the input patterns in SSPM-Sorting for the illustration of the nature of the SSPM in simple manner.

From the Figure. 4.2 (b), the input patterns for this SSPM-Sorting with examples are;

$IP1$: $X \subset Y$

Let $X = [1,2,3]$ and $Y = [2,3,1,5,6]$, then $X \cap Y = [1,2,3] \neq \phi$

For this particular pattern the applicable technique was ‘Insert’, where the additional elements in the current list (Y) can be inserted to appropriate places in the previously sorted list X.

$IP2$: $X \nsubseteq Y,\ Y \nsubseteq X \Rightarrow X \cap Y \neq \phi \ or \ X \cap Y = \phi$

1. Let $X = [1,2,3,4]$ and $Y = [2,3,1,5,6]$, then $X \cap Y = [1,2,3] \neq \phi$

Here, the both the techniques ‘Delete’ and ‘Insert’ had to be used. First, the additional element/s from the list X was/were deleted, and the additional element/s of the list Y was/were inserted into the sorted X.

Or

2. Let $X = [1,2,3]$ and $Y = [4,5,6]$, then $X \cap Y = [] = \phi$

In this scenario, no common elements were present. Therefore, the ‘Quicksort’ algorithm is applied directly.

$IP3$: $Y \subset X$

Let $X = [1,2,3,4,5,6]$ and $Y = [2,3,1]$, then $X \cap Y = [1,2,3] \neq \phi$

The ‘Delete’ was applied to remove additional element of the sorted list X.

$IP4$: $X = Y \neq \phi$

Let $X = [1,2,3]$ and $Y = [2,3,1]$, then $X \cap Y = X = Y = XUY = [1,2,3] \neq \phi$

Here, ‘Equal’ was used. No additional operation was applied other than displaying the previous sorted list X as the X and Y are equal.

Sample Scenario:

Sample Input, $X = [7,2,5,1,8]$

As in the scenarios of SSPM-FC and SSPM-QES, the knowledge regarding the insert, delete, equal, and sort should be stored in the system and should be organized with the smaller tactics memory before use a particular technique. Then, at first the system recognizes the input list and compare with the most recent input. However, initially, the system doesn’t have a previously sorted input. In that case, the elements of the incoming list are sorted with the quicksort using the record (s,1,sorCalcModule), which was stored in the smaller tactics memory. Here, the system works according to the four input patterns as in section 4.3, and frequent modules will be created accordingly.

For example, if the input is belonged to the ‘Same List’ category (*IP4*), it will not execute the input further, but it displays the previously sorted list without unnecessarily taking additional time.

Then, until a particular technique is not frequent, the system dynamically creates a file, calling the particular method importing the particular library (collecting related things together for processing) file without filling the memory unnecessarily. If a particular operation frequently inserts into the system, the file, which was created dynamically according to the incoming operation will permanently store in the system adding that label into the smaller tactics memory. E.g. (s,1,sorCalcModule, SorCalcModuleMain). This is similar to the SSPM-FC.

Here, it has been considered SSPM-S-Equal, and SSPM-S-Insertion specifically considering the techniques Equal (*IP4*), and Insert (*IP1, and IP2.1*) respectively.

However, there are a few other restrictions in selecting the particular computing technique, such as size of the list, and size of the cache line and the standard deviation of the elements in the list. These has been discussed in the evaluation chapter. Overall, by selecting the appropriate computing technique as per the rising conditions or requirements, while executing the program in consecutive program execution cycles, the performance of computing can be enhanced.

5.5 Simulated Processes Scheduler (SSPM-PS)

As introduced, this was the very first example customized by SSPM. However, it was a simple demonstration to show the ability of SSPM to improve processing time by identifying the nature of the inputs and appropriately selecting the serving algorithm.

Three processes such as P1, P2 and P3 were created. In the first round, those have been considered in the first come first serve (FCFS) basis in the incoming order P1, P2, and P3 assigning burst times manually. Then the turnaround times have been recorded. After executing the processes in the first round, system has learned the burst times, turnaround times and waiting times of each process. Then, the system did some preprocessing by comparing the burst time before enter into the next round. Then, the scheduling algorithm was changed to Shortest Job First (SJF) Algorithm rearranging the order of the processes in the ascending order of the respective burst times.

There have been three processes such as P1, P2 and P3 used in this system with the given burst time as shown in the Table 6.1. In the first round, those have been executed in the first come first serve (FCFS) basis in the order P1, P2, and P3. Then the turnaround times have been recorded. See the Table 6.1. Initially, the processes that have been queued for the execution do not reveal any fact on the burst time before the execution. Therefore, FCFS would have been the best approach. In this case, average waiting time is 4.6667 ms and average turnaround time is 8.6667 ms.

Table 6.1: Process execution with FCFS basis.

Process	Burst Time (ms)	Waiting Time (ms)	Turnaround Time (ms)
P1	5	0	5
P2	4	5	9
P3	3	9	12

But after executing the processes in the first round, system has learned the burst time of each process. Then, the system has done preprocessing by comparing the burst time before the next round and has changed the scheduling algorithm to Shortest Job First (SJF) Algorithm. Then, the result of the second round has been recorded in the Table 6.2. In this case, average waiting time is 3.3333 ms and average turnaround time is 7.3333 ms.

Table 6.2: Process execution with SJF basis.

Process	Burst Time (ms)	Waiting Time (ms)	Turnaround Time (ms)
P3	3	0	3
P2	4	3	7
P1	5	7	12

According to the results appeared in the Table 6.3, it has been apparent that the system has gained improvements.

Table 6.3: Results after Selection

Round	Average Waiting Time (ms)	Average Turnaround Time (ms)
1	4.6667	8.6667
2	3.3333	7.3333

Normally, in an OS, the scheduling techniques are applied in different levels such as long-term scheduling level, medium-term scheduling level, short-term scheduling level, and I/O scheduling level. As the short-term scheduling level does immediate process execution and directly involves in processing, it was considered for the demonstration of SSPM. In fact, the above description includes customizing a scheduling process that has the scheduling techniques FCFS and SJF with three processes. This attempt was further extended by involving some other scheduling techniques to schedule the processes together with the newly incoming processes in each processing execution cycles. There, the scheduling techniques such as Round Robin (RR), and Shortest Remaining Time (SRT) could be applied with the preemption technique. Hence, it could improve the waiting time and turnaround time by changing both the scheduling technique and the schedule of processes as in the above scenario. Further, the techniques such as exponential averaging can be used by referring the previously recorded burst times to predict future burst times and can accordingly allocate a suitable scheduling technique as well. In this scenario, the smaller tactics memory contains the entries to drive the scheduling process. This idea is yet to be developed, tested, and compared. Overall, the processing speed could be improved over subsequent OS processing cycles by customizing the scheduling process with the SSPM.

5.6 Summary

This chapter has presented how the proposed system works. First, it has started with explaining how the features of the models evident in the Fraction Calculator. Next, it has discussed a sample scenario with a given input. Latter, it has briefly mentioned how the other three systems: QES, Sorting, and PS work.

The next chapter has discussed the testing and the evaluation process conducted over the customized programs. Further, it describes the formal verification of the SSPM.

CHAPTER 06
TESTING AND EVALUATION

6.1 Introduction

The preceding chapter explained how the programs customized by the SSPM worked. This chapter reports the testing and evaluation processes, which were conducted over the proposed model. The testing has conducted for all the programs that have been customized by the proposed computing model or part of it. In addition to that, SSPM-FC, SSPM-QES, and SSPM-Sorting were evaluated empirically. Further, this sorting mechanism has been compared with some other exciting methods such as sorting with self-adjusting computing, incremental computing, dynamically tuned libraries, and genetic algorithmic approach. Moreover, the formal verification of the new computing model on the Turin Machine, has also been presented.

6.2 Experimental Mechanism of SSPM-FC.

This section focused on reporting the testing and evaluation process carried out for the prototype SSPM-FC, where the FC is customized by the new computing model with the continuous processing. Further, it tested the results of two key updates applied on the system, while executing the program over generations, through this testing process. Those two were reported separately under two testing setups.

The testing scenario one is allocated for the modification "Generate and store the operating modules for frequent operations". Then the second scenario is for "Eliminate inefficient methods and keep the most efficient method". By evaluating this, it was necessary to determine whether the system improved its processing power due to these modifications through generations of program executions. The following section onwards, it has been discussed the testing scenarios.

6.3 SSPM-FC - Testing Scenario 1

Under this scenario, the effect of "Generating and storing the operating modules for frequent operations through generations of program executions" was checked and evaluated. Further, this scenario has three sub-scenarios for addition, subtraction, and multiplication.

6.3.1 Experimental Setup

Throughout this testing scenario, the analysis was done separately for the operations addition (Plus Operator), subtraction (Minus Operator) and Multiplication (Multiplication Operator) as the process deviations can generated non-normal data [134]. For the same reason, the Division Operator has been omitted under this scenario as it uses Multiplication Operator in the algorithm.

6.3.2 Choice of Expressions and The Responses

For each of the sub scenarios sets of 100 expressions of fractions with relevant operators have been selected. For example, for the plus operator, a set of 100 fractional expressions with the addition has been used before and after the modification. Then, the same set of expressions was executed before and after organizing the smaller tactics memory. The time (nanoseconds) spent for the execution of each fractional expression was collected all over the consequent program execution cycles.

Mean Time Values (ns):

Addition

After = 22746925.12

Before = 43311187.9

Subtraction

After = 22383131.87

Before = 47069304.56

Multiplication

After $\quad$ = 21675694.22

Before = 45193156.85

6.3.3 Testing Scenario 1.1: Addition (Plus Operator):

Under this scenario the modification in the SSPM-FC was “Generate and store the operating module for the plus operator over e frequent operations on plus operator over generations of program executions.”.

Step 1:

As mentioned in the previous section, a set of 100 fractional expressions (using MedCalc, the required minimum sample size was determined as 4) with plus operator has been sequentially executed before and after the modification. The time taken during the each of expression execution has been recorded in nanoseconds. Hence, two sample sets of time values have been collected. As seen in the Figure 6.1, clear difference was evident in the collected paired time values.

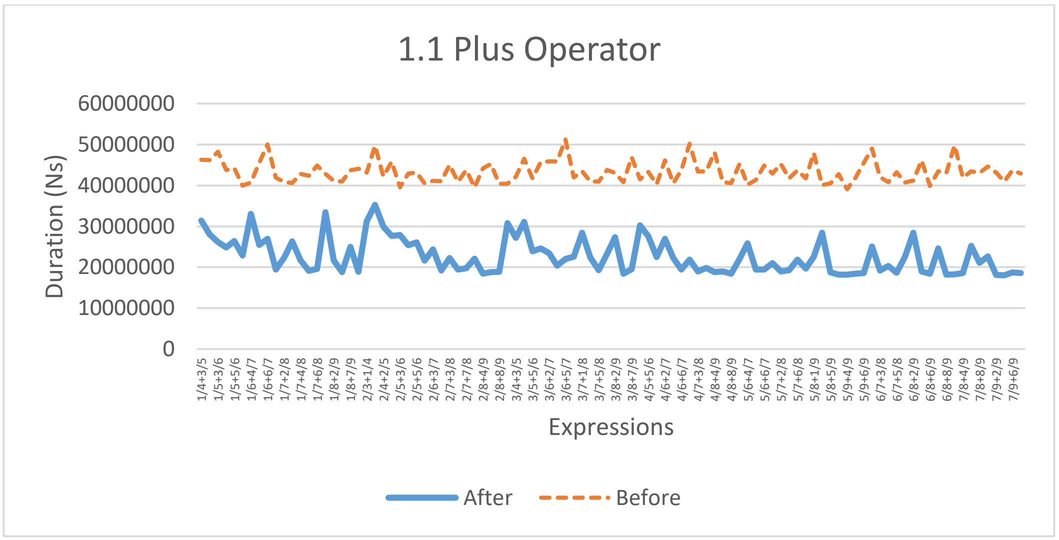


Figure. 6.1: Time values taken for computing fractional before and after organizing the smaller tactics memory of the FC with addition.

Size of the Sample =100

Mean Values:

Before the modification= 43311187.9 ns

After the modifications= 22746925.12 ns

Step 2:

Next, it was required to analyse the paired time values sets to identify the appropriate statistical test. For that, it was necessary to check whether the collected time values fulfil the assumptions or conditions stated in section 6.3.3.

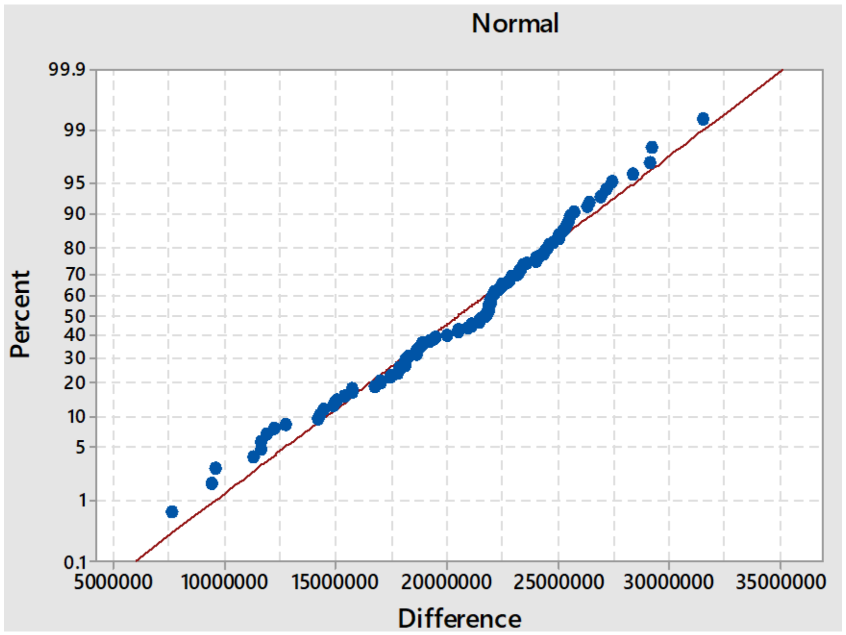
Condition 1: Are the time values in continuous scale?

As the time values were collected at time intervals in nanoseconds, these two sets of time values and their paired differences are from continuous scale.

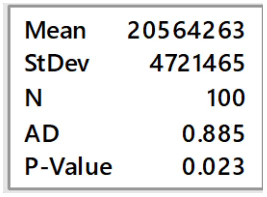
Condition 2: Does the set of differences of the paired time values have a Normal distribution?

Then, normality of the set of time values was determined using Anderson Darlin (AD) test using Minitab 17(with this test, it can be proved that there is no departure from normality) [134] by drawing the probability plot of difference that is showed in Figure. 6.2.

(a)



(b)



Mean	20564263
StDev	4721465
N	100
AD	0.885
P-Value	0.023

Figure. 6.2: Probability plot for addition (differences)

(a) Probability Plot (b)Values

Referring the graph of the Figure. 6.2, which has been drawn with Minitab 17, it has been proved the normality.

As it has given:

P-Value$_{calculated}$(0.023)> P-Value$_{tabulated}$(0.01) and

AD$_{calculated}$(0.885)< AD$_{calculated}$ (1.035).

Condition 3:

Do the differences of the paired time values have significant outliers?

In order to find outliers, the Grubb's test [136] has been applied as follows using Minitab 17.

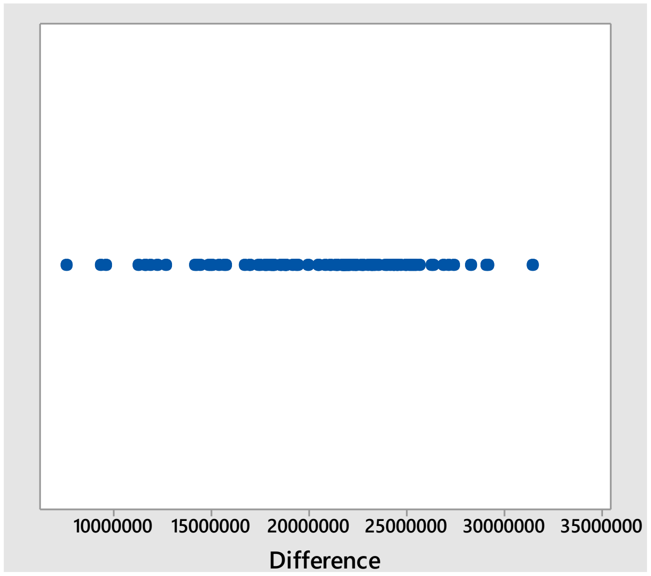
Outlier Test with Difference:

H_0 = All data values come from the same normal population

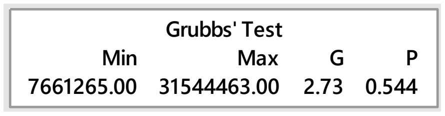
H_1 = Smallest or largest data value is an outlier

Significance level $\alpha = 0.01$ (Figure 6.3)

(a)



(b)



```
              Grubbs' Test
       Min              Max      G       P
7661265.00   31544463.00   2.73   0.544
```

Figure. 6.4: Outlier Plot for addition (a) Outlier Plot (b) Values for Differences

Grubbs' Test:

Table 6.4: Values from Grubbs' Test

Variable	N	Mean (ns)	StDev (ns)	Min (ns)	Max (ns)	G	P
Difference	100	20564263	4721465	7661265	31544463	2.73	0.544

Considering the values from Table 6.4;

Since, P-Value$_{calculated}$ = 0.544 is greater than the P-Value$_{tabulated}$ (= 0.01), #

Hence, the H_0 cannot be rejected and no outlier at the 1% level of significance.

Consequently, the differences of the paired time values have satisfied all the above mentioned conditions. Then, the power and the size of the paired-t test with related to these dependent samples has been checked as follows.

Step 3:

Further, power of the paired t-test with the sample size 100 has also been determined. Figure 6.4, has showed the relevant power curve.

Power and Sample Size for Paired t Test

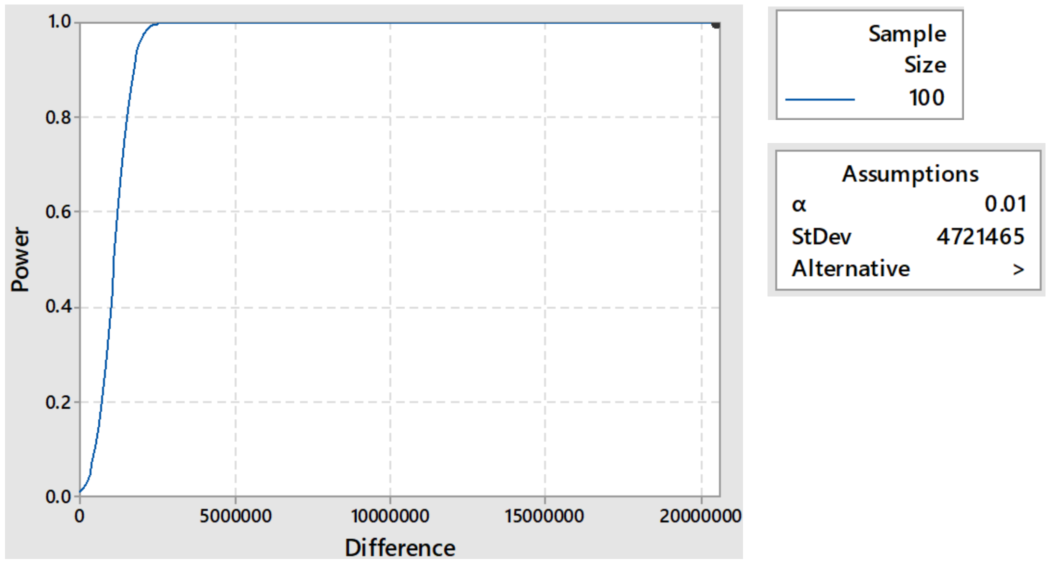


Figure. 6.5. Power Curve for Paired-t Test with size 100 (Addition)

Testing mean paired difference = 0 (versus > 0)

Power for mean paired difference at α = 0.01 and the assumed standard deviation of paired differences = 4721465

Table 6.5: Values from Power Test for Paired-T Test

Difference (ns)	Size	Power
20564263	100	1

For the sample size at 100 with 20564263 as the mean of paired differences as seen in Table 6.5, it has been resulted that the power of applying the paired-t test with 100 values has been high because the calculated Power value is 1.

This is the best justification for choosing the sample size 100. Further, it could prove that the most appropriate test to do this crossover analysis is paired-t test. Therefore, it was not necessary to go for the alternatives.

Step 4:

After checking the conditions, power, and the sample size, the paired-t test was applied for the two dependent set of times values with 99% confidence level.

Test Statistics;

$$T = \frac{\bar{D} - \mu_D}{S_d/\sqrt{n}}$$

$$T_{\text{calculated}} = t = \frac{\bar{d} - d_0}{s_d/\sqrt{n}}$$

$\bar{D}$=Random variable of mean of differences of values

μ_D=Mean of Differences of population

$\bar{d}$=Mean of Differences of values (Estimate of $\bar{D}$)

S_d=Random variable of standard deviation of differences of values

s_d=standard deviation of differences of values (Estimate of S_d)

n = Sample Size

d_0 = Population mean

Applying paired-t Test in Minitab 17 for the paired differences, the results are listed in the Table 6.6.

Table 6.6: Values for The Paired-T Test (Addition)

	N	Mean (ns)	StDev (ns)	SE Mean (ns)
Before Select	100	43311188	2706548	270655
After Select	100	22746925	4253045	425304
Difference	100	20564263	4721465	472147

From Table 6.6;

$s_d/\sqrt{n}$ = SE Mean = 472147 s_d = StDev =4721465 (ns)

$\bar{d}$=20564263 (ns) n = N =100

99% lower bound for mean difference: 19447822 (ns)

t-test of mean difference = 0 (vs > 0): T-Value = 43.55 P-Value = 0.000

Hence, $T_{calculated} = \bar{d}/(s_d/\sqrt{n})$

$= 43.55$

$T_{table}(99,0.01) = 2.365$ (degree of freedom = N-1 = 99)

$T_{calculated} > T_{table}$

Therefore, the H_0 can be rejected.

Hence, it was concluded that there is a significant improvement in the system after organizing the smaller tactics memory as the mean value of the times values collected before organizing the smaller tactics memory through creating and saving the computing modules for frequently executing addition operation, is greater than the mean value of the time values collected after organizing the smaller tactics memory.

6.3.4 Testing Scenario 1.2: Subtraction (Minus Operator)

The modification in the SSPM-FC considered under this scenario was "Generate and store the operating module for the minus operator over e frequent executions on minus operator over generations of program executions".

Step 1:

Here, a set of 100 expressions of fractions (using MedCalc, the required minimum sample size was determined as 4) with minus operator have been selected. Then, the rest of the procedure has been exactly similar to the procedure of scenario 1.1. The two sets of 100 time values per each expression with respect to the set of 100 expressions have been collected for the two cases; before and after the modification. A clear difference was evident in the collected paired time values as seen in the Figure. 6.5.

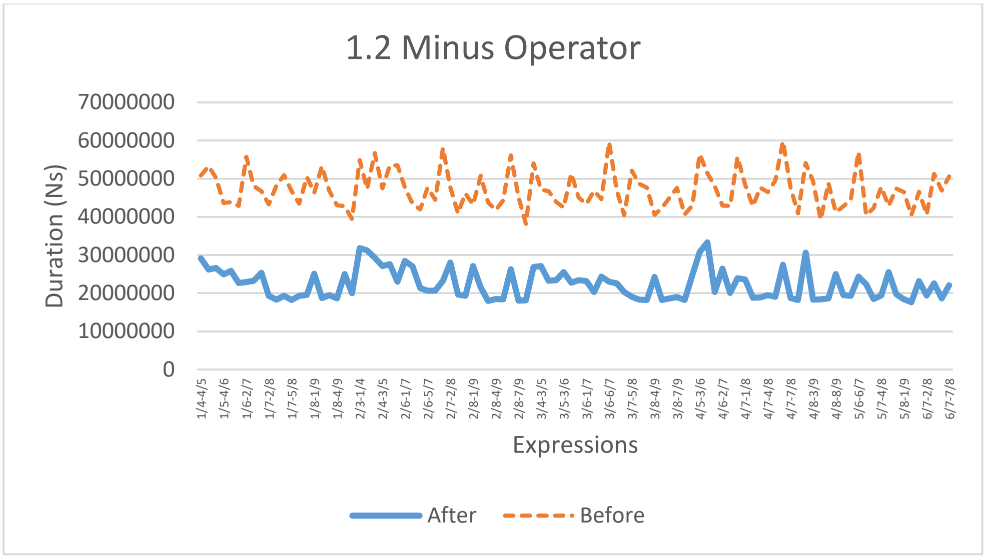


Figure. 6.5: Time values collected before and after organizing the smaller tactics memory of the FC with subtraction.

Sample size = 100

Mean Values:

Before the modification = 47069304.56 (ns)

After the modification = 22383131.87 (ns)

Step2:

As similar to the above scenario 1.1, it has been required to select the most suitable statistical test for this crossover study. Then, the paired samples were checked for the conformity to the conditions that have been mentioned in Testing Scenario 1.1.

Condition 1: Are the time values in continuous scale?

As the time values were collected at time intervals in nanoseconds, these two sets of time values and their paired differences are from continuous scale.

Condition 2: Does the set of differences of the paired time values have a Normal distribution?

Then, normality of the set of time values was determined using Anderson Darlin (AD) test using Minitab 17(with this test, it can be proved that there has not been a departure from normality) [134] by drawing the probability plot of difference that is showed in Figure. 6.6.

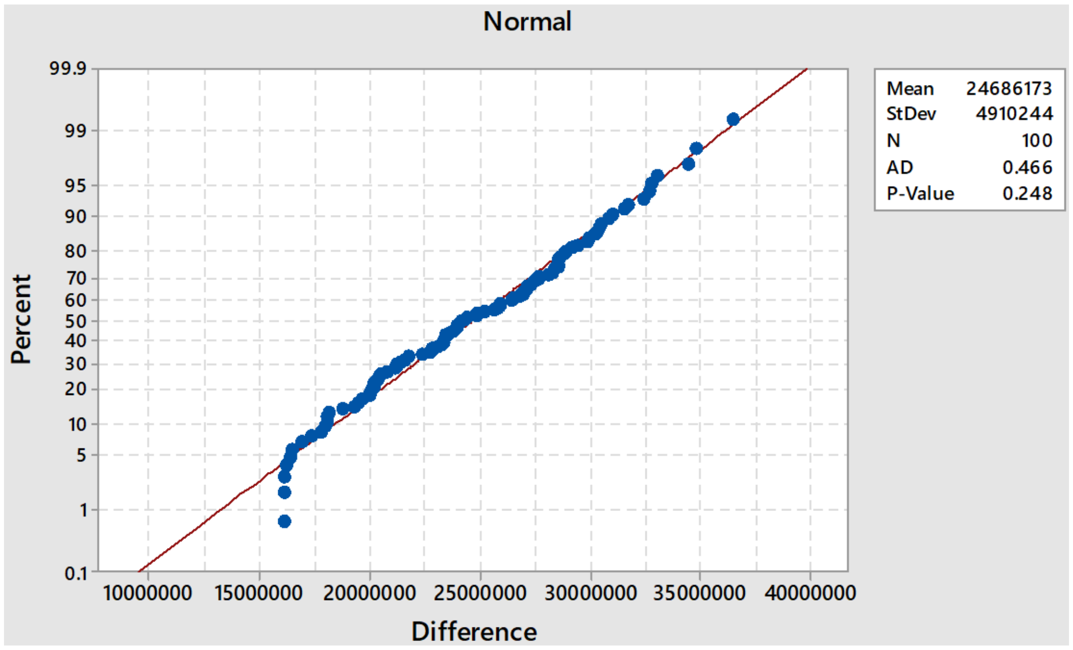


Figure. 6.6: Probability plot of difference for Subtraction

Hence, it has been proved the normality.

As it has given;

P-Value$_{calculated}$(0.248)> P-Value$_{tabulated}$(0.01)

and AD$_{calculated}$(0.466)< AD$_{calculated}$ (1.035) [135] with Minitab 17.

Condition 3:

Do the differences of the paired time values have significant outliers?

In order to find outliers, the Grubb's test [136] has been applied as follows using Minitab 17.

Outlier Test with Difference:

H_0 = All data values come from the same normal population

H_1 = Smallest or largest data value is an outlier

Significance level: α = 0.01 (Figure 6.7)

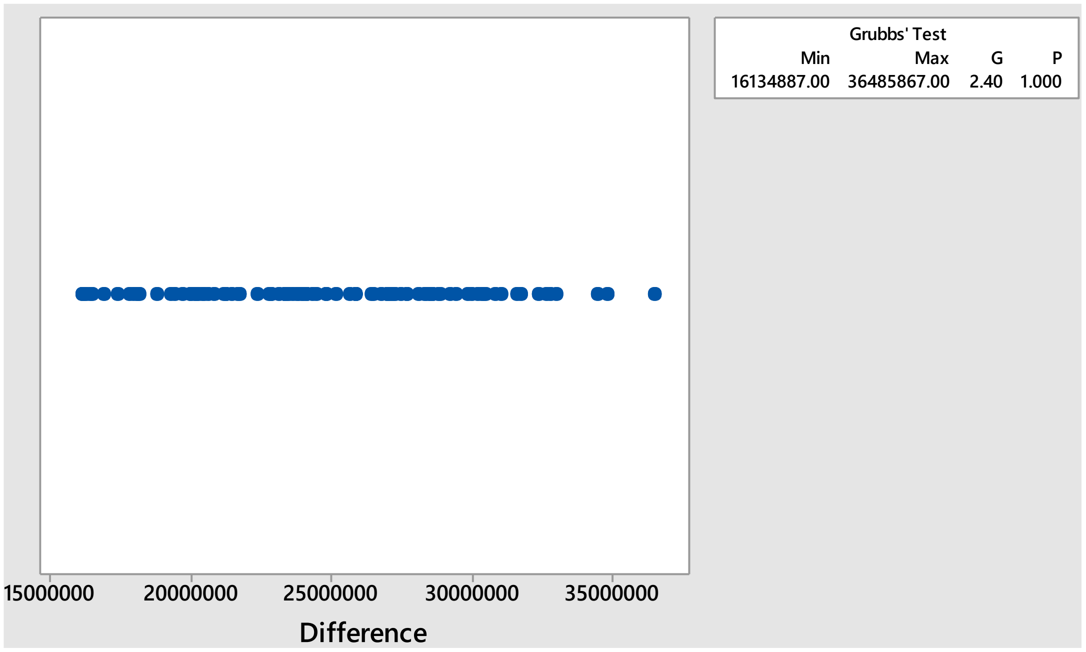


Figure. 6.7: Outlier Plot of Subtraction (a) Outlier Plot (b) Values for difference

Grubbs' Test:

Table 6.7: Values from Grubbs' Test

Variable	N	Mean (ns)	StDev (ns)	Min (ns)	Max (ns)	G	P
Difference	100	24686173	4910244	16134887	36485867	2.40	1.000

Considering the values from the Table 6.7;

Since, P-Value$_{calculated}$ =1.000 is greater than P-Value$_{tabulated}$ =0.01 (Significance Level) as in the Table 6.7.

Hence, the H_0 cannot be rejected and No outlier at the 1% level of significance.

Consequently, the paired differences satisfy all the conditions. Then, the power and the size of the paired-t test with related to these dependent samples was checked as follows.

Step 3:

Further, power of the paired t-test with the sample size 100 has also been determined. Figure 6.8, shows the relevant power curve.

