On Computing Memory as a Result of Processing

Chinthanie Weerakoon[1], Asoka Karunananda[2], Naomal Dias[3]
[1]Department of Statistics & Computer Science, University of Kelaniya, Dalugama, Sri Lanka.
[2]Department of Computational Mathematics, University of Moratuwa, Katubedda, Sri Lanka.
[3]Department of Computer Systems Engineering, University of Kelaniya, Dalugama, Sri Lanka.
email : [1]chinthanie@kln.ac.lk, [2]asokakaru@uom.lk, [3]ngjdias@kln.ac.lk

***Abstract*—It was difficult to find a computing model that has been constructed by imitating an Eastern-Philosophical-Approach-based human mind model to improve the computational efficiency. In this context, introducing a computing model that displays the features of the human mind with its evolving memory and has the ability to improve the processing power in subsequent program execution cycles, was a great research challenge. The Six-state Continuous Processing Model was proposed to fill this gap. This paper presents an extended research work on this model. Further, this new model has been compared with different computing models. As the basis for these comparisons, it takes the implementation and the execution of Quicksort algorithm. In this regard, each of these existing computing models have used different empirical settings. Therefore, the proposed model was compared with these computing models by using different experimental setups and conducting the experiments accordingly and separately. When conducting the experiments, it could identify different ranges of inputs, and experimental setups that enable this model to show better performance than the currently existing models. In some experimental scenarios, the performance improvement of the proposed model was more than 80% with compared to the other computing models.**

Keywords—Continuous Processing; Evolving Memory; Smaller Tactics Memory; Computational Efficiency; Human Memory; Conditional Phenomena; Memory as a Result of Processing.

I. INTRODUCTION

Von-Neumann introduced the computer architecture where memory is separated from the processor [1]. This architecture has been practiced in developing computers with various memory power and processing power. However, the demand for computers with high memory and processing power remains a challenge. In this context, while many hardware solutions including high speed memories (RAM, Cache [2], Registers) and processors (multi cores, GPU) [3] have been introduced, the development of new software technologies to use power of such hardware has been rather insignificant, as non of these software level introductions are still efficient to the expectation[4]. Therefore, finding a new computing model to enhance the computational efficiency has been a continuous research challenge.

This research has identified that the human mind is still the best computer which can generate better solutions over subsequent execution cycles of same program in a shorter period [5]. This has not been the case in computation on Von-Neumann machine. The study in [5] also revealed that mind is quite different from Von-Neumann computer, because the mind has no separate units as memory and processor, but memory is a result of continuous processing [6] in the mind. Mind operates as a continuous flow of thoughts [7] pertaining to an input or conditions. Thought flows can occur with respect to inputs coming from five sensors as well as from the mind itself [7]. The way mind react on an input is dependent on the past state of the mind. For instance, reactions to similar type of inputs inertia will be developed. This memory evolves over the cycles and aids the processing efficiency and accuracy. Therefore, as inspired from the Eastern Philosophical Approach (EPA) [38], we postulate that a computer with evolving memory[8] can be modelled by developing a processing model leading to software solution for improving computational efficiency and accuracy[5].

Knowing the fact that processing speed is dependent on the states and actions to manage the processors, we have critically studied various memory and processing models in both the hardware and softtware level. Most of these introductions were based on the memory and the processor sepretation. When it came to hardware level, there were different processor and memory models and connective mechanisms that were presented to improve computing power. In fact, different processors with different speedups have been invented with the time. For example, so far the world's fastest processor [9] is Intel's core i9 and also it is the best gaming processor [10]. Intel could do this, because of the new chipset. i.e. X299 [9]. Further, it has been expanded its processing power from eight core i9-9900 to eighteen core i9-9980XE [9]. These were achieved through different hardware formations such as increasing the dencity and reducing the size of chips [11]. Further, to support processing, various memory models were introduced, such as RAM, SRAM, DRAM, SDRAM. However, together with the bandwidth-wall, the disparity of speed between the CPU and the memory that is called as the memory-wall [12] degrades the performance of computing. Therefore, the separation between the memory and the processing is evident and this separation is one of the major obstacle in performance gaining in the computer at hardware level. Then, the other obstacles 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.

