Computation Reduction for IRIS Process using DT-MSOF Method

Fazmah Arif Yulianto[1,2], Kuspriyanto[1], and Yudi Satria Gondokaryono[1]

[1]School of Electrical Engineering and Informatics, Institut Teknologi Bandung, Indonesia
[2]School of Computing, Telkom University, Indonesia

Abstract: Defining the process of type IRIS (Increasing Reward with Increasing Service) is one effort to get around the limitations of time available for execution. In this paper, the Double-Track Most Significant Operation First (DT-MSOF) strategy is proposed to improve system efficiency. In this strategy, the order of execution of the computing element according to their importance and the execution need only be done until the result is correct and/or acceptable. To know the performance of this strategy, two simulations are performed and an example of application is demonstrated. The simulation results show that the execution efficiency using DT-MSOF is better than DT-LSOF and DT-RSOF, measured by the percentage of executed computation. The demonstration indicates the possibility of implementation of this strategy on particular problem.

Keywords: IRIS; Double-Track Most Significant Operation First; percentage of executed computation

1. Introduction

Computation can be modeled using Input-Process-Output scheme. The input received by the system will be processed using a particular algorithm to get the desired result. Next, the output is delivered to and used by its user. In general, there are two stopping conditions in computation execution, i.e. expected result's accuracy and target response time. When the former condition is used, then the goal of the execution is to maximize the result's accuracy. On the other hand, there are some computations which their executions are constrained by response time targets. If the processing time is enough to run the computation to completion, then the output will be maximal (100% accurate). Conversely, if there is no enough time to execute all of the computation elements, then the challenge is to deliver output with the highest accuracy it can achieve at the end of the execution.

Efforts to speeding up the execution time can be grouped as 1) explorations in the processing domain and 2) explorations in the memory domain. There are some approaches in the first domain, i.e. hardware intensification (increasing the speed and capacity), hardware and software parallelization, choosing the most suitable algorithm for each particular case characteristic [1] [2], running the program speculatively (example: branch prediction techniques [3] [4], competitive parallel execution [5] [6]), and selecting the most significant variables [7]. Memoization (look up table) [8] [9] is an example of the technique being explored in the memory domain.

All of above approaches are focused on minimizing the duration to execute all part of the computation in order to get the maximum accuracy of the results. Computation is not considered finish before all of the processes are completed and all data have been processed. In other words, the result's accuracy can be determined only after the completion of the execution of all operations. On the other side, there are set of computations which its intermediate results are meaningful and satisfy its user needs. If we can develop an approach for this kind of computation to give a guidance when the computation's result is enough to fulfill the user need, then the system can stop the current execution and move to the next one.

This paper contains the initial concept and is a work in progress as an effort to construct the intended approach. The content of the rest of this paper is as follows. Section II briefly review

Received: April 28th, 2017. Accepted: Februari 25th, 2018
DOI: 10.15676/ijeei.2018.10.1.10

the IRIS (Increasing Reward with Increasing Service) process. Related works are summarized in section III. Our proposed strategy and simulations are described in section VI and V respectively.

2. IRIS Process

When the time constraints are very strict, often there is no enough time for the computations to finish its execution, i.e. run all of its operations. The challenge in these cases is to provide end result with the highest possible accuracy or has the smallest error (deviation from the actual result). Shih, Liu et al. [10] [11] and Dey et al. [12] [13] look into this phenomenon from the point of view of process scheduling problems. They proposed a process characteristic called *Increasing Reward with Increasing Service* (IRIS) and algorithm to schedules it. The value of the computation result will increases as a function of its execution time before it reaches its deadline. IRIS process scheduling aims at generating a schedule which maximizes the number of the process which its mandatory part are finished before the deadline and minimizes the number of unexecuted optional parts. In other words, the goal is to maximize the reward and minimize result's inaccuracy.

IRIS process consists of two parts, i.e. mandatory part and an optional part. For the result to be acceptable all of the operations in the mandatory part must be finished. On the other side, the execution of the optional part will increase the result's quality. The general reward function of IRIS process is as follows:

$$R(x) = \begin{cases} 0, & x < m \\ r(x), & m \leq x \leq o + m \\ r(o + m), & x > o + m \end{cases} \quad (1)$$

Where r(x) is monotonically non-decreasing on x. Execution up to any time before m (finish time of mandatory part) resulting reward value = 0. The mandatory part must finish no later than deadline d. After the execution of mandatory part, if there is enough time than the optional part will be executed (using o unit time). The optional part must also finish no later than deadline d. IRIS approach has its own difficulties in practice since it is not easy to determine which part of the process could be labeled as optional. Furthermore, it is also not trivial to predict the effect of omitting some parts of the process during execution to the whole system [14].