Power and Sample Size for Paired t Test

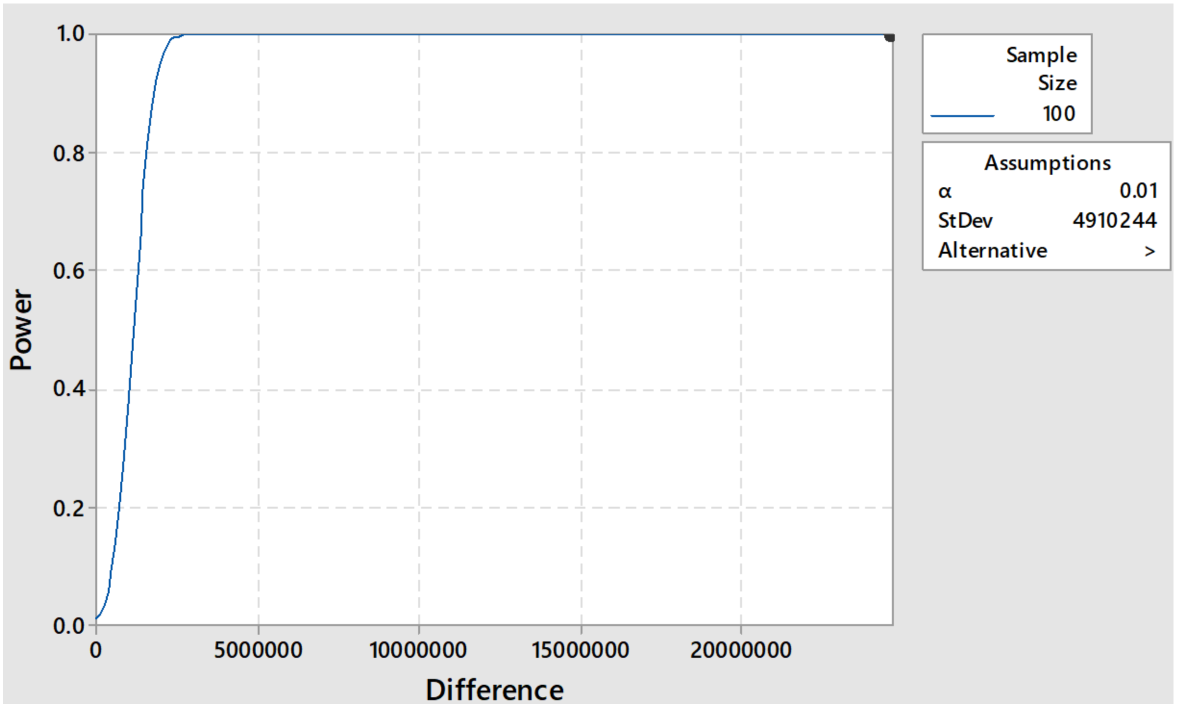


Figure. 6.8. Power Curve for Paired t Test with size 100 (Subtraction)

Testing mean paired difference = 0 (versus > 0)

Calculating power for mean paired difference at α = 0.01 and assumed standard deviation of paired differences = 4910244

Table 6.8: Values from Power Test for Paired-T Test with sample size 100

Difference (ns)	Size	Power
24686173	100	1

For the sample size at 100 with 24686173 as the mean of paired differences as seen in Table 6.8, it has been resulted that the power of applying the paired-t test with 100 values was high because the calculated Power value is 1.

This is the best justification for choosing the sample size 100. Further, it could prove that the most appropriate test to do this crossover analysis is paired-t test. Therefore, it was not necessary to go for the alternatives.

Step 4:

After checking the conditions, power, and the sample size, the paired-t test was applied for the two dependent samples with 99% significance level. Considering the difference between the time values collected before and after generate the frequent Subtraction module, the results were recorded in the Table 6.9.

Table 6.9: Values for The Paired-T Test (Subtraction)

	N	Mean (ns)	StDev (ns)	SE Mean (ns)
Before Select	100	47069305	4969214	496921
After Select	100	22383132	3873931	387393
Difference	100	24686173	4910244	491024

From Table 6.9;

$s_d/\sqrt{n}$ = SE Mean = 491024 s_d = StDev =4910244 (ns)

$\bar{d}$=24686173 (ns) n = N =100

99% lower bound for mean difference: 23525094 (ns)

T-Test of mean difference = 0 (vs > 0): T-Value = 50.27 P-Value = 0.000

Hence, $T_{calculated} = \bar{d}/(s_d/\sqrt{n})$

$$= 50.27$$

$$T_{table}(99,0.01) = 2.365 \text{ (degree of freedom = N-1 = 99)}$$

$$T_{calculated} > T_{table}$$

Further, the calculated P-value is less than the table P-value (significance level- 0.01)

Therefore, the H_0 can be rejected.

Hence, it was concluded that there is a significant improvement in the system after organizing the smaller tactics memory as the mean value of the times values collected before organizing the smaller tactics memory through creating and saving the computing modules for frequently executing subtract operation, is greater than the mean value of the time values collected after organizing the smaller tactics memory.

6.3.5 Testing Scenario 1.3: Multiplication (Multiplication Operator)

"Generate and store the operating module for the multiplication operator over the frequent operations on multiplication operator over generations of program executions" is the modification in SSPM-FC considered here.

Step 1:

Here, a set of 100 expressions of fractions (using MedCalc, the required minimum sample size was determined as 4) with multiplication operator have been selected. Then, the rest of the procedure is exactly similar to the procedure appeared in scenarios 1.1 and 1.2. The two sets of 100 time values per each expression with respect to the set of 100 expressions were collected for the two cases, before and after organizing the smaller tactics memory. A clear difference was evident in the collected paired time values as seen in the Figure. 6.9. However, it should be noted that this testing scenario has been conducted after finishing the below testing scenario 2 of SSPM-FC and selecting the best algorithm, i.e., this scenario has been conducted only with the method MulCalc1() after removing the inefficient algorithm MulCalc2().

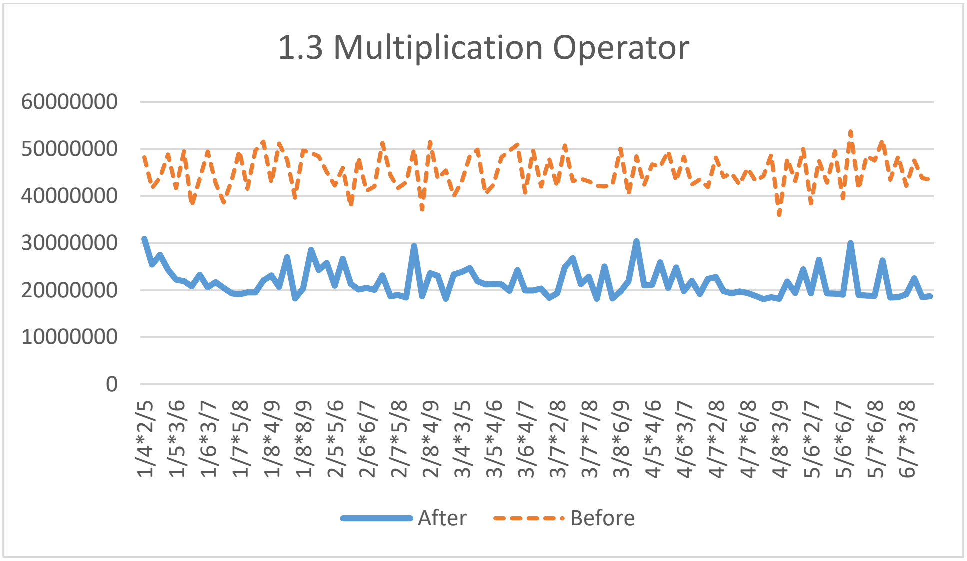


Figure. 6.9: Time values collected before and after organizing the smaller tactics memory of the FC with Multiplication. (it is possible to refer the complete samples in the appendix)

Sample size = 100

Mean Values:

Before the modification = 45193156.85 (ns)

After the modification = 21675694.22 (ns)

Step2:

Here also, first appropriate statistical test has been determined after checking the paired samples for the conditions as similar to the above scenarios 1.1 and 1.2. Since, the time data has been collected at time intervals, the samples and their differences have been in the continuous scale. Then, the differences of the paired samples have been checked for the normality using Anderson Darlin (AD) test and the probability plot of differences has been drawn using Minitab 17 as seen in Figure. 6.10.

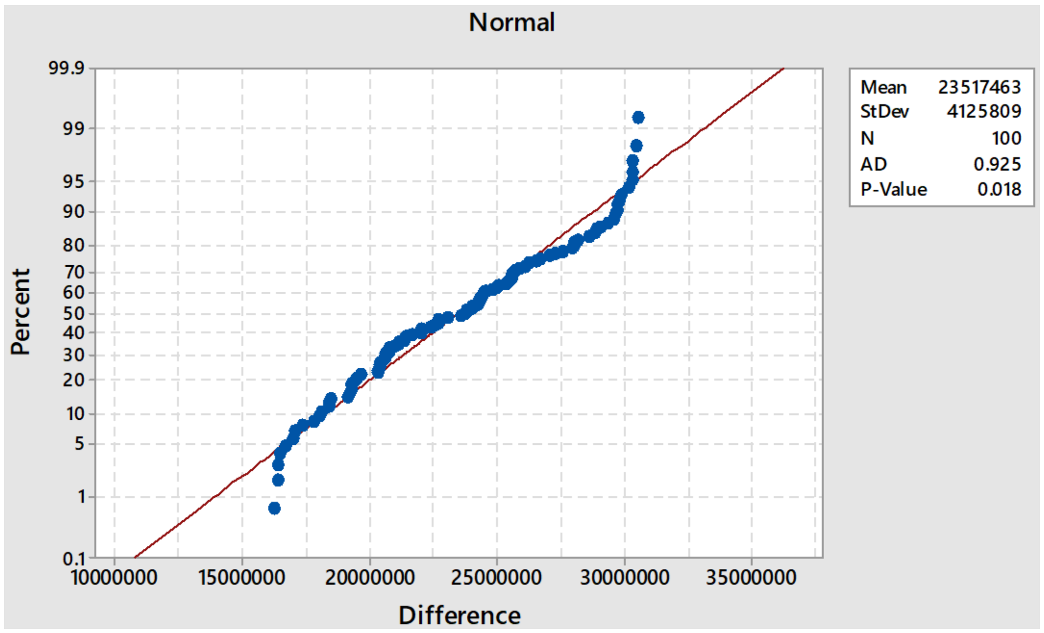


Figure. 6.10: The probability plot of differences (Multiplication)

Hence, it has been proved the normality,

As it has given,

P-Value$_{calculated}$(0.018)> P-Value$_{tabulated}$(0.01) and

AD$_{calculated}$(0.925)< AD$_{tabulated}$ (1.035) with Minitab 17.

Condition 3:

Do the differences of the paired time values have significant outliers?

In order to find outliers, the Grubb's test [136] has been applied as follows using Minitab 17.

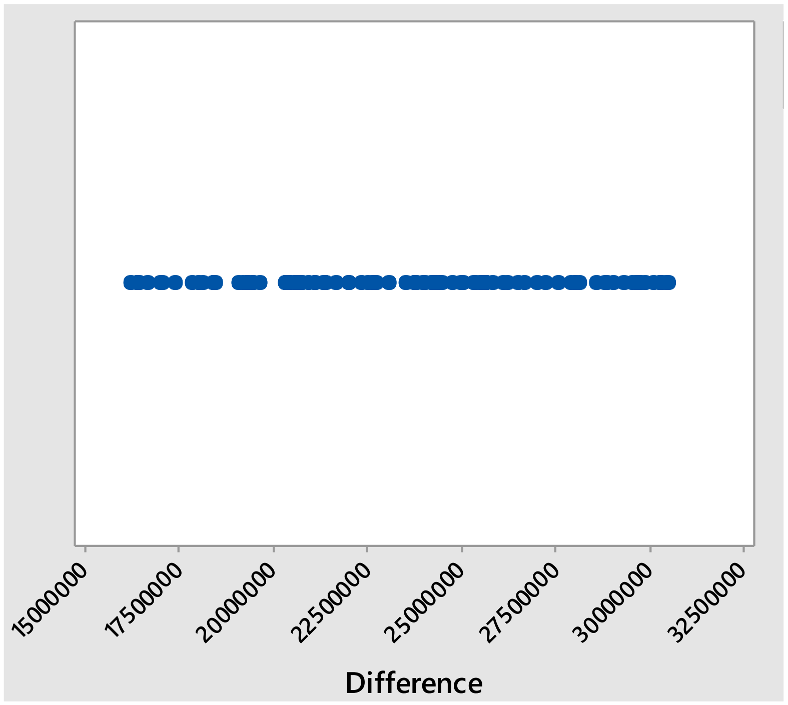
Outlier Test for Difference:

H_0 = All data values come from the same normal population

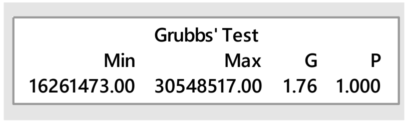
H_1 = Smallest or largest data value is an outlier

Significance level $\alpha = 0.01$ (Figure 6.11)

(a)



(b)



Grubbs' Test			
Min	Max	G	P
16261473.00	30548517.00	1.76	1.000

Figure. 6.11: Outlier Plot of Difference (a) Outlier Plot (b) Values (Multiplication)

Grubbs' Test:

Table 6.10: Values From

Grubbs’ Test

Variable	N	Mean (ns)	StDev (ns)	Min (ns)	Max (ns)	G (ns)	P (ns)
Difference	100	23517463	4125809	16261473	30548517	1.76	1.000

Considering the values from Table 6.10;

Since, P-Value$_{\text{calculated}}$ =1.000 is greater than P-Value$_{\text{tabulated}}$ =0.01 (Significance Level), the null hypothesis cannot be rejected and No outlier at the 1% level of significance.

Thus, the differences of the paired samples have satisfied all the conditions. With all these, as the next step, the power and the size of the paired-t test with related to these dependent samples has been checked as follows.

Step 3:

Further, power of the paired t-test with the sample size 100 has also been determined. Figure 6.12, shown the relevant power curve.

Power and Sample Size for Paired t Test

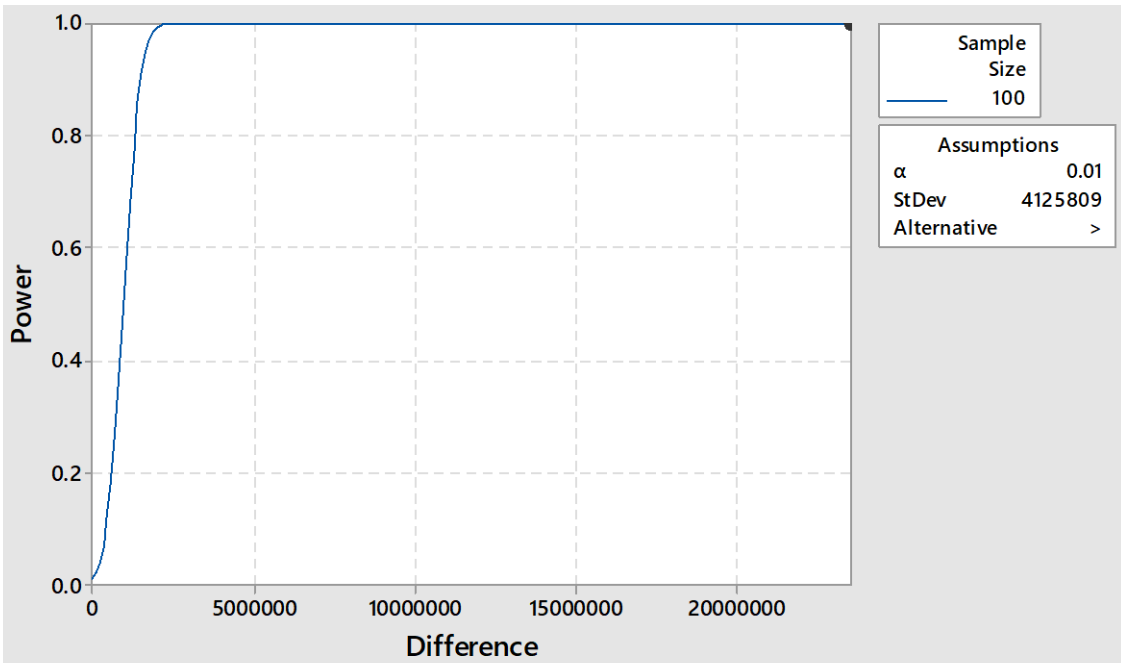


Figure. 6.12. Power Curve for Paired t Test with size 100 (Multiplication)

Testing mean paired difference = 0 (versus > 0)

Calculating power for mean paired difference at α = 0.01 and assumed standard deviation of paired differences = 4125809 (ns)

Table 6.11: Values from Power Test for Paired-T Test with sample size 100

Difference (ns)	Size	Power
23517463	100	1

For the sample size 100 with 23517463 ns as the mean of paired differences, it has been resulted that the power of applying the paired-t test with 100 values has been high because the power value is 1 as showed in Table 6.11. Also this result of the power test has provided a justification for selecting this sample size. Hence, it was determined that the most relevant test to do this analysis was paired-t test and it was not necessary go for the alternatives.

Step 4:

After checking the conditions, power with the given sample size, finally the paired-t test was applied for the two dependent samples with 99% significance level. Considering the difference between the time values collected before and after generate the frequent Multiplication module, the results were recorded in the Table 6.12.

Table 6.12: Values for The Paired-T Test (Multiplication)

	N	Mean (ns)	StDev (ns)	SE Mean (ns)
Before Select	100	45193157	3970568	397057
After Select	100	21675694	3117660	311766
Difference	100	23517463	4125809	412581

From Table 6.12;

$s_d/\sqrt{n}$ = SE Mean = 412581 (ns) s_d = StDev =4125809

$\bar{d}$=23517463 (ns) n = N =100

99% lower bound for mean difference: 22541871 (ns)

T-Test of mean difference = 0 (vs > 0): T-Value = 57.00, P-Value = 0.000

Hence, $T_{calculated} = \bar{d}/(s_d/\sqrt{n}) = 57.00$

$T_{tabulated}(99,0.01) = 2.365$ (degree of freedom = N-1 = 99)

$T_{calculated} > T_{tabulated}$

Therefore, the H_0 can be rejected.
Hence, it was concluded that there is a significant improvement in the system after organizing the smaller tactics memory as the mean value of the times values collected before organizing the smaller tactics memory through creating and saving the computing modules for frequently executing multiplication operation, is greater than the mean value of the time values collected after organizing the smaller tactics memory.

Longer or shorter fractional expressions with mixed operations, or proper or improper or fractions with whole numbers, or bracketed sub fractional expressions are all allowed in our model. In fact, the results retrieved from the executions of fractional expressions with mixed operations are non-normal. Then, for these non-normal data sets, it is possible to apply Wilcoxon Signed Rank test or Sign Test, which are the alternatives for paired-t test, but with lower power. Please note that, since our plan was to apply the most powerful statistical test (paired-t test) to these crossover studies and the paired-t test does not support non-normal data, the different operations were not mixed up in fractional expressions.

6.4 SSPM-FC - Testing Scenario 2:

This scenario was used to check and evaluate the effect of "Elimination of inefficient methods, while keeping the most efficient algorithm in multiplication over program execution cycles". Stated in another way, it was expected to check whether the system had gained an improvement through the modification.

6.4.1 Experimental Setup

As mentioned in the prior subsection 6.3.1, the main hypothesis of this research has been reduced to the following hypothesis.

The hypothesis, which has been tested in this scenario is;

H_0: No difference exists between the means time values.

(No processing speed improvement over consecutive program execution cycles)

(H_0: $\mu_D = d_0$, $d_0=0$)

H_1: The mean value of the time values collected when using the inefficient algorithm is greater than the mean value of the time values collected with the selection process and thereafter.

(processing speed has improved over consecutive program execution cycles)

(H_1: $\mu_D > d_0$, $d_0=0$)

Since, the method MulCalc2() took a longer time to perform multiplication than the method MulCalc1() from the two methods MulCalc1() and MulCalc2(), MulCalc1() was decided as the most efficient algorithm among those two.

If a programmer unknowingly uses inefficient MulCalc2() for multiplication all over the execution cycles, it would be certainly time consuming and requires more processing. Therefore, in early program execution cycles, allocating some time for pre-processing to select best algorithm and continuing with the selected best algorithm, improves quality and speed of program execution than using the inefficient algorithm throughout the cycles.

6.4.2 Choice of Inputs and The Responses

The time values were collected by executing same set of expressions of fractions with multiplication operator during both the selection process and when using the inefficient algorithm. For an arbitrary set of twenty such expressions, time values that were retrieved are shown in the Table 6.13. Next, for these two cases, total time values were computed as shown in the last raw of Table 6.13. (Selection Process = 2323140714 ns, Inefficient Algorithms (MulCalc2()) = 2786388917 ns).

Table 6.13: Selection Process Vs Inefficient Algorithm

Expression	Selection		MulCalc2()
	Duration (ns)	Method	Duration (ns)
1/2*2/3	116316738	M1	272323879
1/2*2/3	128066758	M2	
1/3*1/4	46576950	M1	153578831
1/3*1/4	117959285	M2	
1/3*2/4	55910434	M1	146680499
1/3*2/4	144987812	M2	
1/3*3/4	41370224	M1	135732195
1/3*3/4	122324023	M2	
1/4*1/5	47899063	M1	131248869
1/4*1/5	129464903	M2	
1/4*2/5	55854554	M1	145145888
1/4*2/5	123275507	M2	
1/4*3/5	42021777	M1	122066299
1/4*3/5	115556058	M2	
1/4*4/5	54413837	M1	131729741
1/4*4/5	125718270	M2	
1/5*3/6	51362864	M1	132207953
1/5*3/6	150794411	M2	
1/5*4/6	53335011	M1	127790389
1/5*4/6	166784509	M2	
1/5*5/6	41904696	M1	160376426

2/3*1/4	55239494	M1	124228894
2/3*2/4	42032801	M1	131434375
2/3*3/4	54646100	M1	136603847
2/4*1/5	52533684	M1	152881281
2/4*2/5	46813394	M1	206332993
2/4*3/5	41045968	M1	111717091
2/4*4/5	55861016	M1	136422523
3/4*1/5	43070573	M1	127886944
3/4*2/5	55433363	M1	143891059
Total Time	2323140714		2786388917

Similarly, 20 total values were computed from executing 20 sets of twenty different expressions as seen in the Table 6.14.

Table 6.14. Total Values for Selection Process and Inefficient Algorithm (M2)

Sample	Selection (ns)	MulCalc2() (ns)
1	2323140714	2786388917
2	2134226128	2471722339
3	2438671461	2633183668
4	2737122465	2983123800
5	2160134007	2526952881
6	2534930827	2860031523
7	2140585380	2503459176
8	2223174431	2591368949
9	3217582268	3237629193
10	2141014233	2457161559
11	3376966266	3624723832
12	2251626854	2521410904

13	2177035960	2458623236
14	2505120834	2633126388
15	2768560208	3171579564
16	2521708504	2714581461
17	2202959895	2812596076
18	2355105321	2739438643
19	2785844294	3185859802
20	2138863042	2473804734

6.4.3 Relevant Statistical Tests

This analysis can also be considered as a crossover study, since the same set of expressions of fractions has been used as in testing scenario 1. Therefore, it is not necessary to repeat the description of selection of relevant statistical tests here.

6.4.4 The Scenario

Step 1:

As mentioned earlier, total time values taken during the execution of twenty sets of expressions (using MedCalc, the required minimum sample size was determined as 6) with the efficient algorithm after the selection process and with the inefficient algorithm, has been used in this scenario. Those values have been illustrated in Figure 6.13. There, a clear difference has been observed.

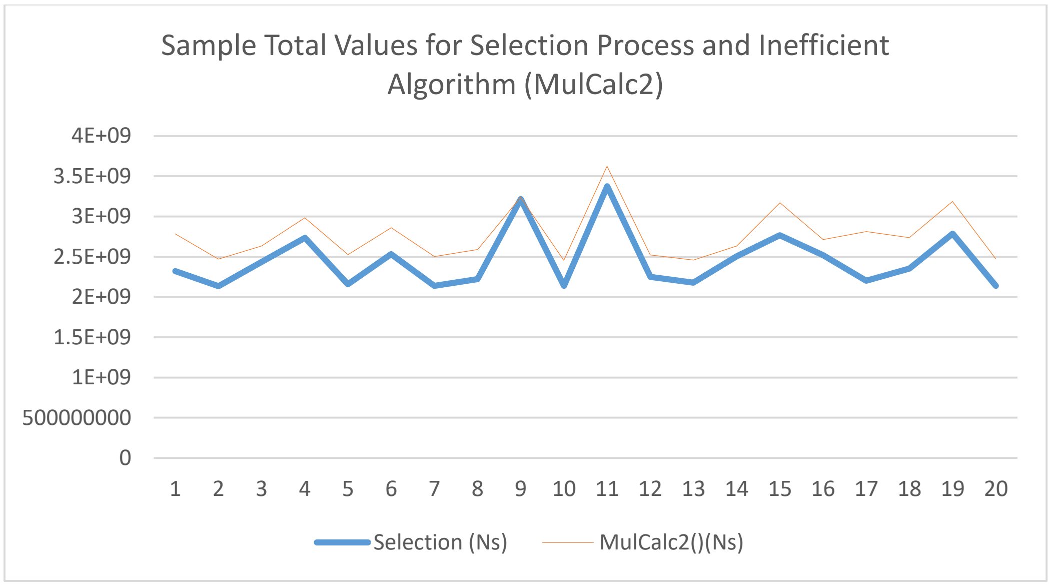


Figure. 6.13: Total time values in nanoseconds for best multiplication algorithms with selection process vs inefficient multiplication algorithm (MulCalc2())

Descriptive statistics:

Variable	N	Mean (ns)	StDev (ns)
Difference	20	316657478	127920870

Mean Values

With inefficient Method M2= 2769338332 (ns)

With the selection and the best algorithms= 2456718655 (ns)

Step2:

As similar to the above testing scenario 1.1, 1.2, and 1.3, the paired samples have been checked for the conditions. All the used time durations have been belonged to the continuous scale.

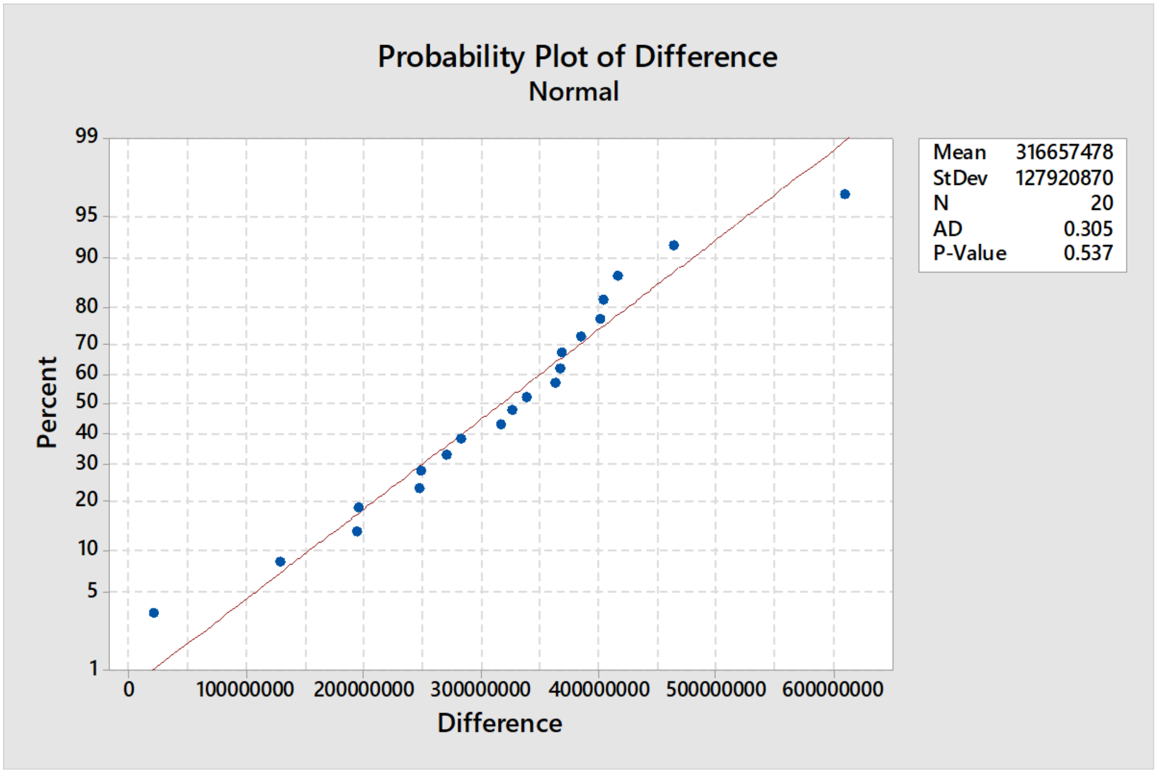


Figure 6.14: The probability plot of differences (Total Time Values)

Then, the differences of the paired samples have been checked for the normality using Anderson Darlin (AD) test and the probability plot of differences has been drawn by Minitab 18 as seen in the Figure. 6.14.

Hence, it has been proved the normality,

As it has given;

P-Value$_{calculated}$(0.537)> P-Value$_{tabulated}$(0.01) and

AD$_{calculated}$(0.305)< AD$_{tabulated}$ (1.035) [135] with Minitab 18.

The last assumption to check was whether the differences of the paired samples have significant outliers. For this, the Grubbs' test [136] was applied as follows using Minitab 18.

Outlier Test for Difference:

H_0 = All data values come from the same normal population

H_1 = Smallest or largest data value is an outlier

Significance level $\alpha = 0.05$ (Figure 6.15)

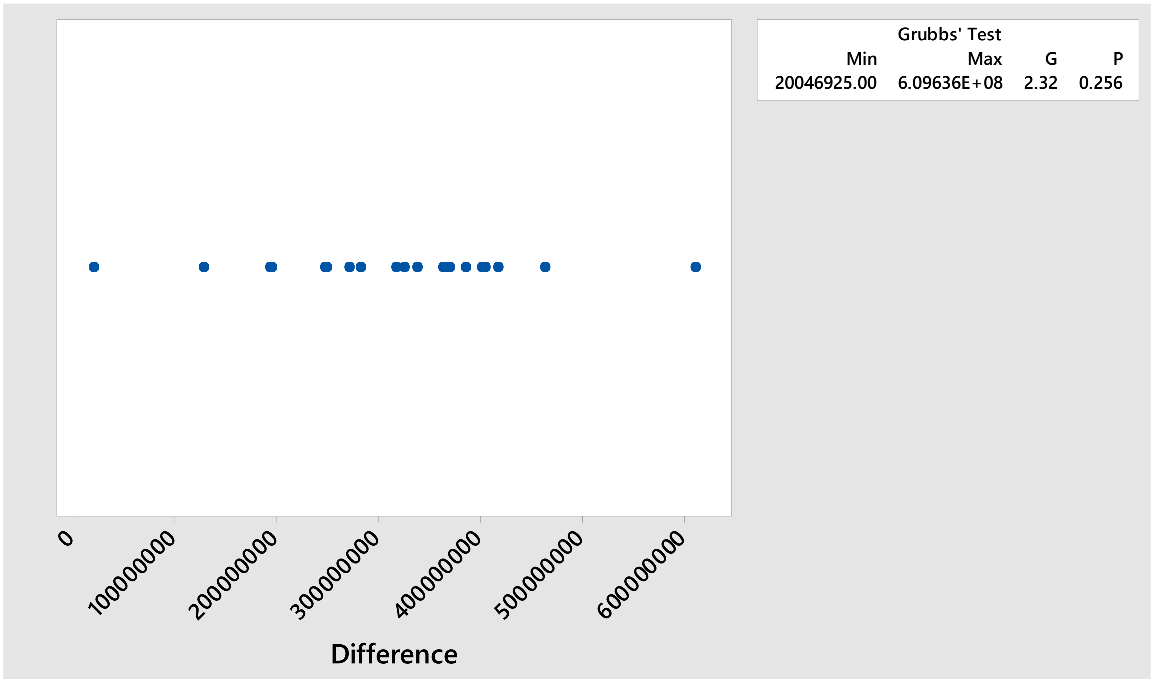


Figure. 6.15: Outlier Plot of Difference (a) Outlier Plot (b) Values (Total Time values)

Grubbs' Test:

Table 6.15: Values from Grubbs' Test

Variable	N	Mean (ns)	StDev (ns)	Min (ns)	Max (ns)	G	P
Difference	20	316657478	127920870	20046925	609636181	2.32	0.256

Considering the values from Table 6.15;

Since, P-Value$_{calculated}$ =0.256 is greater than P-Value$_{tabulated}$ =0.01,

H_0 cannot be rejected.