Further, 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 each other and operated in distributed or parallel manner in the environments, which is 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 finding best solutions, where the focus was sometimes bit deviated from enhancing the computational efficiency. 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 [13] having inspirations from the behavior of natural complex organizations [14] 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 [15]. However, when the 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 [14]. In fact, the concepts such as logical agents[16], long short term memories and reinforcement learning[17], were quite impressive. Many of those had insights from the human mind according to the theories introduced in Bartlett's Remembering [18], constructive memory [19], and Atkinson-Shiffrin [20] and the Baddeley [21] 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 [22]. There are many aspects in the field of computing that are benefited from GA. For example, for CPU scheduling GA has been applied [23] in order to maximize CPU throughput or utilization [24] or optimize the waiting time [25]. 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 modeling systems with the incremental and dynamic slight changes in input data [26]. There, the memory management [27] was done through the graphs and memorization [28]. Self-adjusting computing [29] 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 [30], even using GA [31]. In some cases, it has also been used different program transformation [32] techniques to enable adaptability and achieve speeding up. Further, it has been discussed memory and processing in parallel computing and neural computing also.

Finally, it has been come up with six-state processing model (SSPM) to develop the said evolvable memory. This processing model involves a set of special actions together with an extended Ready state [33] than traditional ready state, and a new Sleep state and a Terminate state that are deviated from traditional Exit state. Identification of new states and actions are based on the EPA. The SSPM has been formally validated for Turing Machine compatibility [37] and tested for some real-world problem solving. The results show that new processor model has been able to aid evolution of the memory and improved efficiency and accuracy in processing, with continuous processing. This model has been used to customize several existing programs such as Fraction Calculator (FC), QuickSort, and Quadretic Equation Solver (QES) introducing SSPM-FC, SSPM Sorting (SSPM Insertion, and SSPM-S-Equal), 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. This paper focuses on the work related to SSPM-Sorting.

The rest of the article is organized as below. The Section 2 explains th proposed model, whereas the Section 3 discusses the research methodology, while Section 4 reports the results and discusses the findings. Finally, the last section concludes the work.

II. Proposed Model

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. 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 by an internal process. Then, the particular operation for the process has been arbitrarily picked out from the bunch of operations that are stored in the initial smaller tactics memory. Further, the instructions saved in the knowledgebase can be executed through this smaller memory. The system shifts to the internal mode once an internal input is entered. The system can receive an external input only when the present inner process sleeps. After moving to the external mode, if there is no external input, the system can move back to the inner process. However, if there is no external input, the inner process can be 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 organizes the smaller 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.

The newly presented computing model comprises of six states, specifically "New", "Ready", "Running", "Blocked", "Sleep", and "Terminate" as shown in figure 1.

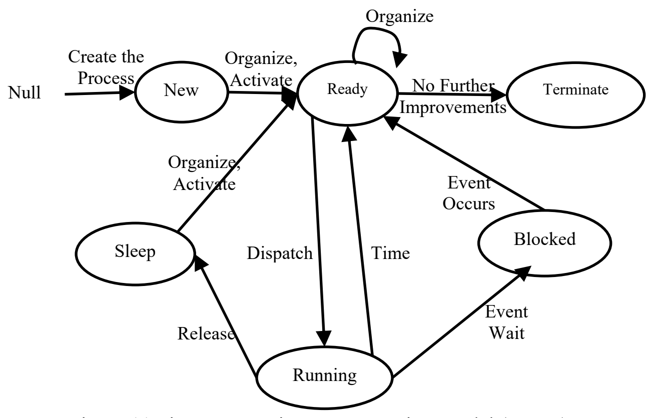

Figure 11 Six-state Continuous Processing Model (SSPM)

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 1. 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.