3. Related Works

A. Online Sensitivity Analysis

Complex industrial systems need accurate information to deliver correct and timely responses when a particular event occur in a dynamic environment, e.g. time-critical decision making. Unfortunately, existing methods cannot fulfill the needs of real-time application since they highly depend on historical data. Tavakoli et al. proposed EventTracker, a platform to do online sensitivity analysis in large scale real-time data acquisition systems [15].

The number of variable included in a computation affects the processing time. On the other side, reducing the number of the variable may cause the degradation of the quality of the result. To minimize the effect of variable overhead, sensitivity analysis could be used. To deal with event-driven real-time systems, a major change in input variable could be identified as a triggering event. Every single event or combination of events may alter the system's states. The bigger the value change in the monitoring interval (compared to its triggering threshold), the bigger the impact of the particular variable to the system's final states and labeled as an important variable. Only the important variables, i.e. have high sensitivity value, will be included in the computation to determine system's response to the occurred event. This technique eliminates input variables which have the small or negligible influence on the final result [7].

The experiment shows that Event Tracker's computation efficiency was 10% higher than entropy-based method, without any loss in result's accuracy. As a tradeoff, it takes CPU time 0.5% longer than entropy-based method. Using this method the input variables are sorted by its significance. One of the unhandled aspects of this approach is the guarantee of result's accuracy interval.

B. Real-Time Arithmetic Unit MFIBVP and Variable Precision Processor

In the hardware domain, Schulte and Swartzlander proposed a hardware design, arithmetic algorithm, and supporting software for a new processor based on variable-precision calculation technique and interval arithmetic. This processor allows detection and correction (if necessary) of implicit error in a finite precision numerical calculation [16]. Mora-mora et al. also proposed another arithmetic unit design which combines the most significant bit-first (MSB-First) calculation technique with the variable-precision technique [17].

By joining four concepts, i.e. MSB-First arithmetic [18], interval arithmetic [19] [20], variable-precision arithmetic, and logarithmic calculation, Kerlooza et al. [21] [22] [23] proposed the design of MFIBVP real-time arithmetic unit. This architecture gives a better performance than a conventional calculation. The result is a consequence of the existence of intermediate results accessible throughout execution time, the boundedness of result's accuracy, and the computation result which has high accuracy from the beginning of the execution.

The combination of these techniques opens the possibility of a trade-off between the processing time and the accuracy of computation result. Using this approach, even if there is no enough time to finish the execution of an arithmetic instruction, the arithmetic unit delivers an intermediate result with the highest possible accuracy. Here are some critics on MFIBVP arithmetic unit:

1. In the case of an arithmetic instruction, the significance of the bits naturally depends on its position, so it can be executed in the order of its position (most significant bit-first). The same pattern does not apply to computation function in general since there is no given sign in each computation element showing its significance.
2. Determining the value of time boundary per arithmetic process which then inserted in the instruction format is not an easy task.
3. To fully implement the MFIBVP method, the whole computer system needs to be redesigned. It needs a modified hardware, compiler, operating system, programming language, and software development kit.

Following Kerlooza's work, Sukemi et al. [24] [25] [26] proposed a variable precision processor architecture. Adding selector and collector modules on the variable bit space adder, the length of processed variables can be chosen to suit the preferred accuracy level and given processing time.

4. Proposed Strategy

A. General description

This research is in the area of real-time systems with the soft timing requirement. The purpose of this study is to produce a strategy to be able to complete computing in the shortest possible time with optimal results for use by users. This strategy operates in the software area so it does not require any special hardware (as MFIBVP approaches) and will execute such computations so that:

1. Obtain a minimum value of $\sum_{i=1}^{n-1}(f_n(x) - f_i(x))^2 . \Delta t_i$; where $f_j(x)$ is the result obtained after the execution of the 1st to j-th computation element, n is the number of computational elements, Δt_i is the time required to execute the i-th computational element.
2. Obtain the minimum computing time for mapping F (x) to G {y}, where F (x) is a numerical function and G {y} is the symbolic set or interval value.

The proposed computational strategy is composed of three main components, namely:

1. An intermediate result that is always available during the execution of the computation.
2. The Most Significant Operation First (MSOF) technique that schedules the execution of computational elements according to the order of significance levels. This technique is inspired by scheduling theory that uses the significance value of each part as a priority determinant.