Hence, no outlier at the 1% level of significance.

Consequently, the mean paired differences have satisfied all conditions. With all of these, as the next step, the power and the size of the paired-t test with related to these dependent samples has been checked as follows.

Step 3:

Further, power of the paired t-test with the sample size 20 has also been determined. Figure 6.16, has showed the relevant power curve.

Power and Sample Size for Paired t Test

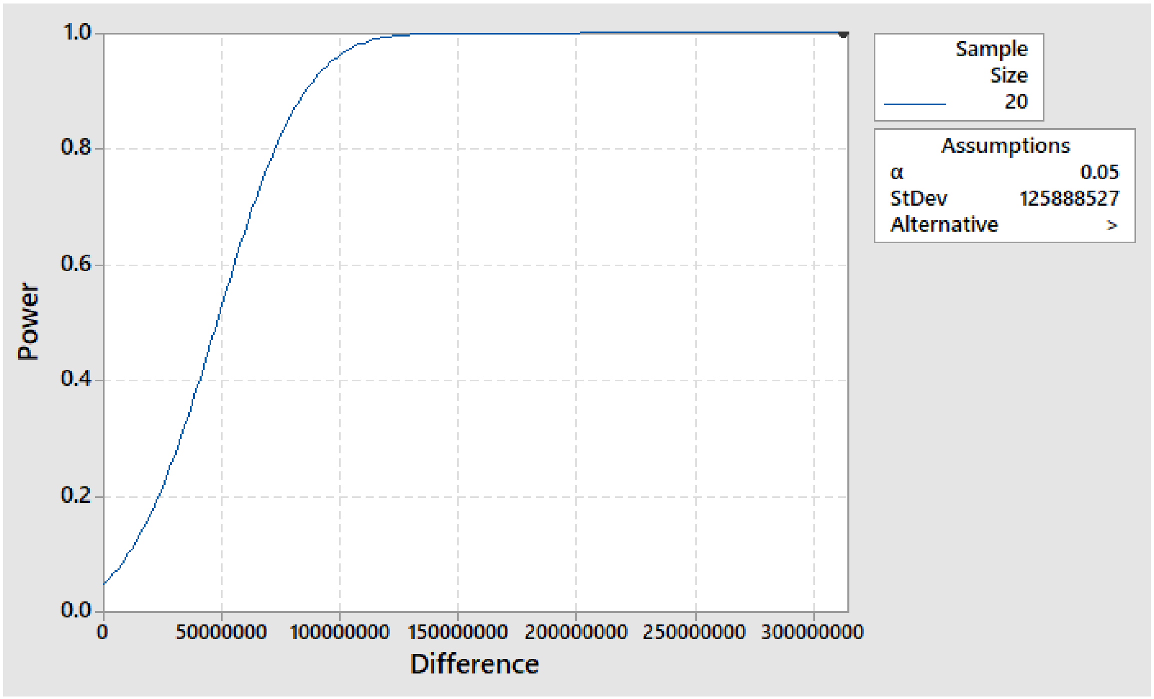


Figure. 6.16: Power Curve for Paired t Test with size 20 (Total Time Values)

Testing mean paired difference = 0 (versus > 0)

Calculating power for mean paired difference

α = 0.05 and assumed standard deviation of paired differences = 125888527

Table 6.16: Values from Power Test for Paired-T Test with sample size 20

Difference (ns)	Size	Power
312619678	20	1

For the sample size 20 with 312619678 as the mean of paired differences, it has been resulted that the power of applying the paired-t test with 20 values has been high because the power value is 1 as in Table 6.16. Further, it could prove that the most appropriate test to do this crossover analysis with 20 values, is paired-t test. Therefore, it was not necessary to go for the alternatives.

Step 4:

After checking the conditions, power with the given sample size, finally the paired-t test was applied for the two dependent samples with 99% significance level. In this scenario, the difference between the times values before and after removing the inefficient algorithm and selecting the best algorithm were considered. The results were reported in the Table 6.17.

Table 6.17: Values for The Paired-T Test (Total Time Values)

	N	Mean (ns)	StDev (ns)	SE Mean (ns)
M2	20	2769338332	322477375	72108133
With Selection	20	2456718655	361500451	80833958
Difference	20	312619678	125888527	28149530

From Table 6.17;

$s_d/\sqrt{n}$ = SE Mean = 28149530 (ns) s_d = StDev =125888527 (ns)

$\bar{d}$=312619678 (ns) n = N =20

99% lower bound for mean difference: 312619678 (ns)

T-Test of mean difference = 0 (vs > 0): T-Value = 11.11 P-Value = 0.000

Hence, $T_{calculated} = \bar{d}/(s_d/\sqrt{n}) = 11.11$

$T_{table}(19,0.01) = 2.539$ (degree of freedom = N-1 = 19)

$T_{calculated} > T_{table}$

Therefore, the H_0 can be rejected.

Hence, it was concluded that there is a significant improvement in the system after organizing the smaller tactics memory as the mean value of the times values collected before organizing the smaller tactics memory through removing inefficient algorithm and keeping most efficient algorithm, is greater than the mean value of the time values collected after organizing the smaller tactics memory.

6.5 Summarizing Results of the Experiments on SSPM-FC

Since, it has been just passed a long line of experiments and results on SSPM-FC, it has been required to summarize the results in a table to convince the significance of the proposed model. Here, the Table 6.18 has briefed the results.

Table 6.18: Summary of Results (SSPM-FC)

Scenario	N	$T_{calculated}$	T_{table}	Null Hypothesis	Improvement has gained (Yes/No)
Testing Scenario 1.1	100	43.55	2.365	Rejected	Yes
Testing Scenario 1.2	100	50.27	2.365	Rejected	Yes
Testing Scenario 1.3	100	57.00	2.365	Rejected	Yes
Testing Scenario 2	20	11.11	2.539	Rejected	Yes

6.6 SSPM-QES Testing Scenario

This testing scenario was conducted for creating and saving modules for positive discriminant as it has been frequently occurring. Moreover, this testing scenario has been similar to the testing scenario 1 of SSPM-FC in [16]. Then, the particular hypothesis was same as the hypothesis mentioned in the 4.10

Method:

A set of 59 quadratic equations (using MedCalc, the required minimum sample size was determined as 5) with positive discriminant has been executed before and after the modification and execution durations were recorded in nanoseconds in consecutive calculation cycles. It could observe a clear difference between the paired samples (two time samples) as shown in the below Figure. 6.17. Then, it was necessary to choose the best statistical test for analysis. Therefore, these paired samples were checked to determine, whether those are in continuous scale and normally distributed, and have no outliers.

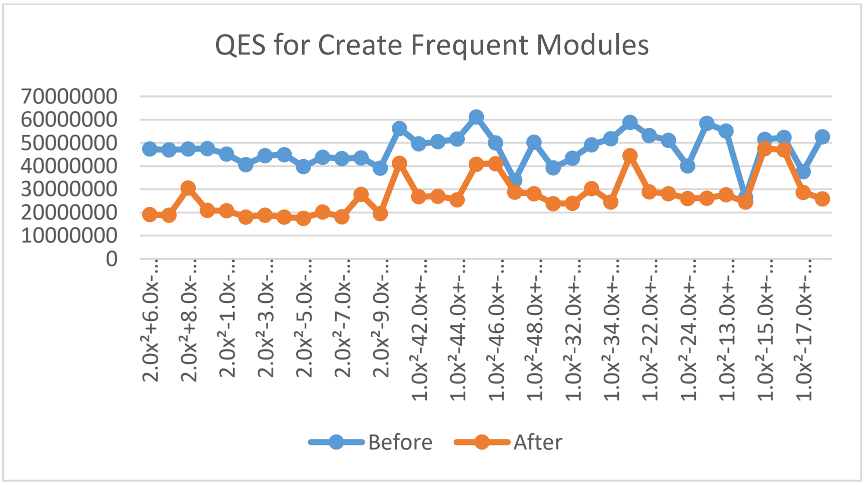


Figure. 6.17: Graph of Time values recorded with the quadratic equations with positive discriminant before and after organizing the smaller tactics memory of the SSPM-QES.

Descriptive Statistics:

Mean values:

Before the modification= 44738692 (ns)

After the modification= 25210417 (ns)

These paired populations are in continuous scale as those have been collected at time intervals in nanoseconds.

After applying Anderson-Darling test for the differences of the paired samples at significance level of 0.05, it has been obtained the P-value as 0.1897.

For the Anderson Darling Test:

Null hypothesis	H_0: Differences of the paired samples follow a normal distribution
Alternative hypothesis	H_1: Differences of the paired samples do not follow a normal distribution

Here, the P-value is greater than the significance level, the null hypothesis cannot be rejected. Therefore, Differences of the paired samples follow a normal distribution.

Next, it was necessary to check for outliers in the paired samples. The Grubb's test has been used for this purpose with the significance level 0.05.

For the Grubb's Test:

Null hypothesis	: All data values come from the same normal population
Alternative hypothesis	: Smallest or largest data value is an outlier

Here also, the critical value obtained is 3.19 and it is greater than significance level 0.05. Then, it has been decided that the null hypothesis cannot be rejected. Therefore, the differences of paired samples have no outliers.

Since, all the three conditions are satisfied by the differences of the paired samples, the paired-t test could be applied for the analysis.

Finally, the paired-t test has been applied for the two dependent samples with 99% significance level.

Test Statistics;

$$T = \frac{\bar{D} - \mu_D}{S_d/\sqrt{n}}$$

$$\mathrm{T}_{\mathrm{calculated}} = \mathrm{t} = \frac{\bar{d} - d_0}{s_d/\sqrt{n}}$$

$\bar{D}$=Random variable of mean of differences of values

μ_D=Mean of Differences of population

$\bar{d}$=Mean of Differences of values (Estimate of $\bar{D}$)

S_d=Random variable of standard deviation of differences of values

s_d=standard deviation of differences of values (Estimate of S_d)

n = Sample Size

d_0 = Population mean

Applying paired-t Test for the difference between Before Create and After Create the particular module:

Table 6.19: Values for The Paired-T Test QES

	N	Mean (ns)	StDev (ns)	SE Mean (ns)
Before create	59	44738692	7356546	957740
After create	59	25210417	7228602	941083

Difference	59	19528275	7233449	941714

From Table 6.19;

$s_d/\sqrt{n}$ = SE Mean = 941714

s_d = StDev =7233449 (ns)

$\bar{d}$=19528275 (ns)

n = N =59

95% lower bound for mean difference: 17643228 (ns)

t-test of mean difference = 0 (vs > 0): T-Value = 20.7369

Hence, $T_{calculated}$ $= \bar{d}/(s_d/\sqrt{n})$

= 20.7369

$T_{table}(58,0.05)$ = 1.658

(degree of freedom = N-1 = 58)

$T_{calculated} > T_{table}$

Therefore, the null hypothesis can be rejected and it is possible to conclude that there is a significant improvement in the system after the modification as the mean value of the sample collected before the modifications (Create and Save the computing modules for frequently executing quadratic equations with positive discriminant) is greater than the mean value of the sample collected after the modification.

Hence, it was proved that the system would become more efficient in succeeding execution cycles due to the conditionally evolving memory. Having this empirical proof of the applicability of the new model [17], as the next step, it was conducted a formal verification by simulating the model in a Turing Machine.

6.7 Experimental Mechanism on the Implementation of Sorting Program

This section has reported on the testing and evaluation process conducted over the Quicksort algorithm customized by the six-state continuous processing model (SSPM), which includes SSPM insertion. First, it has tested the model's ability to evolve over generations of executions by creating and saving computing modules for frequently executing operations. This case was similar to the Testing Scenario 1 of SSPM-FC. Similarly, this scenario was named. Through this scenario, it is expected to determine whether the system has improved its processing power due to the modifications through generations of program executions. Next, several experimental setups were designed in order to compare the SSPM model with the Quicksort algorithms, which has been implemented using parallel computing, incremental computing [29]/ self-adjusting computing [48], [50], a Dynamically Tuned Library [51] and Sorting with Genetic algorithmic approach [89]. These have been considered under the Testing Scenario 2. The following section onwards, it has been discussed the testing scenarios.

6.8 SSPM Sorting - Testing Scenario 1:

Under this scenario, the effect of "Generating and storing the operating modules for frequent operations through the generations of program executions" was checked and evaluated. Further, this scenario has two subsections; first subsection (Testing Scenario 1.1) is allocated to test "when the current input has some/many (not all similar) similar values to the elements of a previous list and some/many (tested with one new element, however, there could be more like one/ two/ five/ half/ all are new elements) were necessary to be inserted into previously sorted list", the second was "when all the elements of the current input equal to all the elements of a previous input."

6.8.1 Experimental Setup

As stated in the above section dedicated to SSPM-FC the overall hypothesis of this research has reduced to the following hypothesis.

The hypothesis, which was tested in this scenario is;

H_0: There is no difference between the means of time values before and after organizing the tactics memory by organizing smaller tactics memory through continuous processing.

(No Performance Improvement over program execution cycles)

(H_0: $\mu_D = d_0$, $d_0=0$)

H_1: The mean value of the time values collected before organizing memory is greater than the mean value of the time values collected after organizing memory.

(Performance has improved over program execution cycles)

(H_1: $\mu_D > d_0$, $d_0=0$)

6.8.2 Choice of Expressions and The Responses

For each of the sub scenarios, lists of 100 integers in-between 0 and 1000 have been randomly generated. For examples, to check *InsCalcModule*, a set of 75 lists of 100 integers have been used before and after creating the module. Then, those list have been sorted before the modification and collected the time taken by each expression for the execution in nanoseconds. In the same way, after the modification, the time values have been collected for the same set of expressions in nanoseconds. This collection process has been conducted in subsequent sorting cycles.

6.8.3 Testing Scenario 1.1: InsCalcModule

Under this scenario the modification in the SSPM-Sorting was “create and save the computing module for frequently executing insertion associated sorting over execution cycles”.

Step 1:

As mentioned in the above section, a set of 75 lists of 100 integers (using MedCalc, the required minimum sample size was determined as 7) have been sequentially sorted before and after the modification. The time taken during the each of lists sorting has been recorded in nanoseconds. Hence, two sample lists of times values have been collected. As seen in the Figure 6.18, it was able to see a clear difference between the time values of two samples.

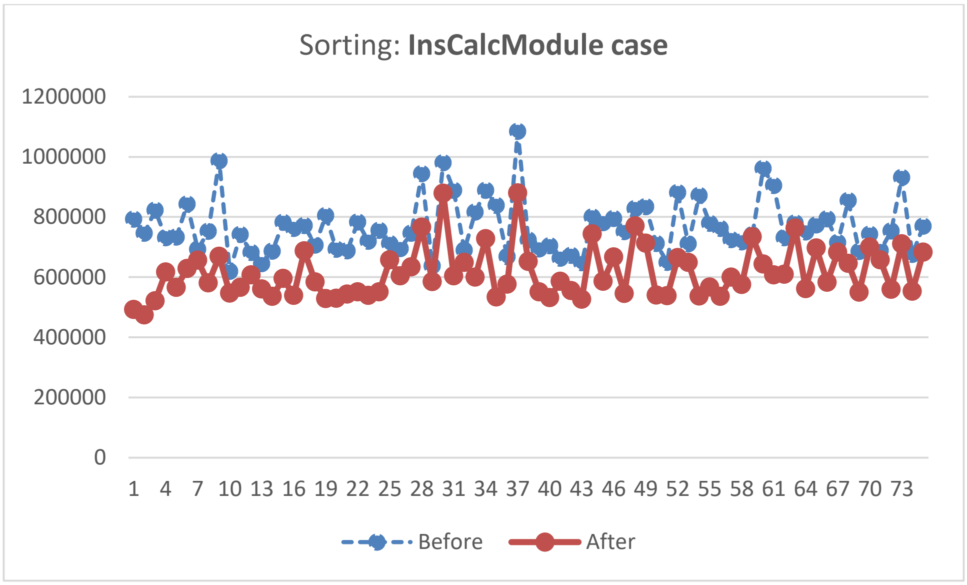


Figure 6.18: Graph of Time values recorded for sorting lists (InsCalcModule) before and after organizing the smaller tactics memory of the sorting program.

Descriptive Statistics:

Mean Values:

Before modification = 766052 (ns)

After modification= 612507 (ns)

Step 2:

Then as the second step, the samples have been analysed to determine which test could be applied in these paired samples. For that, it has been necessary to determine whether the samples satisfy the above stated conditions.

These paired samples are in continuous scale as those have been collected at time intervals in nanoseconds.

After applying Anderson-Darling test for the differences of the paired samples at significance level of 0.05, it has been obtained the P-value as 0.1104.

For the Anderson Darling Test:

Null hypothesis	H_0	: Differences of the paired samples follow a normal distribution.
Alternative hypothesis	H_1	: Differences of the paired samples do not follow a normal distribution.

Since, the P-value is greater than the significance level (0.1104>0.05), the null hypothesis cannot be rejected. Therefore, Differences of the paired samples follow a normal distribution.

Next, it was necessary to check for outliers in the paired samples. The Grubb's test has been used for this purpose with the significance level 0.05.

For the Grubb's Test:

Null hypothesis	: All data values come from the same normal population.
Alternative hypothesis	: Smallest or largest data value is an outlier.

Here also, the critical value obtained is 3.283 and it is greater than the significance level 0.05. Thus, it has been decided that the null hypothesis cannot be rejected. Therefore, the differences of paired samples have no outliers.

Since, all the three conditions are satisfied by the differences of the paired samples, the paired-t test could be applied for the analysis.

Step 3:

After checking the conditions and power with the given sample size, finally the paired-t test has been applied for the two dependent samples with 99% significance level.

Test Statistics;

$$T = \frac{\bar{D} - \mu_D}{S_d/\sqrt{n}}$$

$$T_{\text{calculated}} = t = \frac{\bar{d} - d_0}{s_d/\sqrt{n}}$$

$\bar{D}$=Random variable of mean of differences of values

μ_D=Mean of Differences of population

$\bar{d}$=Mean of Differences of values (Estimate of $\bar{D}$)

S_d=Random variable of standard deviation of differences of values

s_d=standard deviation of differences of values (Estimate of S_d)

d_0 = Population mean

n = Sample Size

Applying paired-t Test for the difference between Before Create and After Create the InsCalc module:

Table 6.20: Values for The Paired-T Test (InsCalcModule)

	N	Mean (ns)	StDev (ns)	SE Mean (ns)
Before Create	75	766052	91253	10537
After Create	75	612507	82545	9531
Difference	75	153545	82511	9528

From Table 6.20;

$s_d/\sqrt{n}$ = SE Mean = 9528 (ns)

s_d = StDev =82511 (ns)

$\bar{d}$=153545 (ns)

n = N =75

t-test of mean difference = 0 (vs > 0): T-Value = 16.115

Hence, $T_{calculated} = \bar{d}/(s_d/\sqrt{n}) = 16.115$

$T_{table}(74,0.05) = 1.667$,

Where, degree of freedom = N-1 = 74

Thus, $T_{calculated} > T_{table}$

Therefore, the null hypothesis can be rejected and it is possible to conclude that there is a significant improvement in the system after the modification as the mean value of the sample collected before the modifications (Create and Save the computing modules for frequently executing quadratic equations with positive discriminant) is greater than the mean value of the sample collected after the modification.

Hence, it was proved that the system would become more efficient in succeeding execution cycles due to the conditionally evolving memory.

6.8.4 Testing Scenario 1.2: Similar List to Sort (EquCalcModule)

Under this scenario the modification in the SSPM-Sorting was "create and save the computing module for frequently sorting the similar list over execution cycles".

Step 1:

Here also, 74 lists of 100 elements (using MedCalc, the required minimum size was determined as 4) have been randomly generated and used for the testing and scenario is exactly similar to the above testing scenario 1.1. As seen in Figure. 6.19, it was able to see a considerable difference between the time values of two populations.

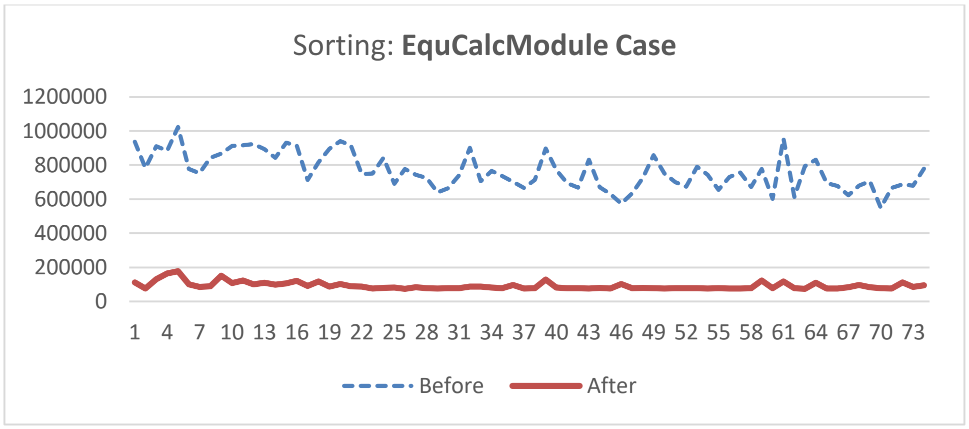


Figure 6.19: Graph of Time values recorded for sorting lists (Same List) before and after organizing the smaller tactics memory of the Sorting program.

Descriptive Statistics:

Mean Values:

Before the modification= 765803 (ns)

After the modification= 91578 (ns)

Step2:

These paired samples are in continuous scale as those have been collected at time intervals in nanoseconds.

After applying Anderson-Darling test for the differences of the paired samples at significance level of 0.05, it has been obtained the P-value as 0.0632.

For the Anderson Darling Test:

Null hypothesis	H_0	: Differences of the paired samples follow a normal distribution
Alternative hypothesis	H_1	: Differences of the paired samples do not follow a normal distribution

Since, the P-value is greater than the significance level (0.0632>0.05), the null hypothesis cannot be rejected. Therefore, Differences of the paired samples follow a normal distribution.

Next, it was necessary to check for outliers in the paired samples. The Grubb's test has been used for this purpose with the significance level 0.05.

For the Grubb's Test:

Null hypothesis	: All data values come from the same normal population.
Alternative hypothesis	: Smallest or largest data value is an outlier.

Here also, the critical value obtained is 3.278 and it is greater than the significance level 0.05. Thus, it has been decided that the null hypothesis cannot be rejected. Therefore, the differences of paired samples have no outliers.

Since, all the three conditions are satisfied by the differences of the paired samples, the paired-t test could be applied for the analysis.

Step 3:
After checking the conditions, and power with the given sample size, finally the paired-t test was applied for the two dependent samples with 99% significance level.

Paired-t Test for the difference between Before Creating and After Creating the particular module:

Table 6.21: Values for The Paired-T Test (EquSortModule)

	N	Mean (ns)	StDev (ns)	SE Mean (ns)
Before modification	74	765803	105742	12292
After modification	74	91578	21444	2493
Difference	74	674225	93875	10,916

From Table 6.21;

$s_d/\sqrt{n}$ = SE Mean = 10,916 (ns)

s_d = StDev =93875 (ns)

$\bar{d}$=674225 (ns)

n = N =74

t-test of mean difference = 0 (vs > 0): T-Value = 61.784

Hence, $T_{calculated} = \bar{d}/(s_d/\sqrt{n})$ = 61.784

$T_{table}(73,0.05)$ = 1.667, where

degree of freedom = N-1 = 73

Thus, $T_{calculated} > T_{table}$

Therefore, the null hypothesis can be rejected and it is possible to conclude that there is a significant improvement in the system after the modification as the mean time value collected before the modifications (Create and Save the computing modules for frequently executing modules) is greater than the mean value of the sample collected after the modification.

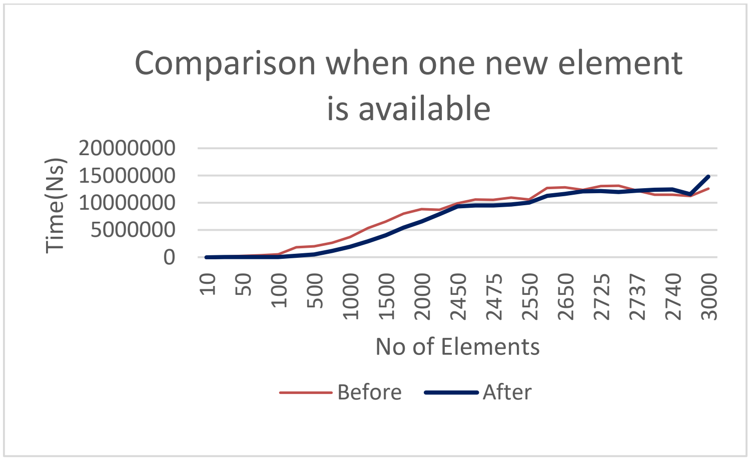
Hence, it was proved that the SSPM-Sorting would become more efficient in succeeding execution cycles due to the conditionally evolving memory.

6.9 Trade-Offs: SSPM Sorting Vs Original Quicksort-Testing Scenario 1

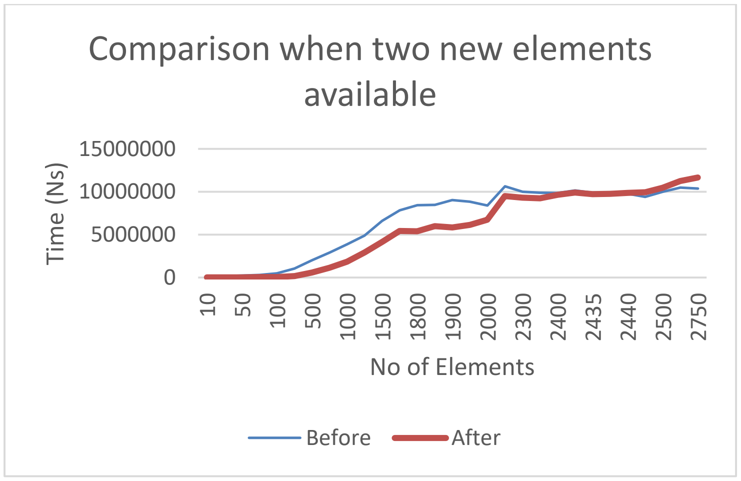
When the number of elements in the current list was ranged from 2 to 2300, SSPM insertion showed better performance for any number of new elements in the current list with compared to the previous list than purely applying the original Quicksort on the current list. For example, when only one new element was present in the current list compared to the previous list, SSPM insertion showed better performance, up to 2737 elements. However, there was no performance gain at all afterwards (see the Figure. 6.20

(a)). Similarly, it was up to 2439 for two new elements (see the Figure. 6.20 (b)) and for five elements it was up to 2344 (see the Figure. 6.20 (c)).

(a)



(b)



(c)

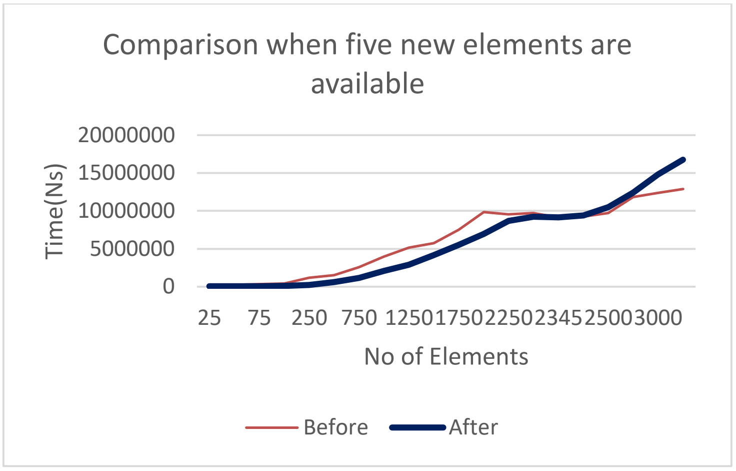


Figure. 6.20: Comparison when (a) one new element was available (b) two new elements were available (c) Five new elements were available.

However, when the number of elements in the current list rose, this pattern was changed, where the better performance appeared only when "some percentage" of the current list was newer than the previous list as in the Figure. 6.21. To understand this "some percentage", let's consider an example: There received a current set Y, which consists of 100,000 elements. Then 10% or less than of this set Y are similar to the elements in a previously received set X, i.e., 90% or more of the set Y are newer than the previously received set X, where input pattern of Y belonged to $IP1$: $X \subset Y$ or $IP2$: $X \nsubseteq Y$ and $Y \nsubseteq X,$ and $X \cap Y \neq \phi$ or $X \cap Y = \phi$. When the SSPM insertion was applied for the set Y, it sorted the set Y faster than applying original Quicksort on Y. However, if the current set with 100,000 elements had 50% or less of new elements than a previous set, SSPM insertion is not better than the original Quicksort. Such a way, each number of elements in a list had a particular threshold-percentage for having new elements, which enables the SSPM to perform better. Then, Figure. 6.21 shows how threshold-percentage of new elements, which contributed to the better performance by the SSPM insertion changed across the total number of element in a current list. Finally, when the total number of elements of a list was greater than or equal 25,000, this threshold-percentage becomes a constant, which is equal to 90%.