III. Research Method

This research was conducted to discover a new computing model to enhance the processing power of the computer. There, the characteristics of the human mind was incarnated in to a new processing model exploiting the EPA. There the mind a continuous flow of thoughts [34]. The continuity of this processing is maintained by several factors such as the inputs receive through physical five sense doors (external inputs), the inputs internally generate in the mind door (internal inputs), and a set of causal relations [5]. 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 [7]. There, 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.

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 power point 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 [19]. 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 memory, which is a result of continuous processing.

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 [5] to produce the conditionally evolving smaller memory. The actions, the constituent of the transitions of the new processing model were formed by utilizing the set of tactics have been derived from a set of fifteen causal relations, namely, Object, Root, Co-Nascence, Association, Mutuality, Pre-Dominance, Presence, Support, Pre-Nascence, Proximity, Karma, Repetition, Disappearance, Post-Nascence, Karma-Result from twenty-four causal relations [5].

This model with the above characteristics and the actions has been incorporated in to a Fraction Calculator (SSPM-FC) [5], quicksort algorithm (SSPM Sorting – particularly SSPM Insertion, and SSPM-S-Equal), 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. The implementation of this SSPM embedded sorting program was based on Quicksort. The Quicksort is highly efficient algorithm for sorting and is based on partitioning the data set into two subsets. There, 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 developed in 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 SSPM-FC [5], except its Input-Content Analyzer, internal input creation and the calculation. 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 below. Further, this had the techniques such as Insert for IP1, Equal for IP4, delete for IP3, and sort for IP2.

$$IP1:\ X \subseteq Y = X$$

$$IP2:\ X \not\subseteq Y,\ Y \not\subseteq X \Rightarrow X \cap Y \neq \phi \text{ or } X \cap Y = \phi$$

$$IP3:\ Y \subseteq X = Y$$

$$IP4:\ X = Y$$

The overall hypothesis of this research has reduced to the following words.

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 applying modifications through continuous processing.

(No Performance Improvement over program

Table 1 Comparison Tables (a) Average run times for different thresholds and number of elements for parallel QS (Source: [35]), (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=1000 (ms)**	**T=5000 (ms)**	**T=50000 (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	0.038061	0.039192	0.003218
100	0.539753	0.07334	0.348982	0.243148	0.002344
1000	3.739901	1.930715	3.801648	2.393156	0.013139
10000	55.54588	170.6803	30.75075	58.83381	0.0598
25000	110.9843	235.1399	105.149	467.4121	0.336187
50000	2.85E+08	9.76E+08	260.1235	1514.95	0.542807
75000	467.8586	2011.655	449.4495	3625.402	0.597209
100000	678.7099	4018.923	719.3369	7379.43	1.139867
150000	1534.874	8671.315	1338.664	16466.74	1.436503

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

For each of the sub scenarios in testing of SSPM-Sorting, 75 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 create the module. Then, those list have been sorted before the modification and collected the time taken by each expression for the execution in Nano seconds. In the same way, after the modification, the time values have been collected for the same set of expressions in Nano seconds. This collection process has been conducted in subsequent sorting cycles. It is an important fact to remind that the entire processing model in the above mentioned programs are managed through a smaller memory with the set of tactics. Finally, all these were tested with many examples and evaluated. The next section has briefly mentioned the respective testing, results and comparisons that have been taken place.

IV. Results and Discussion

By applying paired-t test with 99% confidence level under the Testing Scenario 1 and 95% confidence interval under Testing Scenario 2, it has been able to prove that by customizing the FC with the proposed continuous processing model, helps to improve the performance of an FC, when executes over the generations. Further, it could prove that the SSPM-QES can gain improvement over subsequent execution cycles, similarly applying paired-t test with 95% confidence interval.

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. Even though, the SSPM-S-Equal shows improvement for any total number of elements in the list, the performance of the SSPM-S-Insertion depends on the number of new elements and the total number of elements in the list with compared to the previous list. Next three testing scenarios were allocated to compare the model with the parallel computing [35] (Table 1), incremental computing [36] (Table 3), Self-adjusting computing [29] (Table 4), Dynamically Tuned Library (DTL) [30] for Sorting and Sorting with Genetic Algorithmic Approach (GAA) [31] (Table 4). The respective results are summarized in each section.