3. The execution approach uses Double Track Computation (DTC), which for each computational element will perform the calculation of lower bound and upper bound values of computational results concurrently.

B. Intermediate Results

With the concept of the intermediate result, there is a change of way of view towards the presentation of computational results. Computational results are useful not only at the end of computing, where all operations are completed. The intermediate results at the end of each phase of operation can also be declared useful so it should be recorded and can be presented to the users. How much benefit can be obtained from the results while the accuracy is not 100% depends on the characteristics and on the computation's goal.

C. Most Significant Operation First (MSOF)

As a consequence of the intermediate results at the end of each operation phase, we have hypotheses that the sequence of operation will have an impact on increasing the accuracy of the value of the intermediate results obtained. It is desirable to be able to get the increased accuracy of intermediate results as fast as possible. The use of the concept of Most Significant Operation First (MSOF) requires paying attention to the contribution of each operation to the computational final result. Operations with a more significant contribution will be executed first. It is the fastest way to obtain the results that are approaching 100% accuracy. Figure 1 illustrates the comparison of accuracy improvement for the MSOF, LSOF (Least Significant Operation First), and random sequences approach.

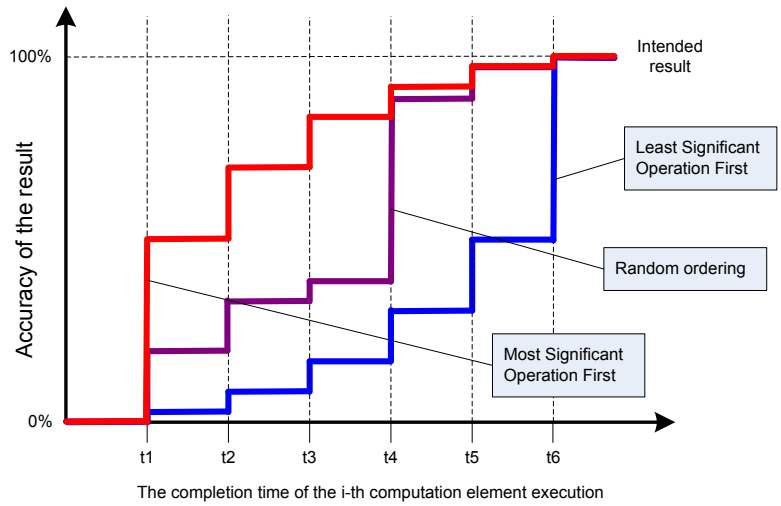


Figure 1. Comparison of accuracy improvement using MSOF, LSOF, and random ordering

There is a group of computation that the importance level of each element can be directly identified a priori from its functional structure. For another group, the importance level cannot be known a priori so it may need to be done a posteriori, for example by doing statistical analysis or involving learning process offline or online. These characteristics are then used in algorithms that are arranged to sort the execution of computational elements. We presume that MSOF is suitable for a group of computation that has the following characteristics: the structure can be partitioned and mutually excluded, the weights between the partitions are significantly different, and there are certain operating patterns.

It can be seen that with intermediate results and MSOF, some operations performed at the end have an effect on increasing accuracy that is not too large compared to other operations performed at the beginning. To shorten the computing time, these operations that have no

significant impacts need not be done. To determine which operations do not need to be done by not changing the user's perception of the computational outcome, the concept of Double-Track Computation is used.

D. Double-Track MSOF (DT-MSOF)

This concept is inspired by the theory of interval arithmetic. The concept originally used to limit calculation errors in the digital computer environment. We expanded its use to limit the range of computational results on each phase. Thus, in addition to calculating the operating results of the original function in each phase, the upper bound value and lower bound value are also calculated. The upper bound and lower bound values are used to determine whether execution needs to be continued to the next operation or the current executable result is considered sufficient to meet the needs of its users. The computational execution conditions that the result is sufficient to meet the needs of its users are referred to as the decisive condition.

If you look at the execution result of a function, then in each phase there will be two types of values, namely: the definite value of the results of all operations that have been executed and the estimated value of all operations that have not been executed. The calculation of the upper bound value for each phase involves a definite result and the maximum value of each operation that has not been executed. On the other hand, the calculation of the lower bound value for each phase involves a definite result and the minimum value of each operation that has not been executed. If the computation ends with a classification, it can be checked whether the lower and upper bound values are in the same class. Computing will be ended when the lower bound class is the same as the upper bound class. If computation does not end in classification, target error can be used (the difference of the value of lower bound and the value of upper bound) to limit the iteration/execution of computational elements. Figure 2 provides an example of the use of double-track MSOF concept in a computation case that ends with a simple classification.