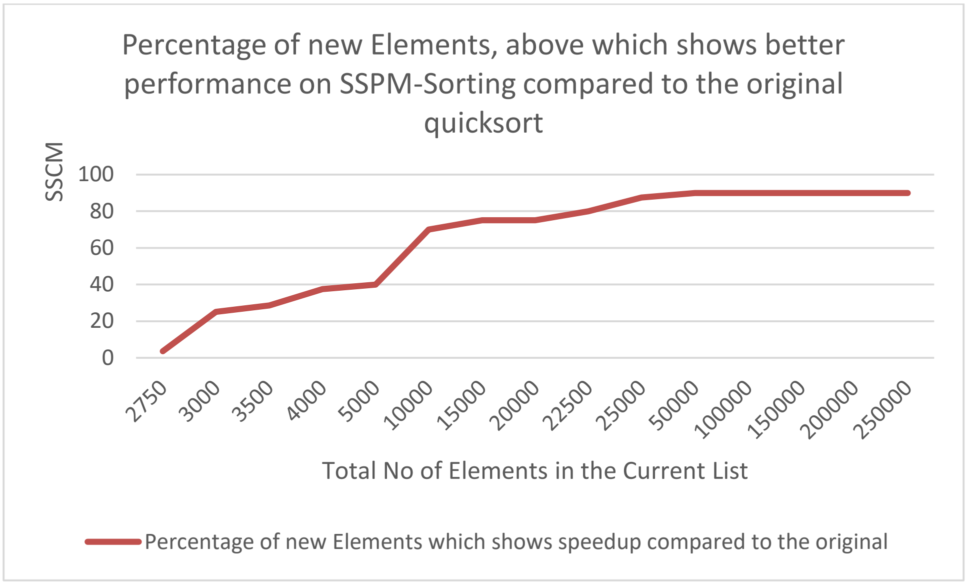
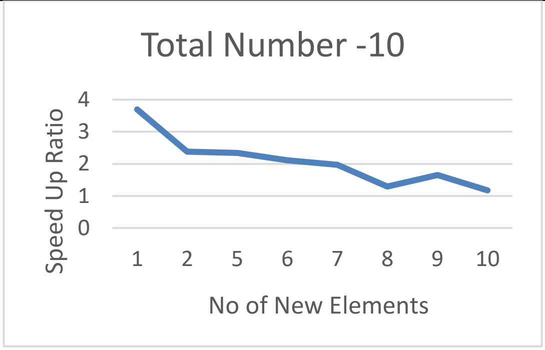
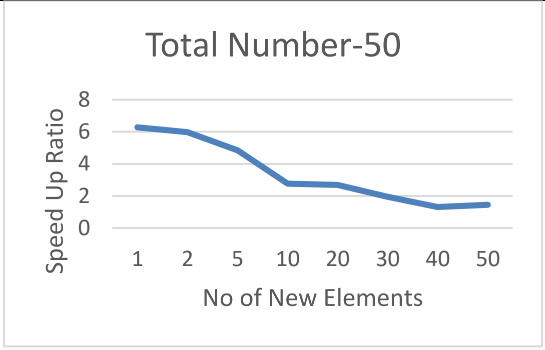
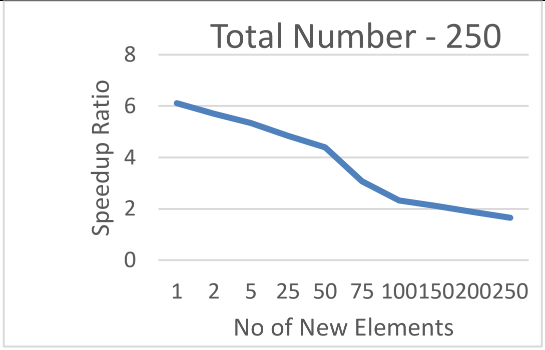


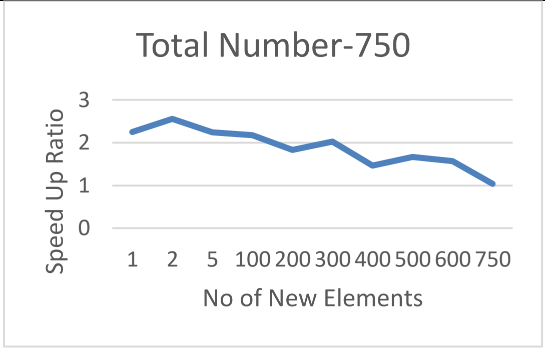
Figure 6.21: Percentage of new Elements in each list, above which showed better performance on SSPM compared to the original

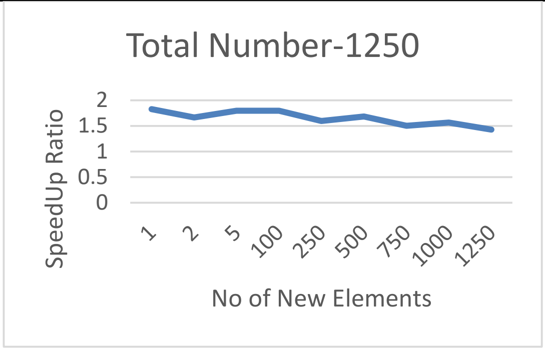
The better performance has illustrated by calculating the speedup. Here, the speedup was calculated by dividing the time (Ns) taken by the original quicksort by the time taken by SSPM-Sorting. If the newer percentage of elements of a list was getting bigger for a given total number of elements, the speedup compared to the original was slowly decreasing in the each of the lists of approximately ranged from 2 elements to 1,250 elements (Figure 6.22). Next, showed a stability between 1,250 and 1,750 exclusively (Figure 6.22). Then, slowly started to rise from 1,750, but with a small drop when the entire list is new as showed in the Figure 6.22.

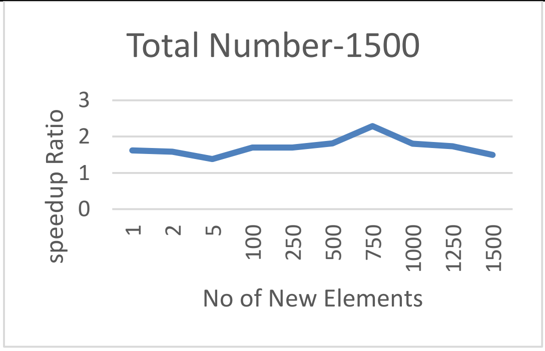

Total Number -10
Speed Up Ratio
4
3
2
1
0
1 2 5 6 7 8 9 10
No of New Elements

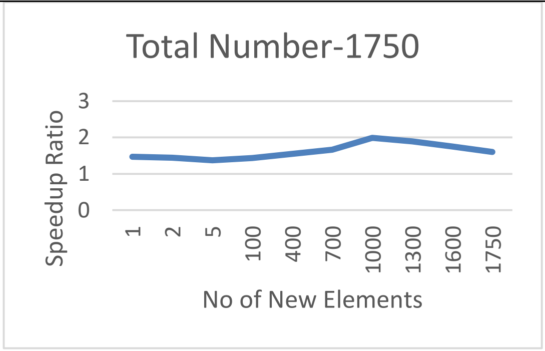

Total Number-50
Speed Up Ratio
8
6
4
2
0
1 2 5 10 20 30 40 50
No of New Elements

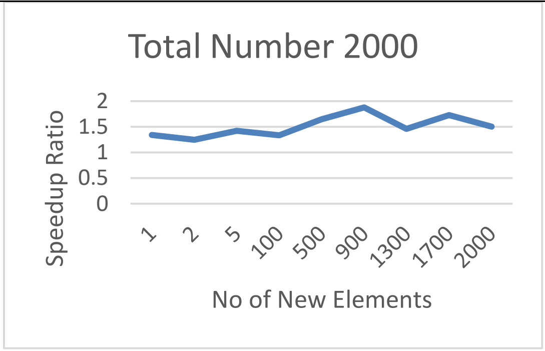

Total Number - 250
Speedup Ratio
8
6
4
2
0
1 2 5 25 50 75 100150200250
No of New Elements

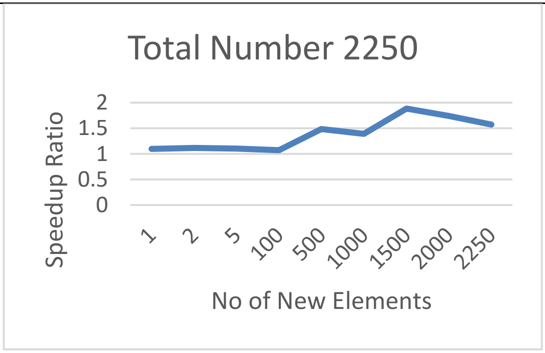

Total Number-750
Speed Up Ratio
3
2
1
0
1 2 5 100 200 300 400 500 600 750
No of New Elements

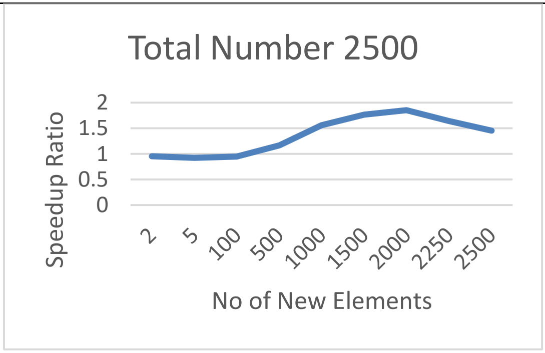

Total Number-1250
SpeedUp Ratio
2
1.5
1
0.5
0
1 2 5 100 250 500 750 1000 1250
No of New Elements

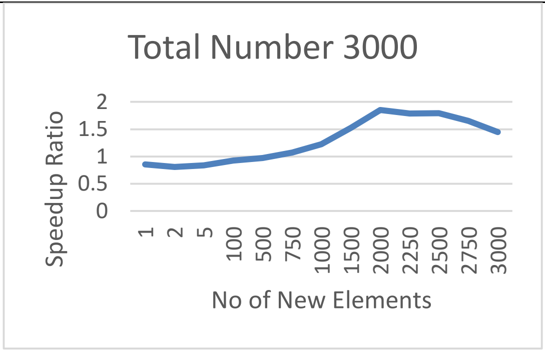

Total Number-1500
speedup Ratio
3
2
1
0
1 2 5 100 250 500 750 1000 1250 1500
No of New Elements

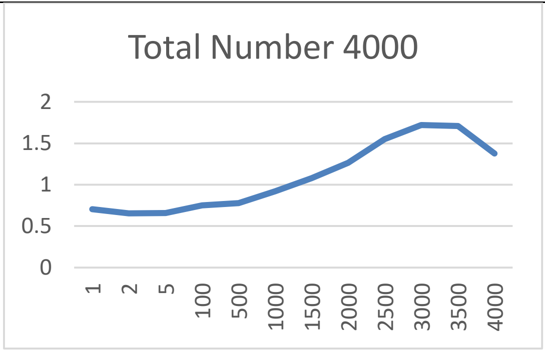

Total Number-1750
Speedup Ratio
3
2
1
0
1 2 5 100 400 700 1000 1300 1600 1750
No of New Elements

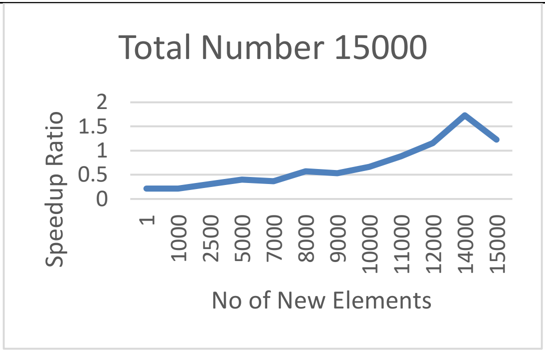

Total Number 2000
Speedup Ratio
2
1.5
1
0.5
0
1 2 5 100 500 900 1300 1700 2000
No of New Elements

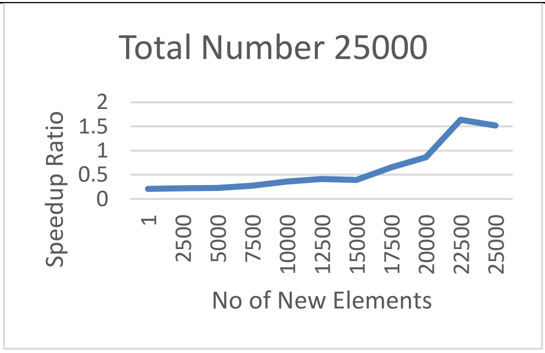

Total Number 2250
Speedup Ratio
2
1.5
1
0.5
0
1
2
5
100
500
1000
1500
2000
2250
No of New Elements

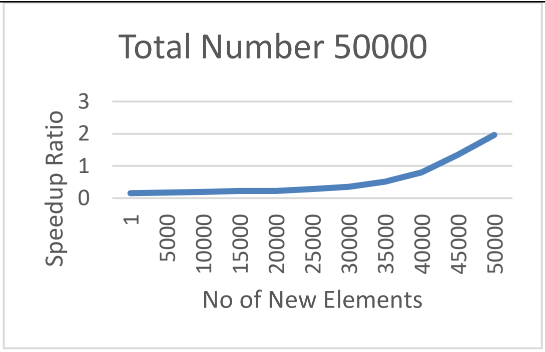

Total Number 2500
Speedup Ratio
2
1.5
1
0.5
0
2
5
100
500
1000
1500
2000
2250
2500
No of New Elements

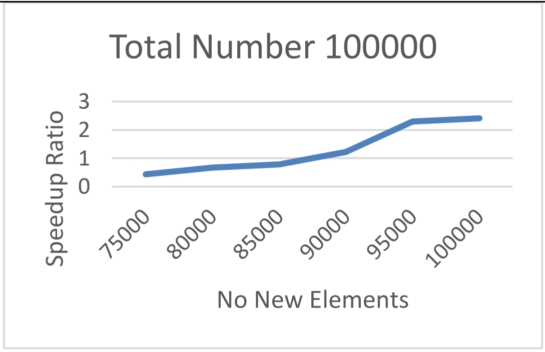

Total Number 3000
Speedup Ratio
2
1.5
1
0.5
0
1
2
5
100
500
750
1000
1500
2000
2250
2500
2750
3000
No of New Elements


Total Number 4000
2
1.5
1
0.5
0
1
2
5
100
500
1000
1500
2000
2500
3000
3500
4000


Total Number 15000
Speedup Ratio
2
1.5
1
0.5
0
1
1000
2500
5000
7000
8000
9000
10000
11000
12000
14000
15000
No of New Elements


Total Number 25000
Speedup Ratio
2
1.5
1
0.5
0
1
2500
5000
7500
10000
12500
15000
17500
20000
22500
25000
No of New Elements


Total Number 50000
Speedup Ratio
3
2
1
0
1
5000
10000
15000
20000
25000
30000
35000
40000
45000
50000
No of New Elements


Total Number 100000
Speedup Ratio
3
2
1
0
75000
80000
85000
90000
95000
100000
No New Elements

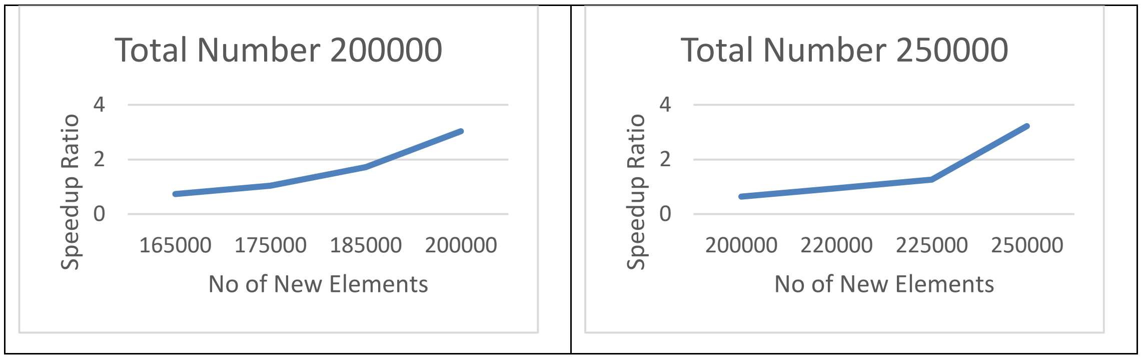


Figure. 6.22: How speedup of SSPM insertion varies depending on the percentage of new elements within a list

6.10 SSPM Sorting - Testing Scenario 2

Other comparisons have stated under this scenario. There are three main subsections; first subsection compares the SSPM-Sorting with Parallel Quicksort, then the second section compares it with the incremental computing and the self-organizing computing, finally, the last section was allocated to compare the SSPM sorting with Dynamically tuned library for sorting and sorting with Genetic algorithmic approach.

6.10.1 SSPM Sorting - Testing Scenario 2.1

This section compares the speedup of the SSPM-Sorting with the speedup of the parallel Quicksort [72]. (Here the speedups are calculated with respect to the original quicksort algorithm). Here, it has randomly generated nine lists with the number of elements: 10, 100, 1,000, 10,000, 25,000, 50,000, 75,000, 100,000, and 150,000. Then, the time taken for each sorting has recorded in milliseconds, before and after the modifications. The obtained values are recorded as in Table 6.22.

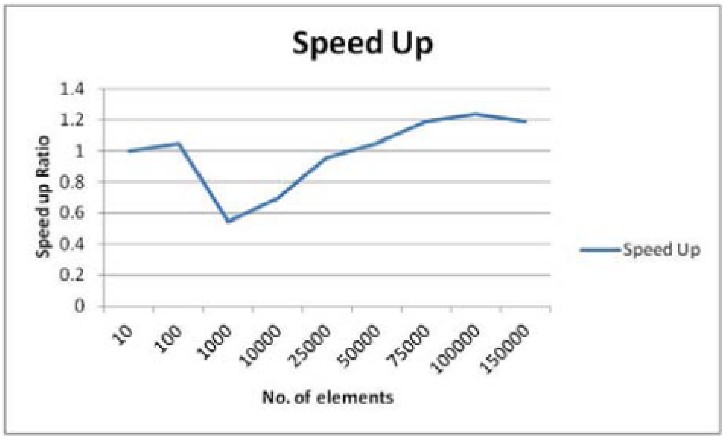
Table 6.22. Comparison Tables (a) Average run times for different thresholds and number of elements for parallel QS (Source: [72]), (b) Relevantly tested SSPM, sorting list results with original QS, when there are 1, half and all new, all equal elements than/to previous list in SSPM

(a) No of elements	T=1,000 (ms)	T=5,000 (ms)	T=50,000 (ms)
10	0.01	0 0.01	0001
100	0.01	0.020001	0.050004
1000	0.250016	0.270011	0.260018
10000	2.010118	2.880166	3.060169
25000	5.380318	6.120344	9.15052
50000	11.36065	11.320644	19.61112
75000	18.14103	18.251045	28.60164
100000	24.91142	22.591294	34.19196
150000	36.41208	34.551976	46.99269

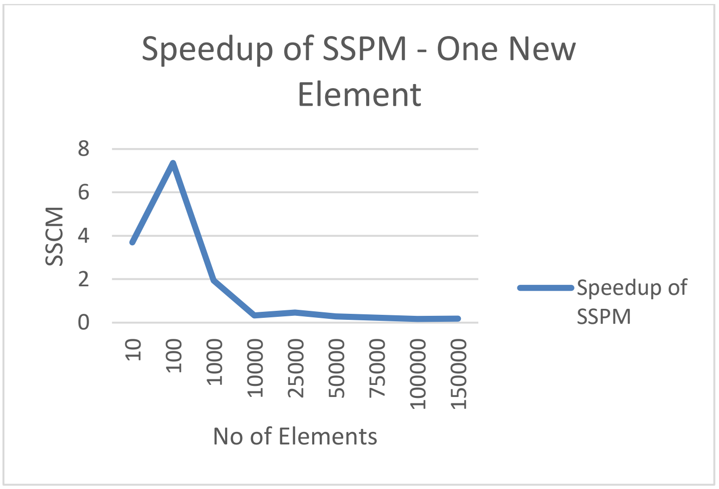
(b) No of Elements	QS (ms) (Original)	SSPM (Ins) Sorting (ms) (New-1)	SSPM (Ins) Sorting (ms) (New-All)	SSPM (Ins) Sorting (ms) (New-Half)	SSPM (Equ) Sorting (ms)
10	0.066632	0.018038	0.03806067	0.0391916	0.003218
100	0.539753	0.07334	0.3489822	0.24314837	0.002344
1,000	3.739901	1.930715	3.8016484	2.39315618	0.013139
10,000	55.54588	170.6803	30.75075283	58.83381293	0.0598
25,000	110.98429	235.13991	105.1489939	467.412058	0.336187
50,000	284860552	975723774	260.123507	1514.949977	0.542807
75,000	467.85857	2011.65497	449.4495079	3625.401822	0.597209
100,000	678.709937	4018.92315	719.3368731	7379.429526	1.139867
150,000	1534.8744	8671.315424	1338.664077	16466.73955	1.436503

The speedup of relevant methods compared to the original methods were considered as in Figure. 6.23. Since, these speedups are in different scales taking those into a single graph would minimize the visibility of their variations.

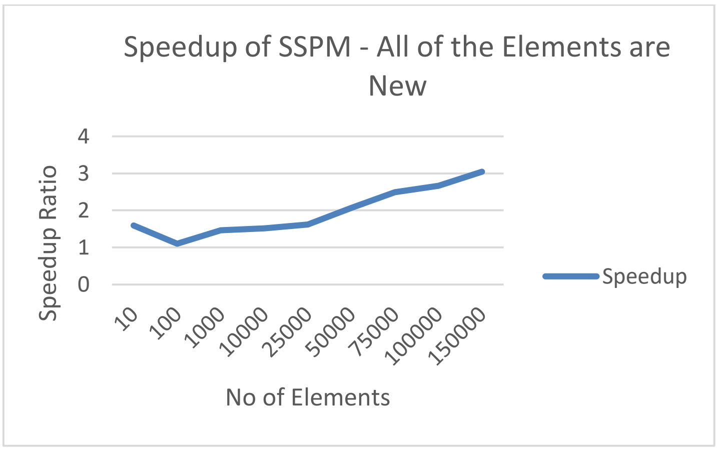
(a)


Speed Up
Speed up Ratio
1.4
1.2
1
0.8
0.6
0.4
0.2
0
10
100
1000
10000
25000
50000
75000
100000
150000
No. of elements
Speed Up

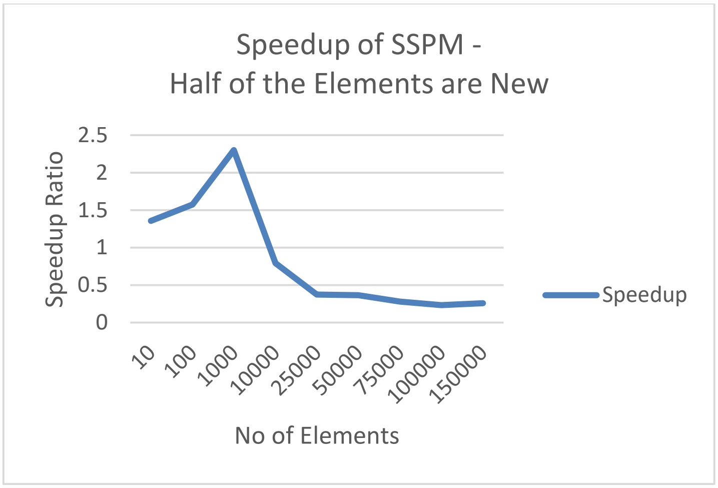
(b)


Speedup of SSPM - One New Element
SSCM
8
6
4
2
0
10
100
1000
10000
25000
50000
75000
100000
150000
No of Elements
Speedup of SSPM

(c)


Speedup of SSPM - All of the Elements are New
Speedup Ratio
4
3
2
1
0
10
100
1000
10000
25000
50000
75000
100000
150000
No of Elements
Speedup

(d)



(e)

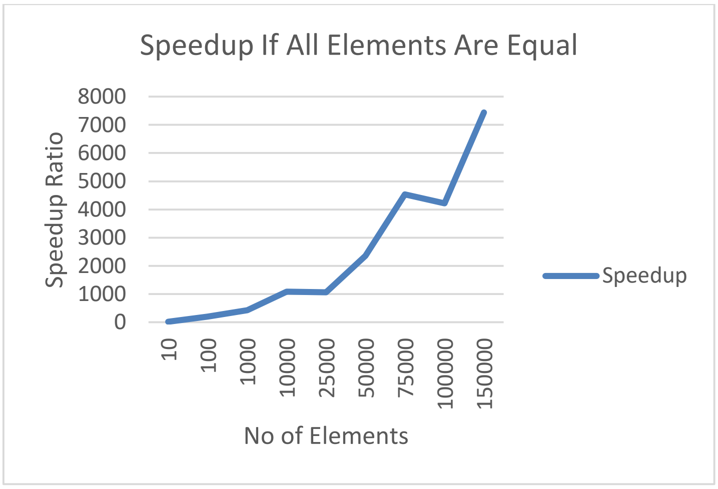


Figure. 6.23: (a) Graph showing speed up ratio by using parallel quicksort. (b) Speedup of the SSPM Sorting when one new element available (c) Speedup of the SSPM Sorting when all are new elements (d) Speedup of the SSPM Sorting when half of the demands are new elements (e) Speedup of the SSPM Sorting when all elements are equal.

6.10.2 SSPM Sorting - Testing Scenario 2.2

This section compares the SSPM sorting with the Quicksort with self-adjusting computing [50] and the incremental computing [29]. The testing results obtained by the respective researchers have been compared here with the results obtained through executing different SSPM-Sorting techniques as seen in the tables Table 6.23 and 6.24.

Case 1: Comparison with Self adjusting sort.

In this scenario, all the tests had used lists with total number of elements 100,000 as the input and compared the speedup gained by those compared to the original quicksort as seen in the Table 6.19

Table 6.23. (a.) Speedup calculation

(b.) Quicksort with Self-adjusting computing [51] Vs SSPM sorting

(a.)	**No. of New elements in the list.**	**Original (O) (ns)**	**SSPM (S) (ns)**	**Difference (ns)**	**Speedup =O/S**
SSPM Sorting	90%	1617721603	1327240833	290480770.4	1.218860634
	95%	1528648990	668087075.9	860561914.2	2.288098431
	100%	1506086212	624955845.1	881130367.2	2.409908194
	Equal	678709937	1139867	677570070	595.4290606

(b.) **Sorting Technique**	**Size of the list**	**Speedup**
Quicksort with Self-Adjusting Computing	$1*10^5$	654.06
SSPM-S-Insertion (90% new)	$1*10^5$	1.218861
SSPM-S-Insertion (95% new)	$1*10^5$	2.288098
SSPM-S-Insertion (100% new)	$1*10^5$	2.409908
SSPM-S-Equal	$1*10^5$	595.45

Case 2: Comparison with Incremental computing sort.

Under this, it has been considered four approaches, which have been upgraded with the incremental sorting. Here also, the size of the lists used consist of 100,000 elements. In addition to the speedup gained, the utilized maximum heap size used for the comparison is shown in the Table 6.24 below.

Table 6.24. Quicksort with ADAPTON [29] with incremental computing Vs SSPM sorting

Sorting technique	**Size of the list**	**Speedup**	**Maximum utilized heap size (MB)**
Quicksort – LazyBidirectional-Eager	$1*10^5$	21600	162
Quicksort – LazyBidirectional-Lazy	$1*10^5$	2020	162
Quicksort – EagerTotalOrder - Eager	$1*10^5$	245	2680
Quicksort – EagerTotalOrder - Lazy	$1*10^5$	22.9	2680
SSPM-S-Insertion (90% new)	$1*10^5$	1.218861	2626
SSPM-S-Insertion (95% new)	$1*10^5$	2.288098	3127
SSPM-S-Insertion (100% new)	$1*10^5$	2.409908	1347
SSPM-S-Equal	$1*10^5$	595.45	2144

6.10.3 SSPM Sorting - Testing Scenario 2.3

This testing scenario, which compares the SSPM-Sorting with the ‘Dynamically Tuned Library (DTL) for Sorting’ [51] and ‘Sorting with Genetic Algorithmic Approach (GAA)’ [89], was complicated than all the tests conducted so far. The DTL research [51] suggested that the characteristics of input data and some architectural features affect the sorting. Particularly, the distribution of data, standard deviation, number of elements in the list, size of the cache, size of the cache line, and number of registers are among the factors. First, six list of normally distributed 2M ($M=2^{20}$) elements have been created. Each list was created so as to have a single standard deviation (stdv) for all the elements in each list, where those six standard deviations were {100, 1,000, 10,000, 100,000, 1,000,000, and 10,000,000}. Same testing scenario has been conducted in two different computers: Intel(R) Xeon(R) CPU ES-2623 V3 @ 3.00 GHz with Turbo Boost up to 2.0GHz with 16GB cache size, 64B cache line size and 4 registers in SUSE Linux (Server) (Figure. 6.24), and Intel(R) Core i7-8550U 1.8GHz with Turbo Boost up to 4.0GHz with 4608MB cache size, 64B cache line size and 8 registers in Windows 10 operating system (Laptop) (Figure. 6.25). There is an apparent speedup gain in the SSPM-

Sorting for the lists with a standard deviation approximately less than 1,000 as shown in the figures Figure. 6.24 and Figure. 6.25.

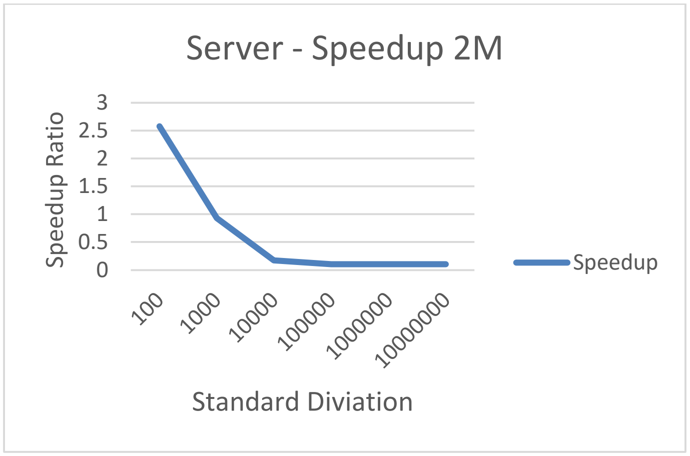


Figure. 6.24: Speedup ratio, when 2M elements are there in the list (Server)

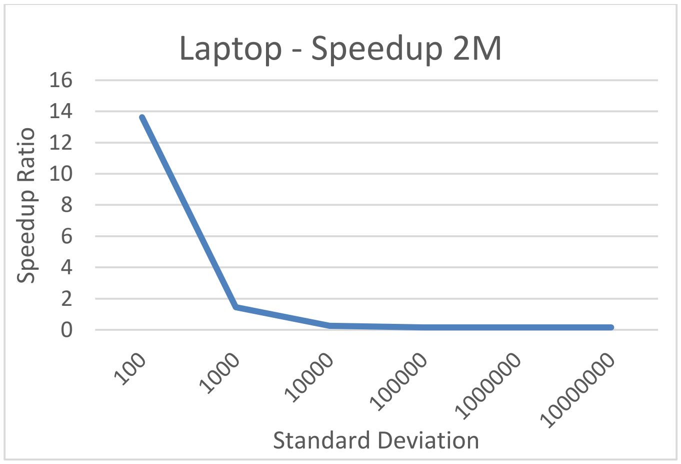


Figure. 6.25: Speedup ratio, when 2M elements are there in the list (Laptop)

Then, again the SSPM-Sorting has been compared with the different improvements gained by the Quicksort after applying different adapting techniques through Dynamically Tuned Library (DTL) and a Genetic Algorithmic Approach (GAA) for sorting (Gene-Sort) [89] in an Intel PIII Xeon computer with 512KB cache size in RedHat 7.3 Operating System as seen in Table 6.25.

Table 6.25: (a.) Speedup Calculation (b.) Comparisons of SSPM sorting with DTL [51] and GAA [89] sorting.

(a.)	**No. of New elements in the list.**	**Original (O) (ns)**	**SSPM (S) (ns)**	**Speedup =O/S**
Server SSPM Sorting	100%	1.72765E+12	1.99998E+11	8.638342601
	95%	1.72143E+12	2.73105E+11	6.303186834
	90%	1.72732E+12	5.30574E+11	3.255558307
Laptop SSPM Sorting	**No. of New elements in the list.**	**Original (O) (ns)**	**SSPM (S) (ns)**	**Speedup =O/S**
	100%	29261182771	3418760879	8.55900246
	95%	28500803520	3569688935	7.984114033
	90%	29699991415	7619283828	3.898003026
	Equal	29683523901	45412649	653.6400002

(b.) **Sorting Technique**	**Speedup**
DTL – Insert Sort at the end	1.1173
DTL – Insert Sort at each partition	1.0465
DTL – Sorting Networks	1.1672
GAA – Gene Sorting	2.5714
SSPM-S-Insertion (90% new) (Server)	3.25556
SSPM-S-Insertion (95% new) (Server)	6.30319
SSPM-S-Insertion (100% new) (Server)	8.63834
SSPM-S-Insertion (90% new) (Laptop)	3.898003
SSPM-S-Insertion (95% new) (Laptop)	7.984114
SSPM-S-Insertion (100% new) (Laptop)	8.559002
SSPM-S-Equal (0% new) (Laptop)	653.64

6.11 Formal Verification

In this work, the new processing model is formally simulated on a Turing Machine (TM) for the formal verification of the model. Then, the model has been classified into the polynomial hierarchy by checking satisfiability. Hence, its real-world applicability has been proved in theoretical level.