A. Compare with Parallel Computing

This section compares the speedup of the SSPM-Sorting with the speedup of the parallel Quicksort [35]. (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, 1000, 10000, 25000, 50000, 75000, 100000, and 150000. Then, the time taken for each sorting has recorded in milliseconds, before and after the modifications. The obtained values are recorded as in Table 1.

B. Compare with Self-Adjusting Computing and Incremental Computing

This This section compares the SSPM sorting with the Quicksort with self-adjusting computing [29] and the incremental computing [36]. 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 2 and 3.

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 1.

Table 2 Quicksort with Self-adjusting computing [29] Vs SSPM sorting

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 3 below.

Table 3 Quicksort with ADAPTON [30] 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

C. Compare with DTL and GAA Sorting

This testing scenario compares the SSPM-Sorting with the Dynamically Tuned Library (DTL) [30] for Sorting and Sorting with Genetic Algorithmic Approach (GAA) [31] was complicated than all the tests conducted so far. The DTL research 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 lists 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 nine stdvs were {100, 1000, 10000,100000, 1000000, and 10000000}. 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), 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). There is an apparent speedup gain in the SSPM-Sorting for the lists with a standard deviation approximately less than 1000.

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) [31] in an Intel PIII Xeon computer with 512KB cache size in RedHat 7.3 Operating System as seen in Table 4.

Table 4 Comparisons of SSPM sorting with DTL[30] and GAA [31] sorting

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

V. Conclusion

The final target of this research was to develop a continuous processing model to improve the computing efficiency of the computer that leads to a new theory of computing. There, the computer memory was modeled as conditional phenomena, which enhance the efficiency of continuous processing over program execution cycles. Further, several real-world processes, which have exhibited the continuous processing and evolving nature of the human mind, have rooted the research idea for improving the computing power. And it was a fantastic idea. SSPM, the model introduced, consists of three features, such as two processes (internal and external), continuous processing, and conditionally evolving memory. Further, the processing states of the proposed processing model were new, ready, running, blocked, sleep and terminate. With these states, set of actions forms the continuation of processing. Furthermore, the new model is advanced than the incremental computing, since the new model refine the entire system through a continuous process, not only the parts related to the modified input. On the other hand, 'Repetition' and 'Classification' can be shown as the major concepts.

References

[1] M. Abd-El-Barr and H. El-Rewini, *Fundamentals of computer organization and architecture*. Hoboken, N.J: Wiley, 2005.

[2] W. J. Starke *et al.*, "The cache and memory subsystems of the IBM POWER8 processor," *IBM J. Res. Dev.*, vol. 59, no. 1, pp. 3:1-3:13, Jan. 2015.

[3] D. A. Patterson and J. L. Hennessy, *Computer organization and design*, 5th ed. oxford: Morgan Kaufmann Publishers, 2014.

[4] 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.

[5] W. A. C. Weerakoon, A. S. Karunananda, and N. G. J. Dias, "Six-state Continuous Processing Model for a New Theory of Computing," presented at the SLAAI International Conference on Artificial Intelligence, University of Moratuwa, 2019, vol. 890, pp. 32–48.

[6] W. A. C. Weerakoon, A. S. Karunananda, and N. G. J. Dias, "A tactics memory for a new theory of computing," 2013, pp. 153–158.

[7] B. Bodhi, *Comprehensive Manual of Abhidhamma: The Psychology of Buddhism (Abhidhammattha Sangaha)*. Buddhist Publication Society, 2006.

[8] W. A. C. Weerakoon, A. S. Karunananda, and N. G. J. Dias, “Conditionally evolving memory for computers,” 2015, pp. 271–271.