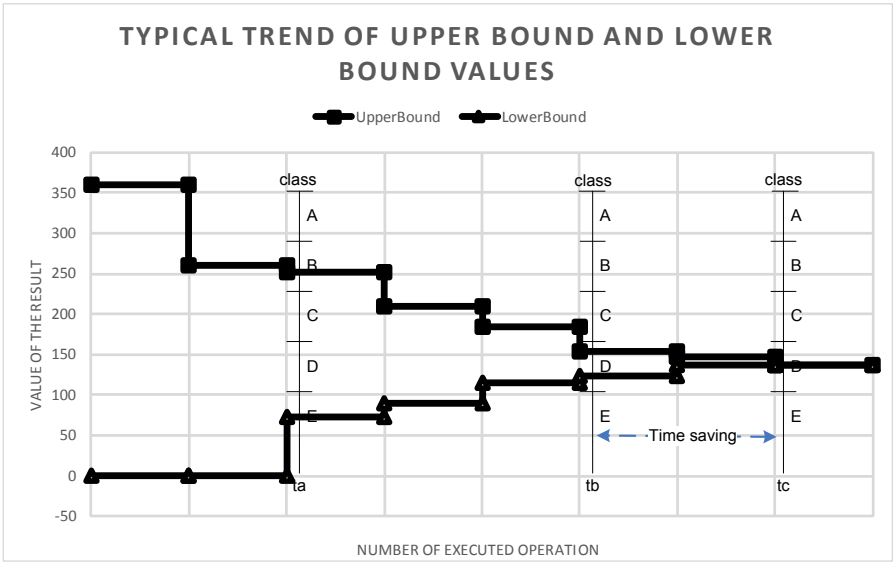


Figure 2. An example of the usage of double-track execution

It appears that the upper bound value moves monotonically down while the lower bound value moves monotonically up. At the end of the execution, both of them will have the same value as the intended result of the original function. If all of the elements of the process are done (at the time point tc), the final result will be in class D. At the time point tb, the lower bound and upper bound are both already in class D, so computation does not really need to continue because

the result is the same with the results at tc. The range between tb and tc can be viewed as a computational time-saving range.

E. How DT-MSOF works

Here is a summary of how the DT-MSOF approach works in pseudocode notation:

```
GET function (F), data (d), computation_type
IF computation_type = classification THEN
    GET criteria
ELSE
   GET target_of_error
ENDIF

Identify all computation elements Ei from F
Calculate the value of significance Si for all Ei
Sort Ei starting from the largest value of Si
upper_bound = ∑(i=1,n) max(Ei(result_maxi), Ei(result_mini))
lower_bound = ∑(i=1,n) min(Ei(result_maxi), Ei(result_mini))
i = 0; Ki = 0

IF computation_type = classification THEN
   upper_bound = classify(upper_bound, criteria)
   lower_bound = classify(lower_bound, criteria)
   WHILE upper_class <> lower_class DO
      i = i + 1 ; Ki = Ki-1 + Ei(datai)
      IF i = n THEN
           upper_bound = Ki ; lower_bound = Ki
      ELSE
           upper_bound = Ki , lower_bound = Ki
           FOR k = i+1 to n DO
              upper_bound = upper_bound + max(Ek(result_maxk),
                            Ek(result_mink))
              lower_bound = lower_bound + min(Ek(result_maxk),
                            Ek(result_mink))
      ENDIF
   ENDWHILE
   class = upper_class
   SHOW class
ELSE
   WHILE upper_bound - lower_bound > target_of_error DO
      i = i + 1 ; Ki = Ki-1 + Ei(datai)
      IF i = n THEN
         upper_bound = Ki ; lower_bound = Ki
      ELSE
         upper_bound = Ki , lower_bound = Ki
         FOR k = i+1 to n DO
            upper_bound = upper_bound + max(Ek(result_maxk),
                          Ek(result_mink))
            lower_bound = lower_bound + min(Ek(result _maxk),
                          Ek(result_mink))
      ENDIF
   ENDWHILE
   SHOW upper_bound, lower_bound
ENDIF
```

F. Performance metric

To show the performance of this approach when used in the execution of a computation, a percentage of executed computation (PEC) is used. This measure has a range of values between 0% and 100%. What is calculated is the portion of computation that needs to be executed (from the total of all computation elements) to get the results that is decisive. The smaller the value of PEC, the performance is considered the better because the computation can be completed more quickly while still able to maintain the correctness of the final result.