6.11.1 Why the Turing Machine has been Used?

Turing machine (TM) is an abstract machine that was introduced nearly eighty years ago by Alan Turing. Further, a TM has been capable of doing anything that a real computer can do, and has been more precise [56] and accurate. Moreover, the tape of TM has been divided into cells and these cells can hold any one of a finite number of symbols called input alphabet. For this system, the input is a finite-length string of symbols chosen from the input alphabet. In addition to that, TM has a pointer, which enables the finite control and is named as 'head'. Further, it reads the tape symbols. Accordingly, at a given time, the system is in any one of finite set of states. Furthermore, the system will be stopped, when the TM finishes reading all the symbols in the string. TM has few variations such as Multiple tape Turing Machines, and Nondeterministic Turing Machines. Further, the Pushdown automata is a computational machine that accepts language, whereas the TM can be used both as a language accepter and a transducer. There are two similarities exist with the Turing Machine and the Pushdown Automata [138]. First similarity is both are Finite-state machines. The second is both have Deterministic and Nondeterministic Machines. Further, this has many features those differentiate this with the other such machines [137]. There, the TM can read from the tape and the write on the tape, and the read-write head can move to both left and right. Moreover, this has an unlimited and unrestricted memory since having an infinite tape, and the effect would be immediately taken place as the special states are there for rejecting and accepting. The most of all, the TM can be used to determine the nature of a problem [138], which includes decidability and satisfiability of the problem. Furthermore, if an algorithm is Turing Complete, then it is applicable in the real-world.

6.11.2 Nondeterministic Turing Machine (NTM)

For the SSPM, a Nondeterministic Turing Machine (NTM) (T) has been designed as a Transducer as seen in figure. 6.26. However, when designing the NTM, it is required to split the 'Ready' state in the new computing model into four states as 'Classified', 'Prioritized', 'Ready', and 'deleted', since the actions in the transitions should be expressed distinctly. Therefore, TM consists of ten states including initial state.

This section discusses the mathematical model designed to simulate the storage and the flow of the proposed continuous processing model. It is a Nondeterministic Turing Machine (NTM) T define by a 7-tuple and a single tape in place of the storage or memory.

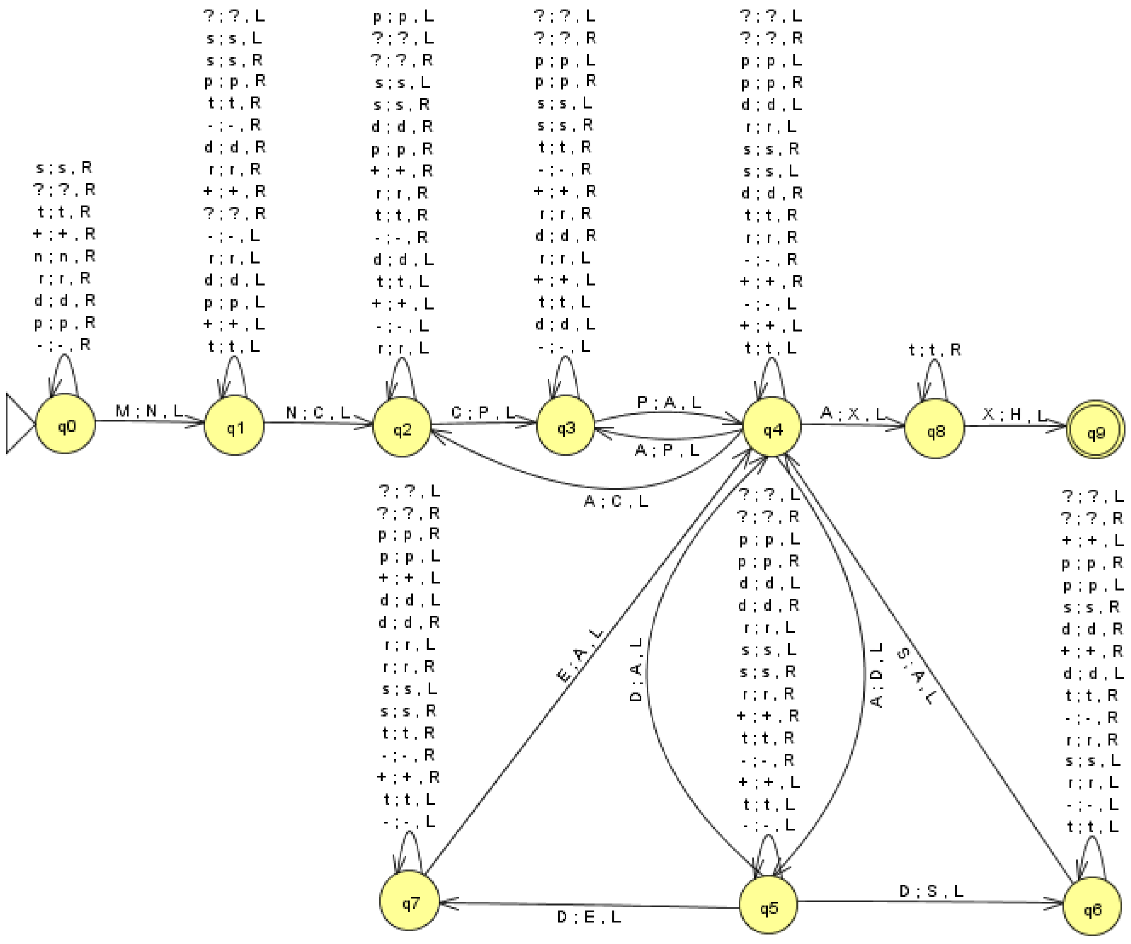


Figure. 6.26: Transition Diagram

Further, the T was designed as a transducer, which produces an output for a given input according to the proposed model. T has 10 internal states. Moreover, the input alphabet consists of 10 symbols and the tape can have 20 symbols, which consists of all the actions, changes in the storage, and the blank symbol. In addition to that, the initial state is q_0, the final state is q_9, and the blank is denoted by B.

The Turing Machine T is given by $T=(Q,\sum,\Gamma,\delta,q_0,B,F)$, where

- $Q=\{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_9\}$ (Finite set of Internal States -All Processing States) where q_0= Initial q_1= New, q_2= Classified, q_3= Prioritized, q_4= Ready, q_5= Running, q_6= Blocked, q_7= Sleep, q_8=Deleted, q_9=Terminate
- $\sum=\{n,?,p,+,d,-,s,r,t,M\}$ (Input Alphabet–All the sub processes), where n=Create, ?=Classify, p=Prioritize, +=Activate, d=Dispatch, -=Delete, s=Event-wait/ Block, t=No further improvements, M=Initial Memory.
- $\Gamma= \{n,?,p,+,d,-,s,r,t,M,N,C,P,R,D,X,S,E,H,B\}$ Tape Symbols, where $\sum\subseteq\Gamma\backslash\{B\}$, and N,C,P,R,D,X,S,E,H are the changed Memories after the actions such as Create, Classify, Prioritize, Activate, Dispatch, Delete, Event-Wait/Block, Release, and No-Improvements respectively.
- δ = Transition function, where $\delta(q,X)=(p,Y,D)$, $q\in Q$ is the current state, $X\in\Gamma$, $p\in Q$ is the next state, $Y\in\Gamma$ is the sub process which replaces the scanned sub process on the tape, D is the Direction (left or right) of the head to move (all transitions are tabulated into Table 6.26).
- $q_0\in Q$ is the start state.
- B: Blank symbol, initially, the input is surrounded by blanks.
- F=Set of Final States, where $F=\{q_9\}$, $q_9\in Q$

Table 6.26: Transition Table

State	Symbol																			
	Actions										Storage									
	B (Blank)	n (Create)	? (Classify)	p (Prioritize)	+(Activate)	d (Dispatch)	- (Delete)	s (Blocked / Event-Wait)	r (Release)	t (Terminate – No	M (Initial Memory)	N (After n)	C (After ?)	P (After p)	A (After +)	D (After d)	X (After -)	S (After s)	E (After r)	T(After t)
(Initial) q_0		(q_0,n,R)	$(q_0,?,R)$	(q_0,p,R)	$(q_0,+,R)$	(q_0,d,R)	$(q_0,-,R)$	(q_0,s,R)	(q_0,r,R)	(q_0,r,R)	(q_1,N,L)									
(New) q_1			$(q_1,?,L)$	(q_1,p,L)	$(q_1,+,L)$	(q_1,d,L)	$(q_1,-,L)$	(q_1,s,L) (q_1,s,R)	(q_1,r,L)	(q_1,t,L)		(q_2,C,L)								
(Classified) q_2			$(q_2,?,L)$	(q_2,p,L)	$(q_2,+,L)$	(q_2,d,L)	$(q_2,-,L)$	(q_2,s,L) (q_2,s,R)	(q_2,r,L)	(q_2,t,L)			(q_3,P,L)							
(Prioritized) q_3			$(q_3,?,L)$	(q_3,p,L)	$(q_3,+,L)$	(q_3,d,L)	$(q_3,-,L)$	(q_3,s,L) (q_3,s,R)	(q_3,r,L)	(q_3,t,L)				(q_4,A,L)						
(Ready) q_4			$(q_4,?,L)$ $(q_4,?,R)$	(q_4,p,L) (q_4,p,R)	$(q_4,+,L)$ $(q_4,+,R)$	(q_4,d,L) (q_4,d,R)	$(q_4,-,L)$ $(q_4,-,R)$	(q_4,s,L) (q_4,s,R)	(q_4,r,L) (q_4,r,R)	(q_4,t,L) (q_4,t,R)					$(q_5,D,L)(q_8,X,L)(q_3,P,L)(q_2,C,L)$					
(Running) q_5			$(q_5,?,L)$ $(q_5,?,R)$	(q_5,p,L) (q_5,p,R)	$(q_5,+,L)$ $(q_5,+,R)$	(q_5,d,L) (q_5,d,R)	$(q_5,-,L)$ $(q_5,-,R)$	(q_5,s,L) (q_5,s,R)	(q_5,r,L) (q_5,r,R)	(q_5,t,L) (q_5,t,R)						$(q_6,S,L)(q_7,E,L)$ (q_4,A,L)				
(Blocked) q_6			$(q_6,?,L)$	(q_6,p,L)	$(q_6,+,L)$	(q_6,d,L)	$(q_6,-,L)$	(q_6,s,L) (q_6,s,R)	(q_6,r,L)	(q_6,t,L)								(q_4,A,L)		
(Sleep) q_7			$(q_7,?,L)$	(q_7,p,L)	$(q_7,+,L)$	(q_7,d,L)	$(q_7,-,L)$	(q_7,s,L) (q_7,s,R)	(q_7,r,L)	(q_7,t,L)									(q_4,A,L)	

(Deleted) q_8							$(q_8,-,R)$			(q_8,t,R)							(q_9,H,L)			
(Terminate)q_9										(q_9,t,R)										

6.11.3 Configurations of NTM

This section develops a notation for the configurations of this Nondeterministic Turing Machine T. These configurations are also called as the array of Instantaneous Descriptions (IDs) and use to formally describe what this NTM does for a typical input [56].

Input (w) = n?p+dr+-tM

Length of the input w = 10.

Tape at the Beginning: Bn?p+dr+-tMBB$\dot{B}$

Table 6.27 shows how the read/write head moves through the tape depending on the input (w). The states in the Table 6.27 shows the position of the head in the tape in each move. Further, there are 83 moves accounted with respect to this particular input with 10 symbols. These moves were formed according to the Transitional Table appeared in Table 6.27. Next, a sample scenario is explained to show how the head moves on the tape.

The initial configuration is α_0. There, the head is in the position of the state q_0. The rest of the symbols of α_0 represent the tape at the beginning. Then consider the second configuration α_1, which represents the first move. There the head reads 'n', which symbolize the action 'Create'. There, the head doesn't do any change on the particular cell and keep the symbol 'n' as it is, according to the transition $\delta(q_0,n)=(q_0,n,R)$ in Table 6.26. The machine is in the state q_0 until the head writes the result of the action 'Create'. That is until α_9. Then after writing in α_{10}, the storage (memory) has been changed from M to N. The state has been changed to q_1. After that, the head moves back on the tape to read the next input '?', which denotes the action 'Classification'. Similarly, the head moves along the tape back and forth to read the input and write the results, until it finishes all the symbols in the input. Further, it is important to consider that this machine is discussed in a high level and it is not explained in the bit level.

Table 6.27: Configurations of the NTM (p(n)=number of moves)

ID	0	1	2	3	4	5	6	7	8	9	10	11
α_0	**q_0**	n	?	p	+	d	r	+	-	t	M	B
α_1	n	**q_0**	?	p	+	d	r	+	-	t	M	B
α_2	n	?	**q_0**	p	+	d	r	+	-	t	M	B
α_3	n	?	p	**q_0**	+	d	r	+	-	t	M	B
α_4	n	?	p	+	**q_0**	d	r	+	-	t	M	B
α_5	n	?	p	+	d	**q_0**	r	+	-	t	M	B
α_6	n	?	p	+	d	r	**q_0**	+	-	t	M	B
α_7	n	?	p	+	d	r	+	**q_0**	-	t	M	B
α_8	n	?	p	+	d	r	+	-	**q_0**	t	M	B
α_9	n	?	p	+	d	r	+	-	t	**q_0**	M	B
α_{10}	n	?	p	+	d	r	+	-	t	**q_1**	N	B
α_{11}	n	?	p	+	d	r	+	-	**q_1**	t	N	B
α_{12}	n	?	p	+	d	r	+	**q_1**	-	t	N	B
α_{13}	n	?	p	+	d	r	**q_1**	+	-	t	N	B
α_{14}	n	?	p	+	d	**q_1**	r	+	-	t	N	B
α_{15}	n	?	p	+	**q_1**	d	r	+	-	t	N	B
α_{16}	n	?	p	**q_1**	+	d	r	+	-	t	N	B
α_{17}	n	?	**q_1**	P	+	d	r	+	-	t	N	B
α_{18}	n	?	**q_1**	P	+	d	r	+	-	t	N	B
α_{19}	n	?	P	**q_1**	+	d	r	+	-	t	N	B
α_{20}	n	?	p	+	**q_1**	d	r	+	-	t	N	B
α_{21}	n	?	p	+	d	**q_1**	r	+	-	t	N	B
α_{22}	n	?	p	+	d	r	**q_1**	+	-	t	N	B
α_{23}	n	?	p	+	d	r	+	**q_1**	-	t	N	B
α_{24}	n	?	p	+	d	r	+	-	**q_1**	t	N	B
α_{25}	n	?	p	+	d	r	+	-	t	**q_1**	N	B
α_{26}	n	?	p	+	d	r	+	-	t	**q_2**	C	B
α_{27}	n	?	p	+	d	r	+	-	**q_2**	t	C	B
α_{28}	n	?	p	+	d	r	+	**q_2**	-	t	C	B
α_{29}	n	?	p	+	d	r	**q_2**	+	-	t	C	B
α_{30}	n	?	p	+	d	**q_2**	r	+	-	t	C	B

α_{31}	n	?	p	+	$\mathbf{q_2}$	d	r	+	-	t	C	B
α_{32}	n	?	p	$\mathbf{q_2}$	+	d	r	+	-	t	C	B
α_{33}	n	?	p	$\mathbf{q_2}$	+	d	r	+	-	t	C	B
α_{34}	n	?	p	+	$\mathbf{q_2}$	d	r	+	-	t	C	B
α_{35}	n	?	p	+	d	$\mathbf{q_2}$	r	+	-	t	C	B
α_{36}	n	?	p	+	d	r	$\mathbf{q_2}$	+	-	t	C	B
α_{37}	n	?	p	+	d	r	+	$\mathbf{q_2}$	-	t	C	B
α_{38}	n	?	p	+	d	r	+	-	$\mathbf{q_2}$	t	C	B
α_{39}	n	?	p	+	d	r	+	-	t	$\mathbf{q_2}$	C	B
α_{40}	n	?	p	+	d	r	+	-	t	$\mathbf{q_3}$	P	B
α_{41}	n	?	p	+	d	r	+	-	$\mathbf{q_3}$	t	P	B
α_{42}	n	?	p	+	d	r	+	$\mathbf{q_3}$	-	t	P	B
α_{43}	n	?	p	+	d	r	$\mathbf{q_3}$	+	-	t	P	B
α_{44}	n	?	p	+	d	$\mathbf{q_3}$	r	+	-	t	P	B
α_{45}	n	?	p	+	$\mathbf{q_3}$	d	r	+	-	t	P	B
α_{46}	n	?	p	+	$\mathbf{q_3}$	d	r	+	-	t	P	B
α_{47}	n	?	p	+	d	$\mathbf{q_3}$	r	+	-	t	P	B
α_{48}	n	?	p	+	d	r	$\mathbf{q_3}$	+	-	t	P	B
α_{49}	n	?	p	+	d	r	+	$\mathbf{q_3}$	-	t	P	B
α_{50}	n	?	p	+	d	r	+	-	$\mathbf{q_3}$	t	P	B
α_{51}	n	?	p	+	d	r	+	-	t	$\mathbf{q_3}$	P	B
α_{52}	n	?	p	+	d	r	+	-	t	$\mathbf{q_4}$	A	B
α_{53}	n	?	p	+	d	r	+	-	$\mathbf{q_4}$	t	A	B
α_{54}	n	?	p	+	d	r	+	$\mathbf{q_4}$	-	t	A	B
α_{55}	n	?	p	+	d	r	$\mathbf{q_4}$	+	-	t	A	B
α_{56}	n	?	p	+	d	$\mathbf{q_4}$	r	+	-	t	A	B
α_{57}	n	?	p	+	d	$\mathbf{q_4}$	r	+	-	t	A	B
α_{58}	n	?	p	+	d	r	$\mathbf{q_4}$	+	-	t	A	B
α_{59}	n	?	p	+	d	r	+	$\mathbf{q_4}$	-	t	A	B
α_{60}	n	?	p	+	d	r	+	-	$\mathbf{q_4}$	t	A	B
α_{61}	n	?	p	+	d	r	+	-	t	$\mathbf{q_4}$	A	B
α_{62}	n	?	p	+	d	r	+	-	t	$\mathbf{q_5}$	D	B
α_{63}	n	?	p	+	d	r	+	-	$\mathbf{q_5}$	t	D	B
α_{64}	n	?	p	+	d	r	+	$\mathbf{q_5}$	-	t	D	B
α_{65}	n	?	p	+	d	r	$\mathbf{q_5}$	+	-	t	D	B
α_{66}	n	?	p	+	d	r	$\mathbf{q_5}$	+	-	t	D	B
α_{67}	n	?	p	+	d	r	+	$\mathbf{q_5}$	-	t	D	B

α_{69}	n	?	p	+	d	r	+	-	**q_5**	t	D	B
α_{70}	n	?	p	+	d	r	+	-	t	**q_5**	D	B
α_{71}	n	?	p	+	d	r	+	-	t	**q_7**	E	B
α_{72}	n	?	p	+	d	r	+	-	**q_7**	t	E	B
α_{73}	n	?	p	+	d	r	+	**q_7**	-	t	E	B
α_{74}	n	?	p	+	d	r	+	**q_7**	-	t	E	B
α_{75}	n	?	p	+	d	r	+	-	**q_7**	t	E	B
α_{76}	n	?	p	+	d	r	+	-	t	**q_7**	E	B
α_{77}	n	?	p	+	d	r	+	-	t	**q_4**	A	B
α_{78}	n	?	p	+	d	r	+	-	**q_4**	t	A	B
α_{79}	n	?	p	+	d	r	+	-	**q_4**	t	A	B
α_{80}	n	?	p	+	d	r	+	-	t	**q_4**	A	B
α_{81}	n	?	p	+	d	r	+	-	t	**q_8**	X	B
α_{82}	n	?	p	+	d	r	+	-	t	**q_8**	X	B
α_{83} (P(n)=83)	n	?	p	+	d	r	+	-	t	**q_9**	H	B

6.11.4 Satisfiability of NTM

The satisfiability of a NTM can be determined by applying the Cook's Theorem.

6.11.4.1 Cook's Theorem

Satisfiability Problem (SAT) is NP-Complete [56], [137]–[139].

According to the proof of the Cook's Theorem, a Boolean Expression $E_{T,w}$ can be derived for the Nondeterministic Turing Machine (NTM) T for a particular input w. The Boolean Expression $E_{T,w}$ can be written as follows;

$E_{T,w} = U \wedge S \wedge N \wedge F$, where

$\wedge$ is used to denote logical AND

U denotes that there is a unique symbol at each cell

S denotes Start of the T

N denotes the Moves according to the rules in T

F denotes the final state of T.

Before proceed in deriving the Boolean Expression, let’s make some clarifications:
Table 6.27 shows the Configurations or the Instantaneous Descriptions (ID) of the NTM T for a particular input w. These configurations are used in deriving the Boolean expression. Further, a single notation can be used to represent the symbols in the Table 6.27 and to denote them as Boolean variables. For example, to represent the X_{ij} (ij^{th} cell) which has A as the content, can be represented with the Boolean variable Y_{ijA}

6.11.4.2 Deriving Boolean Expression

Let’s derive the components of the Boolean Expression separately and take the conjunction of them.

1. *U – There is a unique symbol in each cell*

U = $\bigwedge_{0\le i,j\le p(n)} \sim(Y_{ij\alpha} \wedge Y_{ij\beta})$, where $\alpha \neq \beta$, p(n)=83 $\rightarrow \boxed{1}$

If we refer the Table 6.27, It is visible that each cell has at least one and exactly one symbol.
Therefore, $\forall \propto,\beta \in Q\cup\Gamma$, $\exists Y_{ij\alpha}$ such that $Y_{ij\alpha} \Rightarrow \sim Y_{ij\beta}$ [56], where $\alpha \neq \beta$
And let substitute the values; $Y_{ij\alpha} = 1$, $Y_{ij\beta} = 0$ in above $\boxed{1}$
Then, U = $\bigwedge_{0\le i,j\le p(n)} \sim(1 \wedge 0) = \bigwedge_{0\le i,j\le p(n)} \sim(0) = \bigwedge_{0\le i,j\le p(n)} 1 = 1$

U = 1 $\rightarrow \boxed{2}$

2. *S – The Initial ID/ Configuration appeared in the Table 6.27. (q_0w followed by blanks)*

S =
$Y_{0,0,q0} \wedge Y_{0,1,n} \wedge Y_{0,2,?} \wedge Y_{0,3,p} \wedge Y_{0,4,+} \wedge Y_{0,5,d} \wedge Y_{0,6,r} \wedge Y_{0,7,+} \wedge Y_{0,8,-} \wedge Y_{0,9,t} \wedge Y_{0,10,M} \wedge Y_{0,11,B}$
$\rightarrow \boxed{3}$

3. *F - Finishes Right (Final states)*

F = $Y_{83,9,q_9} \rightarrow \boxed{4}$

4. *N – Next Move is right. (i.e. the Move correctly follows the transition rules of the NTM T)*

$N = \bigwedge_{0 \le i \le p(n)-1} (\bigwedge_{0 \le j \le p(n)} (A_{ij} \vee B_{ij}))$

The expression A_{ij} says:

a) X_{ij} is the state in α_i.

b) There is a move according to the transition rules.

X_{ij+1} is the scanned symbol.

With that, the sequence $X_{ij-1} \wedge X_{ij} \wedge X_{ij+1}$is transformed into $X_{i+1j-1} \wedge X_{i+1j} \wedge X_{i+1j+1}$.

If X_{ij} is the accepting state, then no move will occur and the same sequence will be repeated.

$A_{ij}\ is\ formed\ by\ X_{ij-1} \wedge X_{ij} \wedge X_{ij+1} \wedge X_{i+1j-1} \wedge X_{i+1j} \wedge X_{i+1j+1}$

$A_{00} = Y_{0,0,q_0} \wedge Y_{0,1,n} \wedge Y_{1,0,n} \wedge Y_{1,1,q_0}$

$A_{11} = Y_{1,0,n} \wedge Y_{1,1,q_0} \wedge Y_{1,2,?} \wedge Y_{2,0,n} \wedge Y_{2,1,?} \wedge Y_{2,2,q_0}$

$A_{22} = Y_{2,1,?} \wedge Y_{2,2,q_0} \wedge Y_{2,3,p} \wedge Y_{3,1,?} \wedge Y_{3,2,p} \wedge Y_{3,3,q_0}$

$A_{33} = Y_{3,2,p} \wedge Y_{3,3,q_0} \wedge Y_{3,4,+} \wedge Y_{4,2,p} \wedge Y_{4,3,+} \wedge Y_{4,4,q_0}$

$A_{44} = Y_{4,3,+} \wedge Y_{4,4,q_0} \wedge Y_{4,5,d} \wedge Y_{5,3,+} \wedge Y_{5,4,d} \wedge Y_{5,5,q_0}$

$A_{55} = Y_{5,4,d} \wedge Y_{5,5,q_0} \wedge Y_{5,6,r} \wedge Y_{6,4,d} \wedge Y_{6,5,r} \wedge Y_{6,6,q_0}$

$A_{66} = Y_{6,5,r} \wedge Y_{6,6,q_0} \wedge Y_{6,7,+} \wedge Y_{7,5,r} \wedge Y_{7,6,+} \wedge Y_{7,7,q_0}$

…

$A_{82,9} = Y_{82,8,t} \wedge Y_{82,9,q_8} \wedge Y_{82,10,E} \wedge Y_{83,8,t} \wedge Y_{83,9,q_9} \wedge Y_{83,10,E}$

The expression B_{ij} says:

a) X_{ij} is not the state in α_i.

b) X_{ij-1} and X_{ij+1} are not the states in α_i either, then X_{i+1j} is equal to X_{ij}.

$$B_{ij} = \left(Y_{ij-1q_0} \vee Y_{ij-1q_1} \vee \ldots \vee Y_{ij-1q_9}\right) \vee$$
$$\left(Y_{ij+1q_0} \vee Y_{ij+1q_1} \vee \ldots \vee Y_{ij+1q_9}\right) \vee$$
$$[\left(Y_{ijZ_1} \vee Y_{ijZ_2} \vee \ldots \vee Y_{ijZ_{21}}\right) \wedge$$
$$\{(Y_{ijZ_1} \wedge Y_{i+1jZ_1}) \vee (Y_{ijZ_2} \wedge Y_{i+1jZ_2}) \vee \ldots \vee (Y_{ijZ_{21}} \wedge Y_{i+1jZ_{21}})\}]$$

Here, q_0, q_1, …, q_9 are states and z_1, z_2, …, z_{21} are tape symbols of NTM T.

Here, two special cases j=0 or j=p(n) for both the A_{ij} and B_{ij} must be considered. In one case there is no Y_{ij-1X} variable and in the other case there is no Y_{ij+1X}. Since, the head never moves to the left of its initial position and to the right of its last position, p(n) (=83), particular terms can be eliminated from the expression.

$$N_i = \{(A_{i0} \vee B_{i0}) \wedge (A_{i1} \vee B_{i1}) \wedge (A_{i2} \vee B_{i2}) \wedge \ldots \wedge (A_{i10} \vee B_{i10}) \wedge (A_{i11} \vee B_{i11}) \wedge \ldots \wedge (A_{i83} \vee B_{i83})\}$$

N $= \bigwedge_{0 \le i \le p(n)-1} N_i$ $(= N_0 \wedge N_1 \wedge \ldots \wedge N_{83})$

$= \bigwedge_{0 \le i \le p(n)-1} (\bigwedge_{0 \le j \le p(n)} (A_{ij} \vee B_{ij}))$

$= \bigwedge_{0 \le i \le p(n)-1} \{(A_{i0} \vee B_{i0}) \wedge (A_{i1} \vee B_{i1}) \wedge (A_{i2} \vee B_{i2}) \wedge \ldots \wedge (A_{i10} \vee B_{i10}) \wedge (A_{i11} \vee B_{i11}) \wedge \ldots \wedge (A_{i83} \vee B_{i83})\}$

$= [\{(A_{0,0} \vee B_{0,0}) \wedge \left(A_{0,1} \vee B_{0,1}\right) \wedge \left(A_{0,2} \vee B_{0,2}\right) \wedge \ldots \wedge \left(A_{0,10} \vee B_{0,10}\right) \wedge \left(A_{0,11} \vee B_{0,11}\right) \wedge \ldots \wedge \left(A_{0,83} \vee B_{0,83}\right)\} \wedge$

$\{(A_{1,0} \vee B_{1,0}) \wedge (A_{1,1} \vee B_{1,1}) \wedge (A_{1,2} \vee B_{1,2}) \wedge \ldots \wedge (A_{1,10} \vee B_{1,10}) \wedge (A_{1,11} \vee$
$B_{1,11}) \wedge \ldots \wedge (A_{1,83} \vee B_{1,83})\} \wedge$

$\{(A_{2,0} \vee B_{2,0}) \wedge (A_{2,1} \vee B_{2,1}) \wedge (A_{2.2} \vee B_{2,2}) \wedge \ldots \wedge (A_{2,10} \vee B_{2,10}) \wedge (A_{2,11} \vee$
$B_{2,11}) \wedge \ldots \wedge (A_{2,83} \vee B_{2,83})\} \wedge$

$\{(A_{3,0} \vee B_{3,0}) \wedge (A_{3,1} \vee B_{3,1}) \wedge (A_{3,2} \vee B_{3,2}) \wedge \ldots \wedge (A_{3,10} \vee B_{3,10}) \wedge (A_{3,11} \vee$
$B_{3,11}) \wedge \ldots \wedge (A_{3,83} \vee B_{3,83})\} \wedge$

$\ldots \wedge$

$\{(A_{82,0} \vee B_{82,0}) \wedge (A_{82,1} \vee B_{82,1}) \wedge (A_{82,2} \vee B_{82,2}) \wedge \ldots \wedge (A_{82,10} \vee$
$B_{82,10}) \wedge (A_{82,11} \vee B_{82,11}) \wedge \ldots \wedge (A_{82,83} \vee B_{82,83})\}$

$= [\{(A_{0,0}) \wedge (B_{0,2}) \wedge \ldots \wedge (B_{0,10}) \wedge (\vee B_{0,11}) \wedge \ldots \wedge (B_{0,82})\} \wedge$
$\{((A_{1,1}) \wedge (B_{1,3}) \wedge \ldots \wedge (B_{1,10}) \wedge (B_{1,11}) \wedge \ldots \wedge (B_{1,82})\} \wedge$
$\{((B_{2,1}) \wedge (A_{2.2}) \wedge \ldots \wedge (B_{2,10}) \wedge (B_{2,11}) \wedge \ldots \wedge (B_{2,82})\} \wedge$
$\{(B_{3,0}) \wedge (B_{3,1}) \wedge (A_{3.3}) \wedge \ldots \wedge (B_{3,10}) \wedge (B_{3,11}) \wedge \ldots \wedge (B_{3,82})\} \wedge$
$\ldots \wedge$
$\{(B_{82,0}) \wedge (B_{82,1}) \wedge (B_{82,2}) \wedge \ldots \wedge (B_{82,10}) \wedge (B_{82,11}) \wedge \ldots \wedge (A_{82,82})\}$

$=$
$[\{(\mathrm{Y}_{0,0,\mathrm{q}_0} \wedge \mathrm{Y}_{0,1,n} \wedge \mathrm{Y}_{1,0,n} \wedge \mathrm{Y}_{1,1,\mathrm{q}_0}) \wedge (Y_{0,2,?} \wedge Y_{1,2,?}) \wedge (Y_{0,3,p} \wedge Y_{1,3,p}) \wedge (Y_{0,4,+} \wedge Y_{1,4,+}) \wedge$
$(Y_{0,5,d} \wedge Y_{1,5,d}) \wedge (Y_{0,6,r} \wedge Y_{1,6,r}) \wedge (Y_{0,7,+} \wedge Y_{1,7,+}) \wedge (Y_{0,8,-} \wedge Y_{1,8,-}) \wedge (Y_{0,9,t} \wedge Y_{1,9,t}) \wedge (Y_{0,10,M} \wedge Y_{1,10,M})$
$\wedge (Y_{0,11,B} \wedge Y_{1,11,B}) \wedge \ldots\} \wedge$