[9] Mihir Patkar, “Intel Core i9 vs. i7 vs. i5: Which CPU Should You Buy?,” *Technology Explained*, 06-Dec-2018. [Online]. Available: https://www.makeuseof.com/tag/intel-core-i9-vs-i7-vs-i5-cpu/.

[10] Chaim Gartenberg, “Intel announces its latest 9th Gen chips, including its ‘best gaming processor’ Core i9,” *Circuit Breaker*, 08-Oct-2018. [Online]. Available: https://www.theverge.com/2018/10/8/17950968/intel-9th-gen-core-chips-2018-desktop-processors-8-core-i9-9900k.

[11] W. Stallings, *Computer organization and architecture: designing for performance*. Upper Saddle Rive, N.J.: Pearson Prentice Hall, 2006.

[12] [12] S. A. McKee and others, “Reflections on the memory wall.,” in *Conf. Computing Frontiers*, 2004, p. 162.

[13] [13] M. J. Wooldridge, *An introduction to multiagent systems*, 2nd ed. Chichester, U.K: John Wiley & Sons, 2009.

[14] 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.

[15] J. Liu and K. C. Tsui, “Toward nature-inspired computing,” *Commun. ACM*, vol. 49, no. 10, pp. 59–64, 2006.

[16] 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.

[17] B. Bakker, “Reinforcement learning with long short-term memory,” in *Advances in neural information processing systems*, 2002, pp. 1475–1482.

[18] F. C. Bartlett, F. C. Bartlett, and W. Kintsch, *Remembering: A study in experimental and social psychology*, vol. 14. Cambridge University Press, 1995.

[19] D. L. Schacter, “Constructive memory: past and future,” *Dialogues Clin. Neurosci.*, vol. 14, no. 1, p. 7, 2012.

[20] 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.

[21] A. Baddeley, “Working memory,” *Curr. Biol.*, vol. 20, no. 4, pp. R136–R140, 2010.

[22] K. A. De Jong, *Evolutionary Computaion: A Unified Approach*. London, England: The MIT Press, Cambridge, Massachusetts, 2006.

[23] 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.

[24] Sindhwani P. and Wadhwa V., “Genetic algorithm Approach for Optimal CPU Scheduling,” *IJCST*, vol. 2, no. 2, pp. 92–95, Jun. 2011.

[25] 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.

[26] M. Carlsson, “Monads for incremental computing,” in *The seventh ACM SIGPLAN international conference on Functional programming*, 2002, pp. 26–35.

[27] 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.

[28] U. A. Acar, G. E. Blelloch, and R. Harper, *Selective memoization*, vol. 38. ACM, 2003.

[29] 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.

[30] Xiaoming Li, Maria Jesus Grzaran, and David Padua, “A dynamically Tuned Sorting Lirary,” Los Alamitos, Calif., 2004.

[31] Xiaoming Li, Maria Jesus Grzaran, and David Padua, “Optimizing sorting with genetic algorithms,” New York, NY, USA., 2005.

[32] Y. A. Liu, “Efficient computation via incremental computation,” in *Pacific-Asia Conf. on Knowledge Discovery and Data Mining*, 1998, vol. 1574, pp. 194–203.

[33] W. A. C. Weerakoon, A. S. Karunananda, and N. G. J. Dias, “New Processing Model for Operating Systems,” University of Kelaniya, 2016, p. 29.

[34] W. F. Jayasuriya, *The Psychology & Philosophy of Buddhism*. Onalaska, USA: pariyatti, 2016.

[35] 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.

[36] M. A. Hammer, K. Y. Phang, M. Hicks, and J. S. Foster, “Adapton: Composable, demand-driven incremental computation,” 2014.

[37] Weerakoon C, Karunananda A, Dias N (2019b) Formal verification of conditionally evolving memory. Int J Comput Eng Inf Technol 11(11):243–257

[38] Weerakoon, C., Karunananda, A. & Dias, N. Human-mind-inspired processing model for computing. Mind Soc 19, 237–256 (2020). https://doi.org/10.1007/s11299-020-00236-2