5. Simulation

In this section, we present simulation results for two computational cases, i.e. one whose level of significance can be known a priori, and one which must be determined a posteriori. The first type of computation example is the calculation of the Sum of Product function. The computation of the second type is represented by the process to identify dot matrix characters. As a complement of the two simulations, an application example was created to demonstrate the possibility of applying this strategy in the case of determining similarity of time series data using the Dynamic Time Warping (DTW) algorithm.

A. Case study 1: Sum of Product

The general form of Sum of Product function used is as follows:

$$Y = \sum_{i=1}^{n} coefficient_i \times data_i \tag{2}$$

The possible range of values of Y = [0..maxY] is then divided into 5 classes {A, B, C, D, E} of equal width. The simulation is performed to compare the least number of computational elements that need to be executed to achieve the decisive condition (when the correct class can be determined from a Y value and the subsequent calculations will not change the classification result). The approaches being used are the DT-LSOF (Least Significant Operation First), the DT-RSOF (Random Significant Operation First), and the DT-MSOF approach. The calculation results are presented as a percentage relative to the total amount of all computational elements (as in the case where it is fully executed).

Following is a brief explanation of the simulation. The length of the Sum of Product function (n) used in the simulation varies between 2 and 100 computational elements. The coefficient value used varies within the set $\{2^n, 3^n, ..., 10^n\}$. Data is a random number (uniformly distributed) in a range of [0 ... 100]. For each coefficient value, 10,000 repetitions are performed, comprising the process of calculating the Y value, determining the lower and upper bound values, the classification, and calculating the number of computational elements that need to be executed until a decisive condition is reached.

To calculate the values of the lower bound and upper bound, we use the formula as mentioned in paper [27] as follows:

$$f(d)_n = \begin{cases} \sum_{i=1}^{N} c_i.\underline{d_i} \ , \ n = 0 \\ \sum_{i=1}^{n} c_i.d_i \ , \ 0 < n \leq N \end{cases} \tag{3}$$

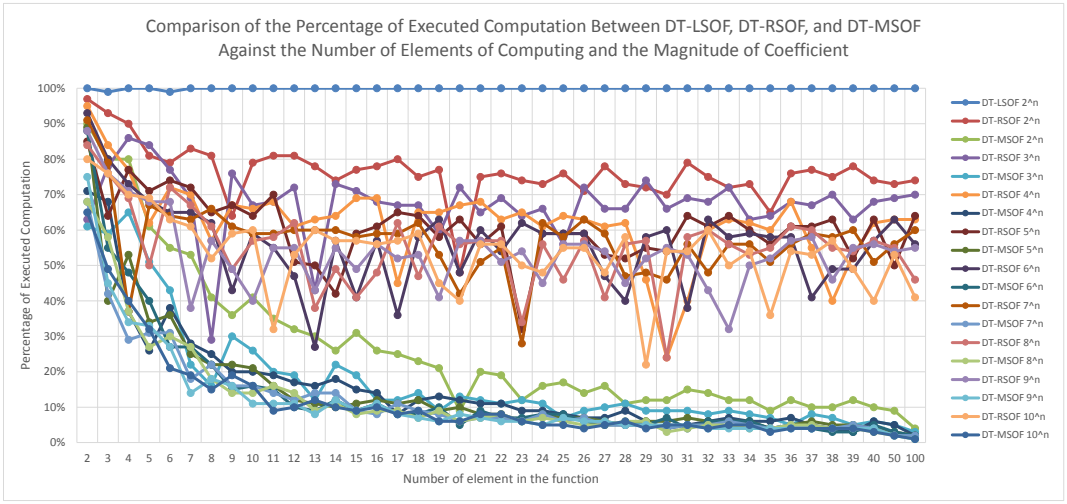
$$\underline{f}(d)_n = \begin{cases} \sum_{i=1}^{N} c_i.\underline{d_i} \ , \ n = 0 \\ f(d)_n \ , \ n = N \\ f(d)_n + \sum_{i=n+1}^{N} c_i.\underline{d_i} \ , \ else \end{cases} \tag{4}$$

$$\overline{f}(d)_n = \begin{cases} \sum_{i=1}^{N} c_i.\overline{d_i} \ , \ n = 0 \\ f(d)_n \ , \ n = N \\ f(d)_n + \sum_{i=n+1}^{N} c_i.\overline{d_i} \ , \ else \end{cases} \tag{5}$$