$\{(\mathrm{Y}_{1,0,n} \wedge \mathrm{Y}_{1,1,\mathrm{q}_0} \wedge \mathrm{Y}_{1,2,?} \wedge \mathrm{Y}_{2,0,n} \wedge \mathrm{Y}_{2,1,?} \wedge \mathrm{Y}_{2,2,\mathrm{q}_0}) \wedge (Y_{1,3,p} \wedge Y_{2,3,p}) \wedge (Y_{1,4,+} \wedge Y_{2,4,+}) \wedge$
$(Y_{1,5,d} \wedge Y_{2,5,d}) \wedge (Y_{1,6,r} \wedge Y_{2,6,r}) \wedge (Y_{1,7,+} \wedge Y_{2,7,+}) \wedge (Y_{1,8,-} \wedge Y_{2,8,-}) \wedge (Y_{1,9,t} \wedge Y_{2,9,t}) \wedge (Y_{1,10,M} \wedge Y_{2,10,M})$
$\wedge (Y_{1,11,B} \wedge Y_{2,11,B}) \wedge \ldots\} \wedge$

$\{(Y_{2,0,n} \wedge Y_{3,0,n}) \wedge (\mathrm{Y}_{2,1,?} \wedge \mathrm{Y}_{2,2,\mathrm{q}_0} \wedge \mathrm{Y}_{2,3,\mathrm{p}} \wedge \mathrm{Y}_{3,1,?} \wedge \mathrm{Y}_{3,2,p} \wedge \mathrm{Y}_{3,3,\mathrm{q}_0}) \wedge (Y_{2,4,+} \wedge Y_{3,4,+}) \wedge$
$(Y_{2,5,d} \wedge Y_{3,5,d}) \wedge (Y_{2,6,r} \wedge Y_{3,6,r}) \wedge (Y_{2,7,+} \wedge Y_{3,7,+}) \wedge (Y_{2,8,-} \wedge Y_{3,8,-}) \wedge (Y_{2,9,t} \wedge Y_{3,9,t}) \wedge (Y_{2,10,M} \wedge Y_{3,10,M})$

$\wedge(Y_{2,11,B} \wedge Y_{3,11,B}) \wedge \ldots\} \wedge$

$\{(Y_{3,0,n} \wedge Y_{4,0,n}) \wedge (Y_{3,1,?} \wedge Y_{4,1,?}) \wedge (\mathrm{Y}_{3,2,p} \wedge \mathrm{Y}_{3,3,\mathrm{q}_0} \wedge \mathrm{Y}_{3,4,+} \wedge \mathrm{Y}_{4,2,p} \wedge \mathrm{Y}_{4,3,+} \wedge \mathrm{Y}_{4,4,\mathrm{q}_0}) \wedge$

$(Y_{3,5,d} \wedge Y_{4,5,d}) \wedge (Y_{3,6,r} \wedge Y_{4,6,r}) \wedge (Y_{3,7,+} \wedge Y_{4,7,+}) \wedge (Y_{3,8,-} \wedge Y_{4,8,-}) \wedge (Y_{3,9,t} \wedge Y_{4,9,t}) \wedge (Y_{3,10,M} \wedge Y_{4,10,M})$

$\wedge(Y_{3,11,B} \wedge Y_{4,11,B}) \wedge \ldots\} \wedge$

$\ldots \wedge$

$\{(Y_{82,0,n} \wedge Y_{83,0,n}) \wedge (Y_{82,1,?} \wedge Y_{83,1,?}) \wedge (Y_{82,2,p} \wedge Y_{83,2,p}) \wedge (Y_{82,3,+} \wedge Y_{83,3,+}) \wedge$

$(Y_{82,4,d} \wedge Y_{83,4,d}) \wedge (Y_{82,5,r} \wedge Y_{83,5,r}) \wedge (Y_{82,6,+} \wedge Y_{83,6,+}) \wedge (Y_{82,7,-} \wedge Y_{83,7,-}) \wedge$

$(\mathrm{Y}_{82,8,\mathrm{t}} \wedge \mathrm{Y}_{82,9,\mathrm{q}_8} \wedge \mathrm{Y}_{82,10,\mathrm{X}} \wedge \mathrm{Y}_{83,8,\mathrm{t}} \wedge \mathrm{Y}_{83,9,\mathrm{q}_9} \wedge \mathrm{Y}_{83,10,\mathrm{H}}) \wedge (Y_{82,11,B} \wedge Y_{83,11,B}) \wedge \ldots\}] \rightarrow \boxed{5}$

Now, Let’s combine the components ($\boxed{2}$, $\boxed{3}$, $\boxed{5}$ *and* $\boxed{4}$) together:

$\mathrm{E}_{\mathrm{T,w}} = \mathrm{U} \wedge \mathrm{S} \wedge \mathrm{N} \wedge \mathrm{F}$

$= 1 \wedge$

$(\mathrm{Y}_{0,0,\mathrm{q}0} \wedge \mathrm{Y}_{0,1,n} \wedge \mathrm{Y}_{0,2,?} \wedge \mathrm{Y}_{0,3,p} \wedge \mathrm{Y}_{0,4,+} \wedge \mathrm{Y}_{0,5,d} \wedge \mathrm{Y}_{0,6,r} \wedge \mathrm{Y}_{0,7,+} \wedge \mathrm{Y}_{0,8,-} \wedge \mathrm{Y}_{0,9,t} \wedge \mathrm{Y}_{0,10,M} \wedge \mathrm{Y}_{0,11,B} \wedge \ldots) \wedge$

$[\{(\mathrm{Y}_{0,0,\mathrm{q}_0} \wedge \mathrm{Y}_{0,1,n} \wedge \mathrm{Y}_{1,0,n} \wedge \mathrm{Y}_{1,1,\mathrm{q}_0}) \wedge (Y_{0,2,?} \wedge Y_{1,2,?}) \wedge (Y_{0,3,p} \wedge Y_{1,3,p}) \wedge (Y_{0,4,+} \wedge Y_{1,4,+}) \wedge$

$(Y_{0,5,d} \wedge Y_{1,5,d}) \wedge (Y_{0,6,r} \wedge Y_{1,6,r}) \wedge (Y_{0,7,+} \wedge Y_{1,7,+}) \wedge (Y_{0,8,-} \wedge Y_{1,8,-}) \wedge (Y_{0,9,t} \wedge Y_{1,9,t}) \wedge (Y_{0,10,M} \wedge Y_{1,10,M})$

$\wedge(Y_{0,11,B} \wedge Y_{1,11,B}) \wedge \ldots\} \wedge$

$\{(\mathrm{Y}_{1,0,n} \wedge \mathrm{Y}_{1,1,\mathrm{q}_0} \wedge \mathrm{Y}_{1,2,?} \wedge \mathrm{Y}_{2,0,n} \wedge \mathrm{Y}_{2,1,?} \wedge \mathrm{Y}_{2,2,\mathrm{q}_0}) \wedge (Y_{1,3,p} \wedge Y_{2,3,p}) \wedge (Y_{1,4,+} \wedge Y_{2,4,+}) \wedge$

$(Y_{1,5,d} \wedge Y_{2,5,d}) \wedge (Y_{1,6,r} \wedge Y_{2,6,r}) \wedge (Y_{1,7,+} \wedge Y_{2,7,+}) \wedge (Y_{1,8,-} \wedge Y_{2,8,-}) \wedge (Y_{1,9,t} \wedge Y_{2,9,t}) \wedge (Y_{1,10,M} \wedge Y_{2,10,M})$

$\wedge(Y_{1,11,B} \wedge Y_{2,11,B}) \wedge \ldots\} \wedge$

$\{(Y_{2,0,n} \wedge Y_{3,0,n}) \wedge (\mathrm{Y}_{2,1,?} \wedge \mathrm{Y}_{2,2,\mathrm{q}_0} \wedge \mathrm{Y}_{2,3,\mathrm{p}} \wedge \mathrm{Y}_{3,1,?} \wedge \mathrm{Y}_{3,2,p} \wedge \mathrm{Y}_{3,3,\mathrm{q}_0}) \wedge (Y_{2,4,+} \wedge Y_{3,4,+}) \wedge$

$(Y_{2,5,d} \wedge Y_{3,5,d}) \wedge (Y_{2,6,r} \wedge Y_{3,6,r}) \wedge (Y_{2,7,+} \wedge Y_{3,7,+}) \wedge (Y_{2,8,-} \wedge Y_{3,8,-}) \wedge (Y_{2,9,t} \wedge Y_{3,9,t}) \wedge (Y_{2,10,M} \wedge Y_{3,10,M})$

$\wedge(Y_{2,11,B} \wedge Y_{3,11,B}) \wedge \ldots\} \wedge$

$\{(Y_{3,0,n} \wedge Y_{4,0,n}) \wedge (Y_{3,1,?} \wedge Y_{4,1,?}) \wedge (\mathrm{Y}_{3,2,p} \wedge \mathrm{Y}_{3,3,\mathrm{q}_0} \wedge \mathrm{Y}_{3,4,+} \wedge \mathrm{Y}_{4,2,p} \wedge \mathrm{Y}_{4,3,+} \wedge \mathrm{Y}_{4,4,\mathrm{q}_0}) \wedge$

$(Y_{3,5,d} \wedge Y_{4,5,d}) \wedge (Y_{3,6,r} \wedge Y_{4,6,r}) \wedge (Y_{3,7,+} \wedge Y_{4,7,+}) \wedge (Y_{3,8,-} \wedge Y_{4,8,-}) \wedge (Y_{3,9,t} \wedge Y_{4,9,t}) \wedge (Y_{3,10,M} \wedge Y_{4,10,M})$

$\wedge\left(Y_{3,11,B} \wedge Y_{4,11,B}\right) \wedge \ldots\} \wedge$

$\ldots \wedge$

$\{\left(Y_{82,0,n} \wedge Y_{83,0,n}\right) \wedge \left(Y_{82,1,?} \wedge Y_{83,1,?}\right) \wedge \left(Y_{82,2,p} \wedge Y_{83,2,p}\right) \wedge \left(Y_{82,3,+} \wedge Y_{83,3,+}\right) \wedge$
$\left(Y_{82,4,d} \wedge Y_{83,4,d}\right) \wedge \left(Y_{82,5,r} \wedge Y_{83,5,r}\right) \wedge \left(Y_{82,6,+} \wedge Y_{83,6,+}\right) \wedge \left(Y_{82,7,-} \wedge Y_{83,7,-}\right) \wedge$
$\left(\mathrm{Y}_{82,8,\mathrm{t}} \wedge \mathrm{Y}_{82,9,\mathrm{q}_8} \wedge \mathrm{Y}_{82,10,\mathrm{X}} \wedge \mathrm{Y}_{83,8,\mathrm{t}} \wedge \mathrm{Y}_{83,9,\mathrm{q}_9} \wedge \mathrm{Y}_{83,10,\mathrm{H}} \right) \wedge \left(Y_{82,11,B} \wedge Y_{83,11,B}\right) \wedge \ldots\}] \wedge$
$(\mathrm{Y}_{83,9,\mathrm{q}9})$

Finally;
$\mathrm{E}_{\mathrm{T,w}} = \left(\bigwedge_{0 \le i \le 83, 0 \le j \le 10, \forall \mathrm{A} \in \mathrm{Q} \cup \Gamma \setminus \{\mathrm{B}\}} Y_{i,j,A}\right) \wedge \left(\bigwedge_{0 \le i \le 83, 11 \le j \le 83, B \in \Gamma} Y_{i,j,B}\right)$.

6.11.5 Results of Formal Verification

After simplifying the expression $\mathrm{E}_{\mathrm{T,w}}$, it has been visible that the variables in the expression has been operated only by the $\wedge$ operator. Further, for this Boolean expression, 2^{830} ($\cong$6.6681E+240) truth assignment possibilities has been found (considering the symbols other than blank B). If it has been drawn the truth table for this expression, it has got the true value (1), only when all the variables are assigned the value true (1). As the result, the value 1 has been obtained with approximately $1/2^{830}$ probability.
Hence, it could conclude that this problem is a satisfiable problem and according to the Cook's Theorem, this Satisfiability Problem is NP-Complete [137].

6.11.6 Time Complexity

Let's analyze the complexity of the derivation to show that it is performed in polynomial time. First, it is necessary to examine the size of $\mathrm{E}_{\mathrm{T,w}}$ and its variables. It has 83^2 (p^2 (n)) = 6889 cells, and each cell contains at least and exactly one symbol from 30 (=10+20) symbols ($\in Q \cup \Gamma$). As these symbols depends only on the Turing machine, not on the length of the input, n, total number of variables is $O(p^2(n))$. Then, considering the sizes of each part U, S, N, and F of $\mathrm{E}_{\mathrm{T,w}}$, the total size of the Boolean expression can be expressed $O(p^2(n))$ and it is polynomial.

6.12 Summary

From the entire thesis, this chapter has a more weight. This has first introduced the experimental design and the results of the SSPM-FC, which has been discussed under four testing scenarios. Similarly, SSPM-Sorting was tested. Further, it has proved that the system gains improvements through subsequent execution cycles for both the cases, due to the integrated proposed continuous processing model. Moreover, SSPM sorting was compared with some other sorting techniques. Next, a formal verification has been done for the model. There, it has discussed why T was designed as a Turing Machine. Then, the NTM, the transition table, and the configurations of NTM, which demonstrates the moves of the machine have been presented. Afterwards, it has proved the satisfiability of the NTM., further proving the real-world applicability of the model in the theoretical level.

The coming chapter is the last chapter of the thesis, which concludes the work.

CHAPTER 07
CONCLUSION AND FUTURE WORK

7.1 Introduction

Chapter 6 reported the evaluation of the model conducted in two levels. First, it has evaluated the model empirically proving its ability to evolve over generations of program execution cycles. Second, it has formally verified showing its appropriateness to the real-world using a Turin Machine. This chapter has been concluded the work done for a long period of time in modelling memory as conditional phenomena for a new theory of computing. Further, this chapter has given an overall conclusion for the work interpreting the results obtained in the previous chapter, while explaining how each and every objective has been achieved. Finally, this has discussed the encountered limitations and some further works.

7.2 Modelling Memory as Conditional Phenomena for a New Theory of Computing

In particular, the aim of this research work was to model computer memory as conditional phenomena that enhance the efficiency of continuous processing over consecutive program execution cycles. Further, several real-world scenarios that have displayed the continuous processing and evolving nature of the human mind, have rooted the research idea on the new approach in improving the computing power. Then, critically analyzing the existing processing models, the new model was introduced, extending the features of existing models by utilizing the concepts from BTM. As briefed down below by achieving every objective one by one, a new theory of computing was introduced.

7.2.1 Critical study about various models for computing

The Computers have wide-spread usage in the world. Therefore, as mentioned in the chapter one and chapter two, finding new hardware models and software models for computing to improve processing power of the computer have been a constant research challenge. In this continuous effort, the hardware level researchers have been successful in accomplishing their duty by introducing different processors, memory models, data access mechanisms thereby enhancing the processing power in hardware level. Specifically, the modifications that have been applied to the VNA in both the hardware and software level were based on the separation of memory from processing in VNA. In addition to that, the software level developments have been inadequate yet to utilize under laid hardware improvements. Some of these software models have been so specific and have targeted to solve particular real-world problems, while suffering from their own drawbacks such as need more processing or more memory or more resources. Therefore, introducing a new computing model in software level with a new approach to enhance the computing power was a significant challenge. In this background, it was difficult to find a model to improve efficiency of computing with the processing similar to the processing in human mind based on BTM, where the memory is a result of continuous processing and improve the efficiency over the time.

7.2.2 In depth study about the Buddhist Theory of Mind

Chapter 3 has described the BTM, which has set the theoretical foundation for the proposed model with the other inspirations. BTM has explained everything has been arising as per the conditions. Further, BTM has a set of concepts called 24 CRs to explain the relationship between the cause and effect. Stated in another way, BTM has provided theories behind the formation of the continuity, strong establishment or improvement gain through continuous practice, and behind theories of such other formations in the human mind process.

7.2.3 Propose a new computing model where the memory is a result of continuous processing

While Chapter 5 has described how the BTM has exploited in modelling the proposed model, the Chapter 4 presents the proposed model, the Six-state Continuous Processing Model. This model consists of three features, such as two processes (internal and external), continuous processing, and conditionally evolving memory. The processing states of the proposed processing model are new, ready, running, blocked, sleep and terminate have been compared with the states of existing processing models as seen in the Table 7.1.

Table 7.1: Comparison with Existing Processing Models

Proposed Model (Six-State)	**Seven-State Model**	**Linux Six-State Model**	**Five-State Model**	**Three-State Model**	**Two-State Model**
New	New		New		
Ready	Ready	Ready	Ready	Ready	Not-Running
	Ready/ Suspend				
Blocked	Blocked	Interruptible	Blocked	Blocked	Not-Running
		Un-interruptible			
		Stopped			
	Blocked/ Suspend				
Running	Running	Executing	Running	Running	Running
Sleep	Exit	Zombie	Exit		
Terminate					

Although, the terminology used for the states of the newly proposed SSPM are currently using in the existing processing models, as explained in Chapter 4, the transitions of SSPM are more functional than the existing transitions. Specially, the transitions New-Ready, Ready-Ready, Ready-Terminate, Running-Ready, Sleep-Ready, Blocked-Ready, and Running-Sleep are comprehensively defined by exploiting a set of CRs explained in BTM. A transition-wise comparison of the proposed model with the existing OS processing models can be seen in the Table 7.2.

Table 7.2: Transition-wise comparison of the proposed model with existing OS processing models

Proposed Processing Model (Six-State)	**Five-State Model**	**Linux Processing Model**	**Solaris Thread Model**	**Windows Thread Model**
Null → New	Null → New			
New → Ready	New → Ready			
Ready → Ready				
Ready → Running	Ready → Running	Ready → Executing	Run → OnProc	Ready → Standby Standby → Running
Running → Ready	Running → Ready	Executing → Ready	OnProc →Run	Running → Ready
Running → Blocked	Running → Blocked	Running→Stopped Running→ Uninterruptible Running→ Interruptible	OnProc →Sleep OnProc →Stop	Running → Waiting Waiting→ Transition
Blocked → Ready	Blocked → Ready	Stopped→ Ready Uninterruptible→ Ready interruptible→ Ready	Sleep →Run Stop →Run	waiting→ Ready Transition→ Ready
Running → Sleep	Running → Exit	Executing→ Zombie	OnProc→ Zombie Zombie→Free	Running → Terminated
Sleep→ Ready				
Ready → Terminate				

During the subsequent system execution cycles, the new model SSPM examines the inputs and the relevant instructions, and refines the system through a continuous process considering the causal relations. In fact, ‘Repetition’, ‘Proximity’, ‘Karma’, ‘Support’, and ‘Presence’ were the major concepts used in the modelling process. Eventually, the

SSPM completely rely on the fundamental principal concepts that was described in Chapter 3. Therefore, the SSPM is not solely based on the concepts that have been applied in current computing models such as the decentralization, the survival of the fittest, and incrementalism. However, more efficient hybrid models can be introduced combining the SSPM with existing technologies.

7.2.4 Customizing Programs with SSPM

As stated in Chapter 4, the proposed model customized a Fraction Calculator (SSPM-FC), Quadratic Equation Solver (SSPM-QES), Sorting Program (SSPM-Sorting), and a Simulated Process Scheduler (SSPM-PS). Among these, the SSPM-FC covers all the features of the proposed model. As the proposed model SSPM has internal process, continuous processing, and an evolving memory, the systems customized by SSPM can gain self-improvements over program execution cycles through the actions stated in the Chapter 4, when the external inputs or external processes are absent or vice-versa. Such a way, the SSPM-FC can gain improvements. In addition to that, if the incoming expression was longer (for example, $\frac{7}{10} + 8\frac{1}{7} + \frac{12}{15} * \left(\frac{1}{2} + \frac{5}{9} - \frac{1}{12}\right) * \frac{11}{20} - \frac{2}{3} + 5\frac{6}{8} + \frac{4}{5} - \frac{7}{13} + \frac{3}{19} + \frac{6}{17} + \frac{3}{4}$) in the SSPM-FC, then even at the end of the computation of such longer expressions, the performance of SSPM-FC could be enhanced. Moreover, in all above mentioned customized programs, the time taken by the expression evaluation can be improved due to the characteristics of SSPM that can identify and classify the inputs according to the input patterns and the operations, and do computations accordingly.

7.2.5 Evaluate the proposed model

The paired-t test was applied with 99% confidence level under the Testing Scenario 1 and 99% confidence interval under Testing Scenario 2. It has been able to prove that by customizing the FC with the SSPM, it could improve the performance of an SSPM-FC, when it executes over program execution cycles. Further, it could prove that the SSPM-QES could gain improvement over subsequent execution cycles by applying paired-t test with 95% confidence interval as similar to the SSPM-FC.

The program SSPM-Sorting was tested under five testing scenarios. Under the first (SSPM-S-Insertion) and the second (SSPM-S-Equal), it could prove that the system can gain improvement over consecutive program executions, by applying paired-t test with 95% confidence interval as similar to the above two cases. Though, the SSPM-S-Equal shows improvement for any total number of elements in the set, the performance of the SSPM-S-Insertion depends on the number of new elements and the total number of elements in the set with compared to the previous set. Next three testing scenarios were allocated to compare the model with the parallel computing, incremental computing. Self-adjusting computing, dynamically tuned library for sorting and the evolutionary computing. The respective results are summarized in each section. Through the smaller tactics memory and the continuous processing, any of the computing technique in SSPM-Sorting could be applied appropriately as per the arising conditions such as size of the set or standard deviation. Hence, the performance of the SSPM-Sorting could be enhanced consequently.

A formal verification has been provided by simulating the proposed model in TM. For the single-tape NTM T which runs in a polynomial time $p(n) = 83$ with the input w, this Boolean expression is derived. This Boolean expression is a function of Both T and w. There, S (Starts Right) is the only part which depends on the content of the input w. The other parts U, N, and F depend on T and n, the length of the input. The Model is NP complete, and time complexity has been calculated as $O(p^2(n))$ [137].

Then, the simple scenario demonstrated by SSPM-PS also showed an improvement in its processing when executing the system over generations of system executions.

Overall, it could conclude that by customizing applications with SSPM, the performance could be enhanced thru the smaller tactics memory and the continuous processing over consecutive program execution cycles.

7.3 Limitation

First, it should be mentioned that the data recording process has been affected by environmental physical conditions. The condition has been severe as it has been collecting time values in nanoseconds. The selection of the number of inputs was also affected as it was required to minimize variation of utilization of underlined physical resources. The user interface of the SSPM-FC, has initially consisted of test areas to mention input, output, process state and process nature. Then, the time variation was minimum. However, it has been modified to display the tactics smaller tactics memory, the particular changes in the memory after modifications, and recorded time values after each computation (The interface was modified for the illustration purpose). Then, a considerable increase has been appeared in the time values recorded after the modification due to the added additional code lines dedicated to system outputs. It could overcome by using the SSPM-FC in its initial form.

7.4 Further Work

It is possible to adopt the proposed computing model in different ways. Some of the ideas for the adoption are stated below. It is possible to customize the kernel of an OS using the SSPM. For example, the SSPM can improve the OS to apply appropriate scheduling algorithms through recognizing the patterns of iterative and incoming processes [140]. In addition to that, this can customize an expert system [128]. Specifically, the inference engine can be customized. With SSPM, it was expected to build a computer that becomes more efficient in subsequent execution cycles similar to the mind model. With that expectation, an expert system can be developed to solve more complex problems, as the human experts accurately solve problems by using the proper set of steps, inferences, rules/logic and facts in a specific domain with a minimum time duration. There are expert systems developed for different domains such as medicine, business and engineering that are belong to different technical categories such as object-oriented, frame-based, rule-based and induction-based with multiple advantages. For this work, it can be considered rule-based expert systems that consist of a user interface, explanation facility, facts used by the rules, an inference engine, prioritizing facility, an agenda, knowledge acquisition facility and conflict resolution facility. It has been a problem to build expert systems with the ability in generalizing concepts, identifying the CRs among knowledge entities and gaining improvements over generations of execution as human experts do. Another possible application of SSPM would be adjusting the difficulty of levels of a game during the game development scenario. As this process requires extensive refinement cycles, the SSPM can contribute to enhance the processing speed of difficulty adjustment process over subsequent adjustment cycles.

The SSPM can customize a compiler or an interpreter in a program development environment. Then, the program developer can improve the software in consecutive program development cycles as experience in the FC (This could be more applicable while doing the testing). At the end, the software company can deliver an efficient software to their client. (Note: By using this SSPM, within a program, the computing power of the program can be enhanced over the time. The FC contains of both the expression evaluator and the compiler, which compile the incoming expressions. Both of these aspects were affected by the SSPM).

For the continuously processing computing systems, SSPM has been proved to be an efficiency enhancer. In fact, the military or aviation based controlling systems can be drawn as an example for the systems with continuous processing. Further, learning or classification in some other soft computing mechanisms also can be with the SSPM. Moreover, in improving the algorithm of a program, transformation techniques could be applied at the program tree level. In addition to these, it is expected the SSPM to display improvements in the quality and the accuracy more, over program execution cycles as similar to the human mind.

Overall, it can be concluded that the new processing model, the SSPM introduces a novel approach for the computing.

7.5 Summary

This has concluded the entire work that have been introduced a new computing model as stated in the thesis further describing the proposed model and mentioning how the objectives have been achieved. Finally, it has discussed about the limitations and further works of this.

References

[1] W. Stallings, *Computer organization and architecture: designing for performance*. Upper Saddle Rive, N.J.: Pearson Prentice Hall, 2006.

[2] J. Fuegi and Jo Francis, “Lovelace & Babbage and the Creation of the 1843 ‘Notes,’” *IEEE Ann. Hist. Comput.*, vol. 25, no. 4, pp. 16–26, Dec. 2003, doi: 10.1109/MAHC.2003.1253887.

[3] D. A. Patterson and J. L. Hennessy, *Computer organization and design*, 5th ed. oxford: Morgan Kaufmann Publishers, 2014.

[4] R. Das, “Blurring the Lines between Memory and Computation,” *IEEE Micro*, vol. 37, no. 6, pp. 13–15, Nov. 2017, doi: 10.1109/MM.2017.4241340.

[5] S. A. McKee and others, “Reflections on the memory wall.,” in *Conf. Computing Frontiers*, 2004, p. 162.

[6] P. Machanick, “Approaches to addressing the memory wall,” *Sch. IT Electr. Eng. Univ. Qld.*, 2002.

[7] D. A. Patterson and J. L. Hennessy, *Computer organization and design*, RISC-V. Morgan Kaufmann Publishers, 2018.

[8] A. Jog *et al.*, “Anatomy of GPU Memory System for Multi-Application Execution,” in *Proceedings of the 2015 International Symposium on Memory Systems*, New York, NY, USA, 2015, pp. 223–234, doi: 10.1145/2818950.2818979.

[9] K. Wang, X. Ding, R. Lee, S. Kato, and X. Zhang, “GDM: device memory management for gpgpu computing,” *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 42, no. 1, pp. 533–545, 2014.

[10] G. W. Burr *et al.*, “Phase change memory technology,” *J. Vac. Sci. Technol. B Nanotechnol. Microelectron. Mater. Process. Meas. Phenom.*, vol. 28, no. 2, pp. 223–262, 2010.

[11] C. Neumann, S. S. Qin, and X. Zheng, “Stanford Memory Trends,” *Stanford Nanoelectronics Lab*, 2018. https://nano.stanford.edu/stanford-memory-trends (accessed Mar. 29, 2019).

[12] S. Ambareesh and J. Fathima, “A Cached Middleware to Improve I/O Performance Using POSIX Interface,” *Procedia Comput. Sci.*, vol. 85, pp. 125–132, 2016, doi: 10.1016/j.procs.2016.05.197.

[13] M. Abd-El-Barr and H. El-Rewini, *Fundamentals of computer organization and architecture*. Hoboken, N.J: Wiley, 2005.

[14] G. E. Moore, “Moore’s Law at 40. Chapter 7,” in *Understanding Moore’s Law: Four decades of innovation edited by D. C. Brock*, Philadelphia, PA.: Chemical Heritage Foundation, 2006, pp. 67–84.

[15] A. S. Karunananda, P. R. Goldin, G. Rzevski, S. Fernando, and H. R. Fernando, “ON COMPUTING THE BEHAVIOR OF THE MIND FROM AN EASTERN PHILOSOPHICAL PERSPECTIVE,” *Int. J. Des. Nat. Ecodynamics*, vol. 10, no. 3, pp. 224–232, 2015, doi: 10.2495/DNE-V10-N3-224-232.

[16] W. A. C. Weerakoon, A. S. Karunananda, and N. G. J. Dias, “Six-state Continuous Processing Model for a New Theory of Computing,” University of Moratuwa, 2019, vol. 890, pp. 32–48, doi: https://doi.org/10.1007/978-981-13-9129-3_3.

[17] W. A. C. Weerakoon, A. S. Karunananda, and N. G. J. Dias, “A tactics memory for a new theory of computing,” Apr. 2013, pp. 153–158, doi: 10.1109/ICCSE.2013.6553901.

[18] W. A. C. Weerakoon, A. S. Karunananda, and N. G. J. Dias, “Conditionally evolving memory for computers,” Aug. 2015, pp. 271–271, doi: 10.1109/ICTER.2015.7377704.

[19] A. S. Karunananda, “Computer modelling of thought processes,” *Int. J. Comput. Appl. Technol.*, vol. 6, no. 2/3, pp. 135–140, 1993.

[20] Karunananda A. S., “Using an eastern philosophy for providing a theoretical basis for some Heuristics used in artificial neural networks,” *Malayasian J. Comput. Sci.*, vol. 15, no. 2, pp. 28–33, Dec. 2002.

[21] L. P. Ranatunga, “ON COMPUTING MENTAL STATES,” *Malays. J. Comput. Sci.*, vol. 9, no. 2, pp. 43–53, 1996.

[22] W. Stallings, *Operating systems: internals and design principles*, 7th ed. Boston: Prentice Hall, 2012.

[23] A. S. Tanenbaum and H. Boss, *Modern operating systems*, 4th ed. Upper Saddle River, N.J: Pearson/Prentice Hall, 2015.

[24] H. Kasim, V. March, R. Zhang, and S. See, “Survey on parallel programming model,” in *InIFIP International Conference on Network and Parallel Computing*, Oct. 2008, pp. 266–275.

[25] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. 2015.

[26] M. J. Wooldridge, *An introduction to multiagent systems*, 2nd ed. Chichester, U.K: John Wiley & Sons, 2009.

[27] Black Paul E., D. Richard Kuhn, and Carl J. Williams, “Quantum computing and communication,” *Adv. Comput.*, vol. 56, pp. 189–244, 2002.

[28] Eiben A.E. and Smith J.E., *Introduction to Evolutionary Computing*, Second. Springer, 2015.

[29] M. A. Hammer, K. Y. Phang, M. Hicks, and J. S. Foster, “Adapton: Composable, demand-driven incremental computation,” 2014.

[30] I. J. Fister, X. S. Yand, I. Fister, J. Brest, and D. Fister, “A Brief Review of Nature-Inspired Algorithms for Optimization,” *Electrotech. Rev.*, vol. 80, no. 3, 2013.

[31] K. F. Man, K. S. Tang, and S. Kwong, “Genetic algorithms: concepts and applications [in engineering design],” *IEEE Trans. Ind. Electron.*, vol. 43, no. 5, pp. 519–534, Oct. 1996, doi: 10.1109/41.538609.

[32] P. J. Bentley, T. G. W. Gordon, J. Kim, and S. Kumar, “New trends in evolutionary computation,” 2001, vol. 1, pp. 162–169, doi: 10.1109/CEC.2001.934385.