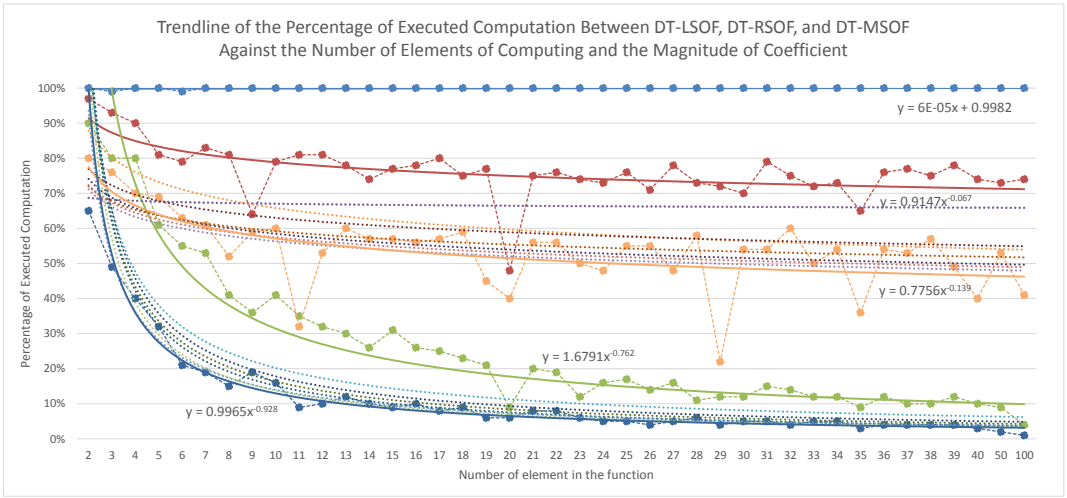
$f(d)_n$: exact value after execution of the n-th operation
$\underline{f}(d)_n$: value of the lower bound after the n-th operation
$\overline{f}(d)_n$: value of the upper bound after the n-th operation
c_i: coefficient of the i-th operation
d_i: data of the i-th operation

$\underline{d_i}$: the smallest possible value of i-th data
$\overline{d_i}$: the biggest possible value of i-th data
N: number of operation

Figure 3 gives the simulation results of the comparison of the percentage of executed computation (PEC) between DT-LSOF, DT-RSOF, and DT-MSOF. For all values of n (number of computational elements in function), DT-MSOF produces smaller PCE compared to DT-LSOF and DT-RSOF. In other words, the use of DT-MSOF results in the shortest classification time compared to the other two approaches. The DT-LSOF approach can be said to be the wrong approach if used in the case of this computation because for all n values a new decisive condition can be achieved when all computational elements have been executed.



(a)



(b)

Figure 3. Comparison (a) and trendline (b) of the PEC Between DT-LSOF, DT-RSOF, and DT-MSOF against the number of computing elements and the magnitude of coefficient

With the larger number of computational elements, it is seen that on the use of DT-RSOF there is a tendency for the PCE to go (asymptotically) around 50%, while on DT-MSOF the PCE will be closer (asymptotically) to 0%.

B. Case study 2: dot matrix character identification

If in the case of the Sum of Product function the degree of significance of the computational element can be estimated primarily from the magnitude of the coefficient value, then in the case of dot matrix character identification we cannot immediately identify processing which point has a higher significance than the other. That is why we need information about the statistical patterns associated with the input data.

The simulation is divided into two parts: 1) determining the order of process importance to the determination of the value of the threshold for each character, and 2) identification by using the statistics of the appearance of letters in the document in several languages. The simulation input data are dot matrix character images of {A, .., Z} each of 5 columns and 7 lines as shown in Figure 4. Each character image is encoded into a 35-bit sequence, each bit showing the color code at a given position in sequential order (0 for white and 1 for black). Thus, each character can be represented by a polynomial of the degree of 35.

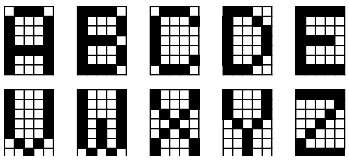


Figure 4. Examples of dot matrix characters

Bit positions are then sorted according to the level of importance. In this simulation, the sequencing is performed on the basis of the frequency of occurrence of number 1 on each bit position and is performed by ascending and descending order. For each character, a search starts from the leftmost bit (MSB) to get the least number of bits that are already unique enough to identify the character (reaching decisive condition).

At the identification stage, each input character is also converted into a 35-bit code sequence. In accordance with the sorting rules used, each polynomial element (starting at the far left) is calculated and compared to the character threshold values. If the polynomial count is within a certain range, then the character label corresponding to that range is the result of the identification and evaluation of the polynomial need not be continued. To test the capability of the identification technique, we used the frequency of the occurrence of letters in Indonesian, Malay, and English documents from Shah et al. [28]. Figure 5 shows that the use of a double-track approach provides a smaller percentage of computed computations compared to a single-track approach. Comparison of percentage of executed computation between DT-RSOF, DT-MSOF (ascending), and DT-MSOF (descending) can be seen in Figure 6.

To identify 1,000,000 letters each for Indonesian, Malay, and English, DT-MSOF generally performs better than DT-RSOF. If we focus on the comparison between DT-MSOF (ascending) and DT-MSOF (descending), it appears that their performances influenced by the language being used. DT-MSOF (ascending) is more appropriate to use in Indonesian and Malay language documents (both of which have similarities in the distribution pattern of the occurrence of letters). As for the English document, DT-MSOF (descending) is more suitable to use.

This fact confirms that the rules of ordering the computing elements affect the performance of DT-MSOF. To get the best result for the different data characteristics we may need a different sorting policy. Knowledge of input data statistics can help in determining the ordering.

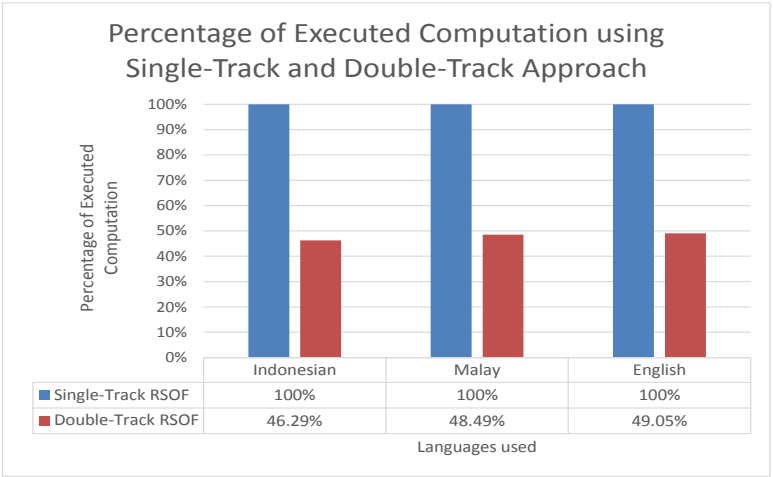


Figure 5. Comparison of PEC between single-track and dual-track computation

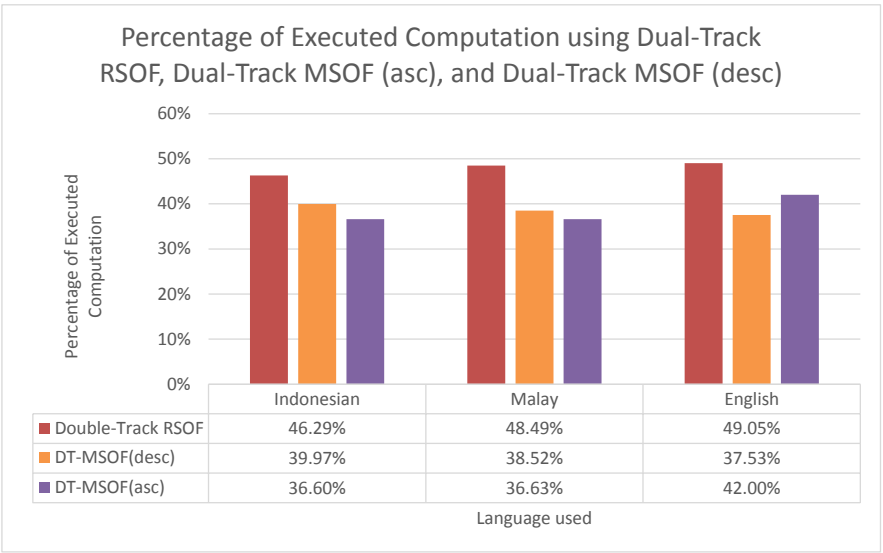


Figure 6. Comparison of PEC between DT-RSOF, DT-MSOF (ascending), and DT-MSOF (descending)

C. Application example: determining time series similarity

To demonstrate that this DT-MSOF strategy can be applied to real computational problems, an application for determining the similarity of time series data is made. The Dynamic Time Warping (DTW) algorithm is modified to become an algorithm whose behavior is consistent with the DT-MSOF strategy (called DTW*). The constants that can be specified in the DTW* algorithm are the number of significance levels and similarity threshold values. This study used the number of significance level = 16 and the value of similarity threshold in the range 5% to 95%.