[33] A. K. Kar, “Bio inspired computing – A review of algorithms and scope of applications,” *Expert Syst. Appl.*, vol. 59, pp. 20–32, 2016, doi: https://doi.org/10.1016/j.eswa.2016.04.018.

[34] N. Siddique and H. Adeli, “Nature Inspired Computing: An Overview and Some Future Directions,” *Cogn. Comput.*, vol. 7, no. 6, pp. 706–714, Dec. 2015, doi: DOI 10.1007/s12559-015-9370-8.

[35] A. S. Karunananda, *Artificial intelligence*. Tharanji Prints, 2004.

[36] J. Liu and K. C. Tsui, “Toward nature-inspired computing,” *Commun. ACM*, vol. 49, no. 10, pp. 59–64, 2006.

[37] S. Costantini and G. De Gasperis, “Memory, experience and adaptation in logical agents,” in *Management Intelligent Systems*, Springer, 2013, pp. 17–24.

[38] B. Bakker, “Reinforcement learning with long short-term memory,” in *Advances in neural information processing systems*, 2002, pp. 1475–1482.

[39] F. C. Bartlett, F. C. Bartlett, and W. Kintsch, *Remembering: A study in experimental and social psychology*, vol. 14. Cambridge University Press, 1995.

[40] D. L. Schacter, “Constructive memory: past and future,” *Dialogues Clin. Neurosci.*, vol. 14, no. 1, p. 7, 2012.

[41] Kenneth J. Malmberg, Jeroen G. W. Raaijmakers, and Richard M. Shiffrin, “50 years of research sparked by Atkinson and Shiffrin (1968),” *Mem. Cognit.*, vol. 47, no. 4, pp. 561–574, 2019.

[42] A. Baddeley, “Working memory,” *Curr. Biol.*, vol. 20, no. 4, pp. R136–R140, 2010, doi: https://doi.org/10.1016/j.cub.2009.12.014.

[43] K. A. De Jong, *Evolutionary Computaion: A Unified Approach*. London, England: The MIT Press, Cambridge, Massachusetts, 2006.

[44] M. Sharma, P. Sindhwani, and V. Maheshwari, “Genetic Algorithm Optimal approach for Scheduling Processes in Operating System,” *Int. J. Comput. Sci. Netw. Secur.*, vol. 14, no. 5, pp. 91–94, 2014.

[45] Sindhwani P. and Wadhwa V., “Genetic algorithm Approach for Optimal CPU Scheduling,” *IJCST*, vol. 2, no. 2, pp. 92–95, Jun. 2011.

[46] M. U. Siregar, “A New Approach to CPU Scheduling: Genetic Round Robin,” *Int. J. Comput. Appl.*, vol. 47, no. 19, pp. 18–25, Jun. 2012.

[47] M. Carlsson, “Monads for incremental computing,” in *The seventh ACM SIGPLAN international conference on Functional programming*, 2002, pp. 26–35.

[48] Matthew A. Hammer and Umut A. Acar, “Memory Management for Self-Adjusting Computation,” presented at the Proceedings of the the 2008 International Symposium on Memory Management, Tucson, Arizona, USA, 2008.

[49] U. A. Acar, G. E. Blelloch, and R. Harper, *Selective memoization*, vol. 38. ACM, 2003.

[50] U. A. Acar, G. E. Blelloch, M. Blume, R. Happer, and K. Tangwongsan, “An experimental Analysis of Self-Adjusting Computation,” *ACM Trans. Program. Lang. Syst.*, vol. 32, no. 1, 2009.

[51] Xiaoming Li, Maria Jesus Grzaran, and David Padua, “A dynamically Tuned Sorting Lirary,” Los Alamitos, Calif., 2004.

[52] Y. A. Liu, “Efficient computation via incremental computation,” in *Pacific-Asia Conf. on Knowledge Discovery and Data Mining*, 1998, vol. 1574, pp. 194–203.

[53] W. F. Jayasuriya, *The Psychology & Philosophy of Buddhism*. Onalaska, USA: pariyatti, 2016.

[54] R. C. N. Thero, *Pattana Maha Pakarana Sannaya*, 6th ed. Pokunuwita, Sri Lanka: Sri Chandravimala Darma P, 2014.

[55] B. Bodhi, *Comprehensive Manual of Abhidhamma: The Psychology of Buddhism (Abhidhammattha Sangaha)*. Buddhist Publication Society, 2006.

[56] M. Sipser, *Introduction to the Theory of Computation*, Second. Thomson Course Technology, 2007.

[57] Wong, H. S. P. and Salahuddin, S., "Memory leads the way to better computing," *Nat. Nanotechnol.*, vol. 10, no. 3, pp. 191–194, Mar. 2015.

[58] S. Jain, A. Ranjan, K. Roy, and A. Raghunathan, "Computing in memory with spin-transfer torque magnetic ram," *IEEE Trans. Very Large Scale Integr. VLSI Syst.*, vol. 26, no. 3, pp. 470–483, 2017.

[59] A. J. Smith, "Cache Memories," *ACM Comput. Surv.*, vol. 14, no. 3, pp. 473–530, Sep. 1982.

[60] W. J. Starke *et al.*, "The cache and memory subsystems of the IBM POWER8 processor," *IBM J. Res. Dev.*, vol. 59, no. 1, p. 3:1-3:13, Jan. 2015, doi: 10.1147/JRD.2014.2376131.

[61] LARRY HARDESTY, "Computer Chip Cores Communicate by Networks Instead of Bus," *SciTechDaily*, 2012. https://scitechdaily.com/computer-chip-cores-communicate-by-networks-instead-of-bus/.

[62] Kevin Lee, "Best RAM 2019: the top memory for your PC," *Techrader: Source for Tech buying Advices*, Oct. 04, 2019. https://www.techradar.com/news/best-ram.

[63] Scharon Harding, "What Is CAS Latency in RAM? CL Timings Explained," *Tom's Hardware*, Sep. 03, 2019. https://www.tomshardware.com/reviews/cas-latency-ram-cl-timings-glossary-definition,6011.html (accessed Oct. 26, 2019).

[64] S. Ghose, K. Hsieh, A. Boroumand, R. Ausavarungnirun, and O. Mutlu, "Enabling the adoption of processing-in-memory: Challenges, mechanisms, future research directions," *ArXiv Prepr. ArXiv180200320*, 2018.

[65] H. Asghari-Moghaddam, Y. H. Son, J. H. Ahn, and N. S. Kim, "Chameleon: Versatile and practical near-DRAM acceleration architecture for large memory systems," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Taipei, Taiwan, Oct. 2016, pp. 1–13, doi: 10.1109/MICRO.2016.7783753.

[66] Mihir Patkar, "Intel Core i9 vs. i7 vs. i5: Which CPU Should You Buy?," *Technology Explained*, Dec. 06, 2018. https://www.makeuseof.com/tag/intel-core-i9-vs-i7-vs-i5-cpu/.

[67] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "GPU computing," 2008.

[68] J. Nickolls and W. J. Dally, "The GPU computing era," *IEEE Micro*, vol. 30, no. 2, pp. 56–69, 2010.

[69] Y. Kim, D. Han, O. Mutlu, and M. Harchol-Balter, "ATLAS: A scalable and high-performance scheduling algorithm for multiple memory controllers," in *HPCA-16*

2010 The Sixteenth International Symposium on High-Performance Computer Architecture, 2010, pp. 1–12.

[70] Z. Guz, E. Bolotin, I. Keidar, A. Kolodny, A. Mendelson, and U. C. Weiser, "Many-core vs. many-thread machines: Stay away from the valley," *IEEE Comput. Archit. Lett.*, vol. 8, no. 1, pp. 25–28, 2009.

[71] F. Gebali, *Algorithms and parallel computing*. Hoboken, N.J: Wiley, 2011.

[72] A. H. Almutairi and A. H. Alruwaili, "Improving of Quicksort Algorithm Performance by Sequential Thread or Parallel Algorithms," *Glob. J. Comput. Sci. Technol. - Hardw. Comput.*, vol. 12, no. 10, 2012.

[73] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating system concepts*, Ninth edition. India: Wiley India Pvt. Ltd., 2016.

[74] Yanhong A. Liu, Scott D. Stoller, and Tim Teitelbaum, "Discovering auxiliary information for incremental computation," in *POPL '96 Proceedings of the 23rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, Florida, USA, Jan. 1996, pp. 157–170.

[75] Liu Y A, "CACHET: an interactive, incremental-attribution-based program transformation system for deriving incremental programs," presented at the Proceedings 1995 10th Knowledge-Based Software Engineering Conference, 1995.

[76] M. A. Hammer *et al.*, "Incremental computation with names," *ACM SIGPLAN Not.*, vol. 50, no. 10, pp. 748–766, Oct. 2015.

[77] Yanhong A. Liu, Scott D. Stoller, and Tim Teitelbaum, "Static caching for incremental computation," *ACM Trans. Program. Lang. Syst. TOPLAS*, vol. 20, no. 3, pp. 546–585, May 1998.

[78] U. A. Acar, G. Blelloch, and R. Harper, "Self-adjusting computation," PhD Thesis, Citeseer, 2005.

[79] U. A. Acar, A. Ahmed, and M. Blume, "Imperative Self-adjusting Computation," *SIGPLAN Not*, vol. 43, no. 1, pp. 309–322, Jan. 2008, doi: 10.1145/1328897.1328476.

[80] P. Bhatotia, A. Wieder, R. Rodrigues, U. A. Acar, and R. Pasquin, "Incoop: MapReduce for incremental computations," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*, 2011, p. 7.

[81] M. A. Hammer, U. A. Acar, and Y. Chen, "CEAL: a C-based language for self-adjusting computation," in *ACM Sigplan Notices*, 2009, vol. 44, pp. 25–37.

[82] P. Bhatotia, P. Fonseca, U. A. Acar, B. B. Brandenburg, and R. Rodrigues, "iThreads: A threading library for parallel incremental computation," in *ACM SIGPLAN Notices*, 2015, vol. 50, pp. 645–659.

[83] K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta, and J. Hennessy, *Memory consistency and event ordering in scalable shared-memory multiprocessors*, vol. 18. ACM, 1990.

[84] A. S. Jensen and J. Villadsen, "A comparison of organization-centered and agent-centered multi-agent systems," *Artif. Intell. Res.*, vol. 2, no. 3, Apr. 2013, doi: 10.5430/air.v2n3p59.

[85] N. R. Jannings, "An agent-based approach for building complex software systems," *Coomunications ACM*, vol. 44, no. 4, pp. 35–41, Jan. 2001.

[86] S. Costantini, "Defining and Maintaining Agent's Experience in Logical Agents," in *Informal Proc. of the LPMAS (Logic Programming for Multi-Agent Systems) Workshop at ICLP 2011, and CORR Proceedings of LANMR 2011, Latin-American Conference on NonMonotonic Reasoning*, Mexico, 2011, pp. 151–165.

[87] Gero J. S., "Understanding behaviors of a constructive memory agent: A markov chain analysis.," *Knowl.-Based Syst.*, vol. 22, no. 8, pp. 610–621, 2009.

[88] P.-S. Liew and J. S. Gero, "Constructive memory for situated design agents," *AI EDAM*, vol. 18, no. 2, pp. 163–198, 2004.

[89] Xiaoming Li, Maria Jesus Grzaran, and David Padua, "Optimizing sorting with genetic algorithms," New York, NY, USA., 2005, doi: 10.1109/CGO.2005.24.

[90] L. Daniel and K. T. Chaturvedi, "Review on Different Evolutionary Computing Techniques In Particle swarm Optimization," *Int. J. Eng. Sci. Math.*, vol. 7, no. 3, pp. 230–236, 2018.

[91] A. Ahilan and P. Deepa, "Improving lifetime of memory devices using evolutionary computing based error correction coding," in *Computational intelligence, cyber security and computational models*, Springer, 2016, pp. 237–245.

[92] Gunther Raidl, "Evolutionary computation: An overview and recent trends," *OGAI*, pp. 2–7, 2005.

[93] P. de Wilde and G. Briscoe, "“Stability of evolving multiagent systems," *" IEEE Trans. Syst. Man Cybern. Part B Cybern.*, vol. 41, no. 4, pp. 1149–1157, 2011.

[94] C. Tacla and J.-P. Barthès, "A multi-agent architecture for evolving memories," 2003.

[95] M. C. Mozer and others, "Neural net architectures for temporal sequence processing," in *Santa Fe Institute Studies in the Sciences of Complexity-Proceedings Volume-*, 1993, vol. 15, pp. 243–243.

[96] M. D. Godfrey and D. F. Hendry, "The computer as von Neumann planned it," *IEEE Ann. Hist. Comput.*, vol. 15, no. 1, pp. 11–21, 1993, doi: 10.1109/85.194088.

[97] L. Wang *et al.*, "Superneurons: dynamic GPU memory management for training deep neural networks," in *ACM SIGPLAN Notices*, 2018, vol. 53, pp. 41–53.

[98] P. Chi *et al.*, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *ACM SIGARCH Computer Architecture News*, 2016, vol. 44, pp. 27–39.

[99] S. Khadka, J. J. Chung, and K. Tumer, "Evolving memory-augmented neural architecture for deep memory problems," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2017, pp. 441–448.

[100] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines," *ArXiv Prepr. ArXiv14105401*, 2014.

[101] J. J. A. Ansel, "Autotuning programs with algorithmic choice," PhD Thesis, Massachusetts Institute of Technology, 2014.

[102] P. Pfaffe, M. Tillmann, S. Walter, and W. F. Tichy, “Online-autotuning in the presence of algorithmic choice,” in *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2017, pp. 1379–1388.

[103] Y. Ding, J. Ansel, K. Veeramachaneni, X. Shen, U.-M. O’Reilly, and S. Amarasinghe, “Autotuning algorithmic choice for input sensitivity,” in *ACM SIGPLAN Notices*, 2015, vol. 50, pp. 379–390.

[104] J. Ansel *et al.*, “Opentuner: An extensible framework for program autotuning,” in *Proceedings of the 23rd international conference on Parallel architectures and compilation*, 2014, pp. 303–316.

[105] J. Ansel, Y. L. Wong, C. Chan, M. Olszewski, A. Edelman, and S. Amarasinghe, “Language and compiler support for auto-tuning variable-accuracy algorithms,” in *International Symposium on Code Generation and Optimization (CGO 2011)*, 2011, pp. 85–96.

[106] J. Ansel *et al.*, “Siblingrivalry: online autotuning through local competitions,” in *Proceedings of the 2012 international conference on Compilers, architectures and synthesis for embedded systems*, 2012, pp. 91–100.

[107] W.-C. Lee *et al.*, “White-box program tuning,” in *Proceedings of the 2019 IEEE/ACM International Symposium on Code Generation and Optimization*, 2019, pp. 122–135.

[108] P. Balaprakash *et al.*, “Autotuning in High-Performance Computing Applications,” *Proc. IEEE*, vol. 106, no. 11, pp. 2068–2083, 2018.

[109] P. Tillet and D. Cox, “Input-aware auto-tuning of compute-bound HPC kernels,” in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017, p. 43.

[110] Chen, G, Church, D. A, Englert, B. G., and Zubairy, M. S., “Mathematical models of contemporary elementary quantum computing devices,” in *Quantum Control: Mathematical and Numerical Challenges*, 2003, vol. 33, pp. 79–117.

[111] Calude C. S and Michel J. D., “Reflections on Quantum Computing,” 2000.

[112] Federman A., “What Buddhism Taught Cognitive Science about Self, Mind and Brain,” *Enrahonar Quad. Filos.*, vol. 47, pp. 39–62, 2011.

[113] Nàrada Mahà Thera, *A Manual of Abhidhamma (Abhidhammattha Saïgaha of Bhadanta Anuruddhàcariya)*. Malaysia: Biddhist Missionary Society, 1987.

[114] Nayanaponika Thero, *Abhidhamma Studies: Researches in Buddhist Psycology*. Kandy, Sri Lanka: Buddhist Publication Society, 1976.

[115] Bhaddanta Dr. Rewata Dhamma, *Process of Consiousness and Matter*. Onalaska, USA: Pariyatti, 2015.

[116] A. Rathnapala, *Abhidharmartha Pradeepika*, vol. 3. Sri Lanka: M. D. Gunasena, 1973.

[117] Rerukane Chandrawimala Nayaka Thero, *Abhidharma Mulika Karunu*, 12th ed. 2009.

[118] Nārada and Pali Text Society, *Conditional relations (Paṭṭhāna)*. Bristol: Pali Text Society, 2010.

[119] “Suwisi Pratya - Pattana Prakarana,” in *Abhidhamma Pritaka, Buddha Jayanthi Tipitaka Series*, vol. 52–3, 2006.
[120] Pinker Steven, “How the mind works,” *Ann. N. Y. Acad. Sci.*, vol. 882, no. 1, pp. 119–127, 1999.
[121] Litch Mary, “Computation, connectionism and modelling the mind,” *Philos. Psychol.*, vol. 10, no. 3, p. 357, 1997.
[122] A. V. Aho, “Computation and Computational Thinking,” *Comput. J.*, vol. 55, no. 7, pp. 832–835, Jul. 2012, doi: 10.1093/comjnl/bxs074.
[123] Vallverdu J, “The Eastern Construction of the Artificial Mind,” *Enrahonar Quardens Filasofia 47*, pp. 171–185, 2011.
[124] Elodzislaw Duch, “What constitutes a good theory of mind,” *Am. Neopragmatism Torun*, vol. 8, pp. 8–12, 1998.
[125] Zhang J., “Memory Process and the Function of sleep,” *J. Theor.*, vol. 6, no. 6, 2004.
[126] D. E. Nee and M. D’Esposito, “Working Memory,” in *Brain Mapping*, A. W. Toga, Ed. Waltham: Academic Press, 2015, pp. 589–595.
[127] P. O. A. Haikonen, “The Challenges for Implementable Theories of Mind,” *J. Mind Theory*, vol. 0, no. 1, pp. 99–110, 2008.
[128] W. A. C. Weerakoon, A. S. Karunananda, and N. G. J. Dias, “Enhancing the Functionality of Rule-Based Expert Systems,” Kuala Lumpur, Malaysia, Sep. 2015, p. 93.
[129] T. J. Cleophas and A. H. Zwinderman, *SPSS for starters and 2nd levelers*, 2. ed. Heidelberg: Springer, 2015.
[130] G. E. Meek, C. Ozgur, and K. Dunning, “Comparison of the t vs. Wilcoxon Signed-Rank Test for Likert Scale Data and Small Samples,” vol. 6, no. 1, pp. 91–106, May 2007.
[131] A. Imam, U. Mohammed, and C. Moses Abanyam, “On Consistency and Limitation of paired t-test, Sign and Wilcoxon Sign Rank Test,” vol. 10, no. 1 Ver.IV, pp. 01–06, Feb. 2014.
[132] S. C. Gupta and V. K. Kapoor, *Fundamentals of Mathematical Statistics*. Sultan Chand & Sons, 2007.
[133] R. E. Walpole, *Probability and statistics for engineers and scientists*. Boston, Mass: Pearson Education, 2012.
[134] Jim Colton, Minitab Inc., *Detecting and Analyzing Non-Normal data*. Florida, 2014.
[135] M. A. Stephen, “The Anderson-Darlin Statistics,” DEPARTMENT OF STATISTICS STANFORD UNIVERSITY STANFORD, CALIFORNIA, 1979.
[136] F. E. Grubbs, “Procedures for Detecting Outlying Observations in Samples,” *Am. Stat. Assoc. Am. Soc. Qual.*, vol. 11, no. 1, pp. 1–21, Feb. 1969.
[137] H. J. Hopcroft, R. Motwani, and D. J. Ullman, *Introduction to Automata Theory Languages, and Computation*, Third. Pearson Education, 2007.
[138] A. T. Sudkamp, *Languages and Machines: An Introduction to the Theory of Computer Science*, Third. Pearson, 2005.

[139] C. J. Martin, *Introduction to Languages and the Theory of Computation*, 2007th ed. McGraw Hill Education, 2014.

[140] W. A. C. Weerakoon, A. S. Karunananda, and N. G. J. Dias, “New Processing Model for Operating Systems,” University of Kelaniya, 2016, p. 29, [Online]. Available: http://repository.kln.ac.lk/handle/123456789/15931.

APPENDIX A
SELECTED CODE SEGMENTS

A.1 Process Switcher

This is the main controller of the program, which allows to switch between the internal and external processes during each of the "Sleep" state. Further, it made the continuation of the system. The respective source code can be seen in the Figure. A.1.

```
public class ProcessSwitcher {
    ExternalProgram ep = new ExternalProgram();
    InternalProgram ip = new InternalProgram();
    public static void processSwitch() throws IOException, InterruptedException{
        ExternalProgram.externalExecution();
        while(cancelValue==false){
            if(InsertList.txtInsertList.getText().equalsIgnoreCase("")){
                InsertList.lblAlgoName.setText("");
                InsertList.txtAreaAns.setText("");
                InternalProgram.internalExecution();
                InsertList.txtAreaAns.setText("Internal");
                lblstat.setText("");
                systemState=systemState+"\n"+"Sleep -external";
                txtAreaState.setText(systemState);
                lblstat.setText("Sleep -external");
                sleep(1000);
            } else{
                if(InsertList.txtAreaAns.getText().equalsIgnoreCase("Internal"))
                    InsertList.txtAreaAns.setText("External");
                else
                    InsertList.txtAreaAns.setText(InsertList.txtAreaAns.getText());
                lblstat.setText("");
                systemState=systemState+"\n"+"Sleep -internal";
                txtAreaState.setText(systemState);
                lblstat.setText("Sleep -internal");
                sleep(1000);
            }
        }
    }
}
```

Figure. A.1: Source Code Segment for Process Switcher

A.2 Operation Organizer

This tries to recognize the pattern of the input as mentioned in the section 4.3 and pass the input to accomplish the respective actions. The source code for the SSPM-FC Operation Organizer with respect to “Same Op”, “Same Exp”, “Different Op”, and “New Op” is showed in the Figure A.2 (a). In addition to that the longer expressions are handled through recursively calling the same methods, considering the input patterns stated in the section 4.3. Further, a part of the input pattern identification of SSPM-Sorting is showed in the Figure. A.2 (b).

(a)

```
.
.
.
    if(!op.equalsIgnoreCase("*")){
       algorithmNo=1;
    }else if(op.equalsIgnoreCase("*")){
       if(countR>1){
          if(currentMethod.equalsIgnoreCase("MulCalc1()"))
             currentMethod="MulCalc2()";
          else if(currentMethod.equalsIgnoreCase("MulCalc2()"))
             currentMethod="MulCalc1()";
          else
             currentMethod="MulCalc1()";
       }else if(countR==1){
          currentMethod=lowestMethod();
       }
    }
.
.
.
    if(validNum==false)
    {
       txtAreaAns.setText("Invalid Numerator/s");
       resultRaw="Invalid Fraction/s";
       simpResult="Invalid Fraction/s";
       if(wStr.equalsIgnoreCase("External"))
          printAns();
       else if(wStr.equalsIgnoreCase("Internal"))
          soutAns();
    }else{
       classifyNumerator();
       System.out.println("Numerator is Classified; cnc = "+numCategory+"\n");
       if((!currentOp.equalsIgnoreCase(""))&&currentOp.equalsIgnoreCase(op)){
          category="Same Op";
          ++countSmOp;
          if((countSmOp>6)||(!moduleFile.isEmpty())){
             if(!prevExp.equalsIgnoreCase(currentExp)||prevExp.equalsIgnoreCase("")){
                try {
                   executeCalcMain();
                   if(wStr.equalsIgnoreCase("External"))
                      printAns();
                   else if(wStr.equalsIgnoreCase("Internal"))
                soutAns();
            }
.
.
.
          else{
            category="Same Exp";
            if(wStr.equalsIgnoreCase("External"))
              printAns();
            else if(wStr.equalsIgnoreCase("Internal"))
              soutAns();
          }
        }else{
          if(!prevExp.equalsIgnoreCase(currentExp)||prevExp.equalsIgnoreCase("")){
              try {
              executeSubMain();
              if(wStr.equalsIgnoreCase("External"))
                printAns();
              else if(wStr.equalsIgnoreCase("Internal"))
                soutAns();
            }
.
.
.
          }
          else{
            category="Same Exp";
            if(wStr.equalsIgnoreCase("External"))
              printAns();
            else if(wStr.equalsIgnoreCase("Internal"))
              soutAns();
          }
        }
      }else{
        try {
            getModule1(op);
          }
.
.
.
        }
        if(existOp==false){
          s="Create new module for the new operator \""+op+"\"\n Click below OK button\n";
          category="New Op";
          resultRaw="not calculated";
          simpResult="not calculated";
          module="no module";
          currentMethod="no method";
          btnOk.show(true);
          btnClearS.hide();
          btnSubmit.hide();
          btnLoadFile.hide();
          btnCreateModule.hide();
          txtAreaAns.setText(s);
        }else{
            countSmOp=1;
            category="Different Op";
            if(moduleFile.isEmpty()||(!op.equalsIgnoreCase(recordedOp))){
              try {
                System.out.println("Get Module\n");
                setSubMain();
                try {
                  executeSubMain();
```

```
.
.
.
            }else{
              try {
                currentMethod=moduleFile.substring(0, 7)+algorithmNo+"()";
                module=moduleFile.substring(0, 13);
                recordedOp=op;
                executeCalcMain();
.
.
.
            }
            if(wStr.equalsIgnoreCase("External"))
              printAns();
            else if(wStr.equalsIgnoreCase("Internal"))
              soutAns();
        }
      }
        txtAreaAns.setEditable(false);
        Color c = new Color(215, 215, 215);
        txtAreaAns.setBackground(c);
        estimatedTime = System.nanoTime() - startTime;
        dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
        date = new Date();
        prevExp=a[0]+"/"+b[0]+op+a[1]+"/"+b[1];
        countFrequent.calcCount();
        if(!recordedOp.equalsIgnoreCase(op))
          moduleFile="";
        int x=0;
        while(x<noOperators){
         if((opArray[x].opCount==8)&&(opArray[x].modFile.isEmpty())&&(moduleFile.isEmpty())
                                                                        &&(redundantNF==0)){
            try {
              wait=wait+", "+"frequent-not-subsequent";
              redundantF++;
              setCalcClasses(opArray[x].opModule);
.
.
.
          }
          x++;
        }
        if((countSmOp==6)&&(moduleFile.isEmpty())&&(redundantF==0)){
          try {
            wait=wait+", "+"frequent";
            redundantNF++;
            setCalcClasses(module);
.
.
.

        }
        if(op.equalsIgnoreCase("*")){
          updateDuration();
          if((countR!=1)){
            countAlgoNo++;
            if(countAlgoNo>5){
              delInefficientAlgo();
            }
          }
        }
        createLogFile();
        System.out.println("Log file created!\n");
      }
    }

(b)
.
.
1       if(workCurList.containsAll(z)&&(z.size()==workCurList.size())){
2               System.out.println("final :"+z);
      }
.
.
4     z.retainAll(workCurList);
5     workCurList.removeAll(z);
.
.
```

Figure. A.2: (a) Operation Organizer in SSPM-FC with respect to "Same Op", "Same Exp", "Different Op", and "New Op" (b) Part of the Operation Organizer in SSPM-Sorting

In the Figure. A.2 (b), the z is the sorted previous list, whereas the workCurList is the current input (list). Further, the line 1 in Figure. A.2 (b) checks whether the two lists are equal using the containsAll() method, checking the sizes of the lists. In addition to that the line 4, finds the common elements in both the lists and removes the additional elements in the sorted list z. Then, with the line 5, the rest of the elements in the current list that are not common are found. Then, those elements were inserted into the filtered previously sorted list.

A.3 Input-Content Analyser of SSPM-FC

This works as similar to the lexical analyser of a compiler. It identifies the input, and break the input into respective components (this also similar to the concept of breaking up into lexemes). For example, in SSPM-FC, it identifies operators and operands. The source code of the Input-Content Analyser for the SSPM-FC is illustrated in the Figure. A.1. During this analysis of SSPM-FC, it had to identify, brackets, locations, numerators, denominators, and whole parts, apart from the operators and operands. Therefore, it is evident that the Input-Content Analyser is domain dependent.

```
public static void getComplexFrac(String wStr) throws IOException{
    if(openBracArr.length!=0||closeBracArr.length!=0){
      if(openBracArr.length==closeBracArr.length){
          subStr=null;
          int d=0, x=0;
          int j=0;
          for(k=openBracArr.length-1;k>=0;k--) {
            for(j=0;j<closeBracArr.length;j++){
              if(openBracArr[k]<closeBracArr[j]){
                d=closeBracArr[j];
                x=j;
                break;
              }
            }
            subStr=s.substring(openBracArr[k]+1,d );
            openBracArr =Arrays.copyOf(openBracArr, openBracArr.length-1);
            for(int l=x;l<closeBracArr.length-1;l++){
              closeBracArr[l]=closeBracArr[l+1];
            }
            closeBracArr =Arrays.copyOf(closeBracArr, closeBracArr.length-1);
            subStr=parseForFAndS(subStr);
            findWholePart();
            if(sinArr.length!=0){
              handleWholePart();  //add the whole part to denominator
            }
            findResult(wStr);
            replaceString();
            noBrac=findBracPos();
            getComplexFrac(wStr);
          }
      }else{
        InsertEquation.txtAreaAns.setText("Invalid Brackets Handling\n");
      }
    }else{
      subStr=s;
      subStr=parseForFAndS(subStr);
      findWholePart();
      if(sinArr.length!=0){
        handleWholePart();  //add the whole part to denominator
      }
      findResult(wStr);
    }
  }
```

```
public static int findBracPos(){
   int p=0, q=0;
   for(i=0;i<s.length();i++){
      if(Character.isDefined(s.charAt(i))&&(s.charAt(i)=='(')) {
         p++;
      }else if(Character.isDefined(s.charAt(i))&&(s.charAt(i)==')')) {
         q++;
      }
   }
   openBracArr=new int[p];
   closeBracArr=new int[q];
   openBracIn=0;
   closeBracIn=0;
   for(i=0;i<s.length();i++){
      if(Character.isDefined(s.charAt(i))&&(s.charAt(i)=='(')) {
         openBracArr[openBracIn]=i;
         openBracIn++;
      }else if(Character.isDefined(s.charAt(i))&&(s.charAt(i)==')')){
         closeBracArr[closeBracIn]=i;
         closeBracIn++;
      }
   }
   return openBracArr.length;
}

public static void findWholePart(){
   if(finArr!=null){
      wp = new WholeParts[finArr.length];
      for (int i = 0; i < finArr.length; i += 1) {    // initialize the array with the length similar to the length of the finArr
         wp[i] = new WholeParts();
      }
      subOp=-1;
      nextJ=0;
      neg=0;
      for(int p=0;p<finArr.length;p++){      // Read the fractions: '/', whole part, numerator, denominator
         wp[p].fIndex=finArr[p];     // read '/'
         if(nextJ<sinArr.length){
            for(int q=nextJ;q<sinArr.length;q++){
               if(sinArr[q]<finArr[p]){       // Read the fractions: denominator & whole part, considering the spaces
                  wp[p].sIndex=sinArr[q];
                  if((p==0)&&(subStr.charAt(0)=='-')){
                     wp[p].wNumber=(-Integer.parseInt(subStr.substring(subOp+2,sinArr[q])));
                     wp[p].neg=1;
                  }else if((p!=0)&&(subStr.charAt(opArr[p-1].index+1)=='-')){
                     wp[p].wNumber=(-Integer.parseInt(subStr.substring(opArr[p-1].index+2,sinArr[q])));
                     wp[p].neg=1;
                  }else
                     wp[p].wNumber=Integer.parseInt(subStr.substring(subOp+1,sinArr[q]));
                  wp[p].denom=Integer.parseInt(subStr.substring(sinArr[q]+1,finArr[p]));
                  nextJ=q+1;
                  break;
               }else{
                  wp[p].denom=Integer.parseInt(subStr.substring(subOp+1,finArr[p]));
               }
            }
         }else{
            if(subStr.charAt(0)=='-'){
               if(subOp==-1)
                  wp[p].denom=(-Integer.parseInt(subStr.substring(subOp+2,finArr[p])));
                                                   // Read the denominator, if no whole part
               else
                  wp[p].denom=Integer.parseInt(subStr.substring(subOp+1,finArr[p]));
            }else
               wp[p].denom=Integer.parseInt(subStr.substring(subOp+1,finArr[p]));
```

```
          }
          if(p==(finArr.length-1))       // find the limit considering the operator or the end of the array
             subOp=subStr.length();
          else
             subOp=opArr[p].index;
          wp[p].num=Integer.parseInt(subStr.substring(finArr[p]+1,subOp)); // read the numeraor
          fin=p-1;    // find the number of fractions in the considered string/substring
       }
       nextJ=0;
    }
 }

 public static void handleWholePart(){
    for(int i=0;i<finArr.length;i++){
       if(wp[i].neg==1&&wp[i].wNumber!=0)
          wp[i].denom=(wp[i].wNumber*wp[i].num)-wp[i].denom;
           // eg: 3 2/5 = (3*5+2)/5 => denominator=17, numerator=5
       else
          wp[i].denom=(wp[i].wNumber*wp[i].num)+wp[i].denom;
    }
 }

 public static String parseForFAndS(String subStr){
    int fIndex=0, sIndex=0;
    int p=0,q=0,r=0, t=0;
    for(i=1;i<subStr.length();i++)
    {
       if((!Character.isDigit(subStr.charAt(i)))&&(!(subStr.charAt(i)==''))&&(!(subStr.charAt(i)=='/'))&&
                                                                   (!(subStr.charAt(i)=='('))&&(!(s.charAt(i)==')'))){
           // this will find the positions of the operators
          if(t!=i-1){ //pass without reading negative sign
             r++;
          }
          t=i;
       }else if(subStr.charAt(i)=='/') {   // find the positions of the fractions sign ('/')
          p++;
       }else if(subStr.charAt(i)==' ') {   // find the positions of the spaces assuming that the space is used only to separate
the whole part from the denominator
          q++;
       }
    }
    opArr=new opSetArray[r]; finArr=new int[p]; sinArr=new int[q];
    for (int i = 0; i < r; i += 1) {
          opArr[i] = new opSetArray();
    }
    opIn=0; fin=0; spaceIn=0;
    t=0;
    for(i=1;i<subStr.length();i++)
    {
       if((!Character.isDigit(subStr.charAt(i)))&&(!(subStr.charAt(i)==' '))&&(!(subStr.charAt(i)=='/'))&&
                                                                      (!(subStr.charAt(i)=='('))&&(!(s.charAt(i)==')')))
       {   // this will find the positions of the operators
          if(t!=i-1){ //pass without reading negative sign
             opArr[opIn].index=i;
             opArr[opIn].op=""+subStr.charAt(i);
             opIn++;
          }
          t=i;
       }else if(subStr.charAt(i)=='/') {   // find the positions of the fractions sign ('/')
          finArr[fin]=i;
          fin++;
       }else if(subStr.charAt(i)==' ') {   // find the positions of the spaces assuming that the space is used only to separate
                                                                        the whole part from the denominator
          sinArr[spaceIn]=i;
```

```
            spaceIn++;
          }
//          findWholePart();
        }
        return subStr;
      }
```

Figure. A.3: Input-Content Analyser of SSPM-FC

A.4 Small Compiler

The smaller compiler compiles the dynamically changed source codes and allow their activation and execution. The source codes related to the compiling, loading, and execution are included in the Figure. A.4. Further, this small compiler is not domain dependent.

```
public static synchronized void compileExecuteCalcMain() throws MalformedURLException, ClassNotFoundException,
InstantiationException, IllegalAccessException, IOException{
        DiagnosticCollector<JavaFileObject> identifiedD = new DiagnosticCollector<JavaFileObject>();
       //Start Compiling
        JavaCompiler javaCompiler = ToolProvider.getSystemJavaCompiler();
        StandardJavaFileManager manageFile = javaCompiler.getStandardFileManager(identifiedD, null, null);
        //Set the Class Path
       List<String> listOp = new ArrayList<String>();
        listOp.add("-classpath");
        listOp.add(System.getProperty("java.class.path") + ";dist/methodsSubMain.jar");
        Iterable<? extends JavaFileObject> compilationUnit
              = manageFile.getJavaFileObjectsFromFiles(Arrays.asList(SubMain));
        JavaCompiler.CompilationTask job = javaCompiler.getTask(null,  manageFile, identifiedD, listOp,
                                                                    null,compilationUnit);
        if ( job.call()) {
           //Start Loading and Executing
           URLClassLoader cl = new URLClassLoader(new URL[]{new
                                                                                File("./").toURI().toURL()});
           Class<?> loadedClass = cl.loadClass("compilerSubMain.SubMain");
           Object insOb = loadedClass.newInstance();
           if (insOb instanceof dynamicMain) {
              dynamicMain exeMod = (dynamicMain)insOb;
              exeMod.subMainMethod();
           }

        } else {
           for (Diagnostic<? extends JavaFileObject> diagnostic : identifiedD.getDiagnostics()) {
              System.out.format("Error on line %d in %s%n",
                    diagnostic.getLineNumber(),
                    diagnostic.getSource().toUri());
           }
        }
        manageFile.close();
```

Figure. A.4: Smaller Compiler – The execution of dynamically created java module – SubMain.java

A.5 Write Engine

The Figure. A.5 shows the construction of dynamically changing calculation class, before the particular operation become frequent, and it is a part of the Write Engine. In addition to the part displayed in this figure, the Write Engine includes Update Tactic Memory, Create Log File, Create/Delete Module, and Update Program Codes. From these, the Update Tactics Memory, Create/ Delete Modules and Update Program Codes are domain dependent.

```
public static synchronized void setCalcClasses(String m) throws IOException{
    wait=wait+", "+"set Calc-Classes";
    StringBuilder bStr = new StringBuilder(64);
    bStr.append("package compilerSubMain;\n");
    bStr.append("import static FracLibrary."+m+".*;\n");
    bStr.append("public class "+m+"Main implements FracLibrary.methodsSubMain.dynamicMain {\n");
    bStr.append("    public void subMainMethod() {\n");
    bStr.append("        "+m.substring(0, 7)+algorithmNo+"();\n");
    bStr.append("    }\n");
    bStr.append("}\n");
    recordedOp=op;

    subMain = new File("compilerSubMain/"+m+"Main.java");
    subMain.createNewFile();
    if (subMain.exists()) {
      Writer writer = null;
      try {
        try {
          writer = new FileWriter(subMain);
          writer.write(bStr.toString());
          writer.flush();
.
.
.
            writer.close();
          } catch (Exception e) {
          }
        }
     }
      System.out.println("is subMain executable? : "+ subMain.canExecute());
      subMain.setExecutable(true);
      Reader r = new FileReader(subMain);
      r.close();
      String fileName = "F:" + File.separator +  "TestingProgs" + File.separator +"v10"+ File.separator
       +"FracLib1"+File.separator + "OpFile.txt";
      File f = new File(fileName);
      FileReader reader = new FileReader(f);
      BufferedReader buffReader = new BufferedReader(reader);
      String s;
      StringBuilder ns = new StringBuilder();
      while((s = buffReader.readLine()) != null) {
        System.out.println(s);
        String[] tokenModules = s.split(",");
        if(op.equalsIgnoreCase(tokenModules[0])){
          if(tokenModules.length<5){
            ns.append(s+","+tokenModules[3]+"Main()\n");
```

```
                }
                currentMethod=tokenModules[3].substring(0, 7)+algorithmNo+"()";
                if(tokenModules.length>4){
                    moduleFile=tokenModules[4];
                    System.out.println(moduleFile);
                }
                continue;
            }
            ns.append(s+"\n");
        }
    if (subMain.exists()) {
        Writer writer = null;
        try {
            try {
                writer = new FileWriter(f);
                writer.write("");
                writer.write(ns.toString());
                writer.flush();
.
.
.
                try {
                        writer.close();
                } catch (Exception e) {
                }
            }
    }
    System.out.println("class : "+subMain.getClass());
  }
```

Figure. A.5: Write Engine

A.6 Terminating Point

The terminating point is determined once the noimprovements string variable is set 'Yes'. The respective code segment is showed in the Figure. A.6.

```
.
.
        do{
             ProcessSwitcher.processSwitch();
           }while(noImprovements.equalsIgnoreCase("No"));
}
```

Figure. A.6: Terminating Point

A.7 Two Methods for Multiplication

These were used to show the ability in removing the inefficient algorithm. The respective two multiplication methods are displayed in the Figure. A.7.

```
public class MulCalcModule {
   static int g1=1,g2=1;
   static String option="";

   public static void MulCalc1(){
      if(a[0]==b[1]){
         sum=a[1];
         lcm=b[0];
         option="option1";
      }else if(a[1]==b[0]){
         sum=a[0];
         lcm=b[1];
         option="option2";
      }else if(((g1=GCDCalc.gcdCalc(a[0], b[1]))>1)&&((g2=(GCDCalc.gcdCalc(a[1], b[0])))>1)){
         sum=(a[0]/g1)*(a[1]/g2);
         lcm=(b[0]/g2)*(b[1]/g1);
         option="option3";
      }else if((!((GCDCalc.gcdCalc(a[1], b[0]))>1))&&((GCDCalc.gcdCalc(a[0], b[1]))>1)){
         sum=(a[0]/gcd)*a[1];
         lcm=b[0]*(b[1]/gcd);
         option="option4";
      }else if((!((GCDCalc.gcdCalc(a[0], b[1]))>1))&&((GCDCalc.gcdCalc(a[1], b[0]))>1)){
         sum=(a[1]/gcd)*a[0];
         lcm=b[1]*(b[0]/gcd);
         option="option5";
      }else{
         sum=a[0]*a[1];
         lcm=b[0]*b[1];
         option="option6";
      }
      System.out.println(option);
      sysOut.sysPrintOut();
   }

   public static void MulCalc2(){
      sum=a[0]*a[1];
      lcm=b[0]*b[1];
      sysOut.sysPrintOut();
   }
}
```

Figure. A.7: Two Multiplication Methods; MulCalc1(), and MulCalc2()

APPENDIX B

DATA SETS

B.1 SSPM-FC: Plus Operator

Expression	After	Before
1/4+3/5	31398871	46305152
1/5+2/6	28055953	46220762
1/5+3/6	26188344	48219138
1/5+4/6	24858247	43752537
1/5+5/6	26436953	44270663
1/6+3/7	22876977	39920004
1/6+4/7	33091237	40752502
1/6+5/7	25441757	45468092
1/6+6/7	26909842	50043032
1/7+1/8	19439405	41918760
1/7+2/8	22294609	40913680
1/7+3/8	26359786	40584102
1/7+4/8	21690573	42808279
1/7+5/8	19103365	42386708
1/7+6/8	19606665	44867477
1/8+1/9	33416634	42814741
1/8+2/9	21614165	41114772
1/8+6/9	18809140	40928505
1/8+7/9	24965446	43725168
1/8+8/9	18906074	44116708
2/3+1/4	31284450	43188415
2/4+1/5	35251551	49724477
2/4+2/5	29936487	42190558
2/4+3/5	27687221	45784746
2/5+3/6	27863604	39530745
2/5+4/6	25397280	42957673
2/5+5/6	26113457	43140138
2/6+2/7	21655600	40525562
2/6+3/7	24364450	41137200
2/7+1/8	19221587	41077519
2/7+3/8	22277123	45008887
2/7+6/8	19449669	40893153
2/7+7/8	19783429	43819442
2/8+3/9	22085533	39519341
2/8+4/9	18449150	44195776
2/8+5/9	18842592	45277644
2/8+8/9	18915958	40433568
3/4+2/5	30800917	40435849
3/4+3/5	27169476	42150643
3/4+4/5	31109587	46524110
3/5+5/6	23869133	41768227
3/6+1/7	24605076	45751294
3/6+2/7	23441099	45925776
3/6+3/7	20367698	45904869
3/6+5/7	22050941	51226395
3/6+6/7	22534095	41949551
3/7+1/8	28399596	43461353
3/7+4/8	22353150	40963098
3/7+5/8	19237933	40974502
3/8+1/9	23275360	43783328
3/8+2/9	27342057	43080837
3/8+5/9	18453713	40774550
3/8+7/9	19478560	46939220
4/5+4/6	30219688	41509734
4/5+5/6	27565958	43327925
4/6+1/7	22458448	40312685
4/6+2/7	26942534	46162221
4/6+5/7	22266479	40547609
4/6+6/7	19459172	43671949
4/7+1/8	21896986	50231199
4/7+3/8	18942567	43375442
4/7+4/8	19834367	43434363
4/8+4/9	18822825	48070504
4/8+7/9	18982862	40773030
4/8+8/9	18442308	40578020
5/6+1/7	22044859	45284867
5/6+4/7	25862187	40189901
5/6+5/7	19466776	41334491
5/6+6/7	19442066	44910432
5/7+1/8	21029896	42889628
5/7+2/8	18942568	45276884
5/7+5/8	19306738	41709685
5/7+6/8	21888623	43739233
5/7+7/8	19701319	41711206
5/8+1/9	22530293	47934036
5/8+4/9	28453575	40125658
5/8+5/9	18741855	40523660
5/8+6/9	18183435	42726930
5/9+4/9	18181155	39104993
5/9+5/9	18379206	41797877
5/9+6/9	18585620	45532716
6/7+2/8	25045274	49050875
6/7+3/8	19169129	42026719
6/7+4/8	20264682	40821307
6/7+5/8	18631616	43219587
6/8+1/9	22527632	40690541
6/8+2/9	28435329	41198022
6/8+3/9	18948270	46130290
6/8+6/9	18421781	39891114
6/8+7/9	24604317	43453371
6/8+8/9	18208525	42975919
7/8+3/9	18256802	49801265
7/8+4/9	18577256	41933586
7/8+5/9	25263092	43497086
7/8+8/9	21105543	43063730
7/9+1/9	22671323	44643576
7/9+2/9	18129456	43181573
7/9+5/9	18062172	40931167
7/9+6/9	18697760	43731250
7/9+7/9	18566613	42941706

B.2 SSPM-FC: Minus Operator

Row Labels	After	Before
1/4-4/5	29116924	50861864
1/5-2/6	26226368	53276491
1/5-3/6	26639956	50250605
1/5-4/6	24902352	43668544
1/5-5/6	25880063	43862413
1/6-1/7	22686538	42909032
1/6-2/7	22908917	55700240
1/6-3/7	23271948	48092191
1/6-5/7	25319742	46775399
1/7-2/8	19281277	43272822
1/7-3/8	18299003	48284540
1/7-4/8	19331454	50916984
1/7-5/8	18250346	46844203
1/7-6/8	19309027	43453007
1/7-7/8	19574741	50390495
1/8-1/9	25088239	46287304
1/8-2/9	18766191	53242658
1/8-3/9	19490351	46789843
1/8-4/9	18613377	43052343
1/8-5/9	24982181	42777124
1/8-7/9	20026344	39478681
2/3-1/4	31788903	54892450
2/3-3/4	31206154	47348264
2/4-2/5	29265177	56758159
2/4-3/5	27110945	47510962
2/5-1/6	27582314	53479863
2/5-5/6	23040064	53507614
2/6-1/7	28418994	47732202
2/6-2/7	27044041	43478475
2/6-3/7	21345416	41934742
2/6-5/7	20652808	47654653
2/6-6/7	20641404	44502563
2/7-1/8	23264725	58094338
2/7-2/8	28076491	47776677
2/7-3/8	19666355	40939165
2/7-4/8	19347800	46143992
2/8-1/9	27113226	43248113
2/8-2/9	21535865	50733758
2/8-3/9	17948138	43798930
2/8-4/9	18466644	41829064
2/8-5/9	18431291	44322379
2/8-6/9	26254498	56106606
2/8-7/9	18054956	45204675
2/8-8/9	18169378	38161128
3/4-2/5	26880962	54073636
3/4-3/5	27152000	47130065
3/5-1/6	23298937	46763994
3/5-2/6	23392451	43860893
3/5-3/6	25556567	42484420
3/5-4/6	22789935	51319168
3/5-5/6	23460875	45100898
3/6-1/7	23195160	43356832
3/6-3/7	20335394	46833179
3/6-5/7	24336329	44628768
3/6-6/7	23048808	59534675
3/7-3/8	22627237	46939997
3/7-4/8	20360483	40481861
3/7-5/8	19084366	52101869
3/8-1/9	18306986	48663156
3/8-3/9	18218795	47672140
3/8-4/9	24259542	40498966
3/8-5/9	18254908	42356313
3/8-6/9	18617178	45071627
3/8-7/9	18998075	47557339
3/8-8/9	18251487	40640758
4/5-1/6	24801236	42924237
4/5-3/6	30792565	56432383
4/5-4/6	33301846	51464760
4/6-1/7	20363905	48064060
4/6-2/7	26475356	42938302
4/6-5/7	20053334	42905990
4/6-6/7	23957714	55692257
4/7-1/8	23558190	48392119
4/7-2/8	18802304	42809436
4/7-3/8	18872629	47688486
4/7-4/8	19516961	46480793
4/7-5/8	19076383	49475506
4/7-6/8	27434821	59831942
4/7-7/8	18700428	47585469
4/8-1/9	18194086	40978699
4/8-2/9	30650775	54082380
4/8-3/9	18335878	49363367
4/8-5/9	18420268	39269226
4/8-7/9	18655571	48857785
4/8-8/9	25002708	41174469
5/6-2/7	19451958	42765340
5/6-5/7	19341337	44517388
5/6-6/7	24361418	57025775
5/7-1/8	22313623	40437385
5/7-2/8	18515301	42300813
5/7-4/8	19418506	47806328
5/7-5/8	25493085	42909412
5/7-6/8	19751885	47420110
5/8-1/9	18409624	46494097
5/8-3/9	17633385	40490984
6/7-1/8	23173873	46529450
6/7-2/8	19374030	40644939
6/7-3/8	22603288	51231357
6/7-6/8	18617178	46941518
6/7-7/8	22070337	50587785

B.3 SSPM-FC: Multiplication Operator

Expression	After	Before
1/4*2/5	30884558	48295183
1/4*3/5	25430742	41692215
1/5*1/6	27516930	43978355
1/5*2/6	24353816	48870710
1/5*3/6	22243298	41697917
1/5*4/6	21915621	49531387
1/6*1/7	20803723	37905677
1/6*2/7	23280691	43676907
1/6*3/7	20585524	49471705
1/6*4/7	21736957	42714781
1/7*2/8	20509116	38650364
1/7*4/8	19317389	43389524
1/7*5/8	19122760	49671277
1/7*6/8	19542049	41599081
1/7*7/8	19553073	49697886
1/8*3/9	22065775	51617956
1/8*4/9	23156386	42799552
1/8*5/9	20667634	51169774
1/8*6/9	27041000	47801387
1/8*7/9	18268973	39674832
1/8*8/9	20377589	49710811
2/5*1/6	28587774	49245144
2/5*2/6	24275128	48498937
2/5*3/6	25796053	45122946
2/5*5/6	20943993	42287509
2/6*1/7	26700017	46004862
2/6*2/7	21400156	37806842
2/6*5/7	20126320	48185324
2/6*6/7	20508736	41119350
2/7*1/8	20049531	42063609
2/7*2/8	23125976	51313846
2/7*3/8	18710312	44425015
2/7*5/8	18958920	41696017
2/8*1/9	18465123	42938302
2/8*2/9	29408869	49989831
2/8*3/9	18704229	37194822
2/8*4/9	23630037	51551812
2/8*5/9	23091763	43705417
2/8*7/9	18193325	45487497
3/4*2/5	23357478	40052687
3/4*3/5	23891950	43086936
3/4*4/5	24682633	48528207
3/5*1/6	21893953	49926349
3/5*2/6	21202865	40530899
3/5*4/6	21325649	42467313
3/5*5/6	21242400	48289861
3/6*1/7	19902039	49680021
3/6*3/7	24270946	50999094
3/6*4/7	19953738	40753658
3/6*5/7	19938912	49705109
3/6*6/7	20345658	42046502
3/7*1/8	18338538	48003619
3/7*2/8	19353502	42032817
3/7*3/8	24933524	50799521
3/7*4/8	26818620	43240511
3/7*6/8	21304742	43692113
3/7*7/8	22838972	43181589
3/8*1/9	18175460	42185632
3/8*4/9	25054027	42058666
3/8*5/9	18239702	42514450
3/8*6/9	19748463	50101971
4/5*1/6	21927405	40337029
4/5*2/6	30426874	48468526
4/5*3/6	21002914	42436522
4/5*4/6	21188800	46796686
4/5*5/6	25960272	46308211
4/6*1/7	20465401	49531387
4/6*2/7	24812260	43255716
4/6*3/7	19768991	48379954
4/6*4/7	22011035	42454768
4/6*6/7	19218553	43573130
4/7*1/8	22399915	41922198
4/7*2/8	22803620	48217635
4/7*3/8	19806244	44140673
4/7*4/8	19343619	44877378
4/7*5/8	19753026	42475676
4/7*6/8	19416985	45937198
4/7*7/8	18758969	43199836
4/8*1/9	18067881	44305652
4/8*2/9	18502757	48832316
4/8*3/9	18175840	36020961
4/8*4/9	21838453	47978910
4/8*5/9	19421547	43215042
4/8*8/9	24417299	49997434
5/6*2/7	19334875	38457635
5/6*3/7	26484100	47609038
5/6*4/7	19308646	42888504
5/6*5/7	19253526	49562558
5/6*6/7	19076763	39521256
5/7*3/8	30015186	53766866
5/7*4/8	18984009	41505948
5/7*5/8	18842979	48559758
5/7*6/8	18770373	47611318
5/7*7/8	26317220	51986688
6/7*1/8	18443075	43519151
6/7*2/8	18531268	48437735
6/7*3/8	19089307	42168526
6/7*4/8	22547408	47555818
6/7*5/8	18505798	43840366
6/7*7/8	18712592	43529794

B.4 SSPM-QES – Positive Discriminant

Equation	Before	After
$1.0x^2 0.0x-1.0=0$	36948479	60854485
$1.0x^2-1.0x-2.0=0$	16363724	54086920
$1.0x^2-10.0x-11.0=0$	15614855	45151819
$1.0x^2 10.0x-11.0=0$	21716422	41022020
$1.0x^2-11.0x-12.0=0$	16174035	50909742
$1.0x^2 11.0x-12.0=0$	26516782	44694515
$1.0x^2-12.0x-13.0=0$	17626156	26168956
$1.0x^2-13.0x-14.0=0$	20103124	27660991
$1.0x^2 13.0x-14.0=0$	22494180	29958155
$1.0x^2-14.0x-15.0=0$	15469644	24424892
$1.0x^2-15.0x-16.0=0$	15983208	47542876
$1.0x^2 15.0x-16.0=0$	25737123	38905421
$1.0x^2-16.0x-17.0=0$	15714451	46933517
$1.0x^2 16.0x-17.0=0$	20231230	50548993
$1.0x^2-17.0x-18.0=0$	15749043	28690021
$1.0x^2-18.0x-19.0=0$	19897469	25934033
$1.0x^2 18.0x-19.0=0$	20958431	28019079
$1.0x^2-19.0x-20.0=0$	16294159	26815948
$1.0x^2 19.0x-20.0=0$	20897607	24647271
$1.0x^2-2.0x-3.0=0$	19934723	25773616
$1.0x^2 2.0x-3.0=0$	23526631	25875492
$1.0x^2-20.0x-21.0=0$	15792379	40632000
$1.0x^2 20.0x-21.0=0$	24059200	39478286
$1.0x^2-21.0x-22.0=0$	16274391	44434122
$1.0x^2 21.0x-22.0=0$	19098423	44753816
$1.0x^2-22.0x-23.0=0$	20175729	28914680
$1.0x^2 22.0x-23.0=0$	18821684	27037187
$1.0x^2-23.0x-24.0=0$	15497774	28027063
$1.0x^2-24.0x-25.0=0$	19754158	26043132
$1.0x^2 24.0x-25.0=0$	20888865	21884441
$1.0x^2-25.0x-26.0=0$	15733078	40590565
$1.0x^2 25.0x-26.0=0$	20761899	39505276
$1.0x^2-26.0x-27.0=0$	24853305	44967453
$1.0x^2 26.0x-27.0=0$	18019976	46110522
$1.0x^2-27.0x-28.0=0$	15465461	27196845
$1.0x^2 27.0x-28.0=0$	18850195	29136680
$1.0x^2-28.0x-29.0=0$	14696827	29744518
$1.0x^2 28.0x-29.0=0$	17970179	27537447
$1.0x^2-3.0x-4.0=0$	18544185	25397660
$1.0x^2 3.0x-4.0=0$	25611677	31433463
$1.0x^2-30.0x-31.0=0$	18365901	38906181
$1.0x^2 30.0x-31.0=0$	20159384	57058065
$1.0x^2-31.0x-32.0=0$	16309744	47726861
$1.0x^2 31.0x-32.0=0$	17458516	52243260
$1.0x^2 32.0x-33.0=0$	17562293	24000659
$1.0x^2-33.0x-34.0=0$	16112834	30305599
$1.0x^2 33.0x-34.0=0$	18102087	27722193
$1.0x^2-34.0x-35.0=0$	19323844	24505101
$1.0x^2 34.0x-35.0=0$	21271662	27006017
$1.0x^2-35.0x-36.0=0$	15846737	39596888
$1.0x^2 35.0x-36.0=0$	29032142	40364383
$1.0x^2-36.0x-37.0=0$	15757026	41254282
$1.0x^2 36.0x-37.0=0$	17533022	49586488
$1.0x^2-37.0x-38.0=0$	15013860	26261709
$1.0x^2 37.0x-38.0=0$	17185198	26700387
$1.0x^2-38.0x-39.0=0$	21272042	28445973
$1.0x^2 38.0x-39.0=0$	17690778	28661129
$1.0x^2-39.0x-40.0=0$	17766425	23927294
$1.0x^2 39.0x-40.0=0$	22068808	24047797
$1.0x^2-4.0x-5.0=0$	17160108	27349279
$1.0x^2 4.0x-5.0=0$	22190072	24522207
$1.0x^2-40.0x-41.0=0$	15271212	51488309
$1.0x^2 40.0x-41.0=0$	17077619	67442245
$1.0x^2-41.0x-42.0=0$	15464701	41215508
$1.0x^2 41.0x-42.0=0$	28581681	43710342
$1.0x^2-42.0x-43.0=0$	15118776	26828494
$1.0x^2 42.0x-43.0=0$	19891007	26578363
$1.0x^2-43.0x-44.0=0$	15145767	27008678
$1.0x^2 43.0x-44.0=0$	21573870	27409342
$1.0x^2-44.0x-45.0=0$	24602415	25498396
$1.0x^2 44.0x-45.0=0$	16601690	25915026
$1.0x^2-45.0x-46.0=0$	15289079	40784054
$1.0x^2-46.0x-47.0=0$	15112695	41102228
$1.0x^2 46.0x-47.0=0$	16367145	42119092
$1.0x^2-47.0x-48.0=0$	15240422	28721951
$1.0x^2 47.0x-48.0=0$	27760208	29265927
$1.0x^2-48.0x-49.0=0$	16879188	28070397
$1.0x^2 48.0x-49.0=0$	20780906	27602830
$1.0x^2-49.0x-50.0=0$	19485022	23799948
$1.0x^2 49.0x-50.0=0$	16419984	24036012
$1.0x^2-5.0x-6.0=0$	20467674	37043132
$1.0x^2 5.0x-6.0=0$	23019908	51851720
$1.0x^2-50.0x-51.0=0$	20352873	60901242
$1.0x^2 50.0x-51.0=0$	17861459	40744139
$1.0x^2-51.0x-52.0=0$	14861044	41934726
$1.0x^2-52.0x-53.0=0$	14886894	26309987
$1.0x^2-53.0x-54.0=0$	14734079	27834334
$1.0x^2-54.0x-55.0=0$	18733113	27003355
$1.0x^2-55.0x-56.0=0$	15084565	39745522
$1.0x^2-56.0x-57.0=0$	19356156	43831606
$1.0x^2-57.0x-58.0=0$	14741302	26663515
$1.0x^2-58.0x-59.0=0$	14925668	25541733
$1.0x^2-59.0x-60.0=0$	19431424	24837721
$1.0x^2-6.0x-7.0=0$	16188101	47326958
$1.0x^2 6.0x-7.0=0$	39150228	44588457
$1.0x^2-60.0x-61.0=0$	15341157	46761316
$1.0x^2-61.0x-62.0=0$	15146527	42536102
$1.0x^2-62.0x-63.0=0$	19014033	26869929
$1.0x^2-63.0x-64.0=0$	14944675	27767430
$1.0x^2-64.0x-65.0=0$	14865607	28664171

B.5 SSPM-Sorting

As it has been worked with the lists with larger number of elements such as two million elements, here a part of a sample lists (unsorted and sorted input) are mentioned below.

Unsorted Input: [91, 52, 74, -143, 112, -97, -120, 9, 30, 146, 136, ……..

Sorted list : [-391, -366, -343, -338, -337, -336, -334, -320, ……..

APPENDIX C
PUBLICATION

1. Weerakoon, C., Karunananda, A. & Dias, N. Human-mind-inspired processing model for computing. Mind Soc (2020). https://doi.org/10.1007/s11299-020-00236-2, Springer Verlang GmbH Germany **(Scopus Indexed)**.

2. C Weerakoon, A Karunananda, N Dias. "Formal Verification of Conditionally Evolving Memory." International Journal of Computer Engineering and Information Technology 11, no. 11 (2019): 243-257, Dubai - United Arab Emirates.

3. Weerakoon C., Karunananda A., Dias N. (2019) Six-State Continuous Processing Model for a New Theory of Computing. In: Hemanth J., Silva T., Karunananda A. (eds) Artificial Intelligence. SLAAI-ICAI 2018. Communications in Computer and Information Science, vol 890. Springer, Singapore **(Scopus Indexed).**

   Book: Artificial Intelligence

   Print ISBN: 978-981-13-9128-6, Electronic ISBN: 978-981-13-9129-3

   Copyright Year: 2019

   DOI:https://doi.org/10.1007/978-981-13-9129-3_3

   Chapter: 3

4. W.A.C. Weerakoon, A.S. Karunananda and N.G.J. Dias, (2016) New Processing Model for Operating Systems, Proceedings of International Postgraduate Research Conference 2016, University of Kelaniya, December, 2016, p153.

5. W.A.C. Weerakoon, A.S. Karunananda and N.G.J. Dias, (2015), Enhancing the Functionality of Rule-Based Expert Systems, 4th Int'l Conference on Advances in

Engineering Sciences & Applied Mathematics (ICAESAM'2015) Kuala Lumpur (Malaysia), Dec. 8-9, 2015, p95. **(Received Best Session Paper Award)**

6. W.A.C. Weerakoon, A.S. Karunananda and N.G.J. Dias, (2015), Conditionally Evolving Memory for Computers, Proceedings of the 15th International Conference on Advances in ICT for Emerging Regions, DOI: 10.1109/ICTER.2015.7377704, ISBN: 9781467394413, IEEE **(Scopus Indexed).**

7. Weerakoon W. A. C., Karunanda A. S. and Dias N. G. J. (2013), A Tactics Memory for a New Theory of Computing, The 8th International Conference on Computer Science & Education (ICCSE 2013), April 26-28, 2013, Colombo, Sri Lanka, DOI: 10.1109/ICCSE.2013.6553901, ISBN: 9781467344623, IEEE **(Scopus Indexed).**

Submitted to SCOPUS indexed journal:

1. Weerakoon W. A. C., Karunanda A. S. and Dias N. G. J., On Computing Memory as a Result of Processing