As an application input, we use one of the most popular synthetic data for time series analysis, i.e. CBF (Cylinder, Bell, Funnel) data sets downloaded from http://ama.liglab.fr/~soheily/resources.html. From the data set, we selected TRAIN data instances containing 30 time series (10 Cylinder, 10 Bell, and 10 Funnel) with 128 numeric data each.

The app performs pairwise comparisons of all of the time series against all other time series by calculating DTW distance and determining its similarity based on similarity threshold value. The application records the number of DTW distance operations performed by the original DTW algorithm and the number of operation that performed by the DTW* algorithm to derive similar final results (the similarity that DTW generates is considered to be the correct reference).

The Percentage of Executed Computation of DTW* (relative to the original DTW) is shown in Figure 7. It appears that the PEC value of DTW* is always below 100%, and is in the range of [0.097%, 84.293%]. This indicates that in order to obtain the same final result, DTW* requires fewer operations than the original DTW required. In the best cases (similarity threshold value from 35% to 95%), DTW* can save 99.90% of operations or 1029.32 times less than DTW.

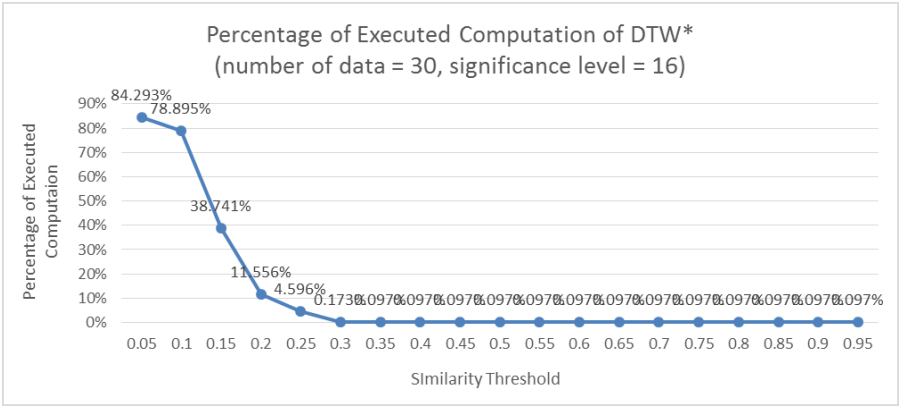


Figure 7. Percentage of Executed Computation of DTW* (relative to original DTW)

DTW is a popular algorithm and is often used in problem solving involving the determination of the similarity of two temporal sequences which may vary in speed and/or the time series classification. The DTW* algorithm proven to be more efficient, so it is possible to replace DTW in some recognition, pattern matching, and classification applications such as voice recognition, hand gesture recognition, etc.

6. Conclusion

A proposed strategy to manage the execution process using Double-Track Most Significant Operation First (DT-MSOF) approach has been presented. Simulations have been done for computational cases where the order of significant of the elements is readily available a priori and cases that must be sought a posteriori. The simulation results show that DT-MSOF gives better performance (executed computation percentage) compared to DT-LSOF and DT-RSOF strategy.

If enough time is available to complete all computing elements, the DT-MSOF strategy can save the amount of computation that needs to be executed to increase the productivity of the system as more computation can be completed over the same time duration. Conversely, if the available time is not sufficient to complete all computing elements, this strategy is expected to help provide the highest quality computational results at the end of the execution time.

7. Future Work

In order to be used more broadly, it is necessary to formulate the method of determining the order of importance of the computational element automatically and can accommodate the diversity of operational and data characteristics (especially cases of a posteriori type). It is necessary to do a simulation with computation cases within the scope of time-critical decision making in order to know how much the contribution of the DT-MSOF strategy in assisting decision-making with tight time constraints. A software prototype could be created as a means to prove the performance of DT-MSOF strategy.

8. References

[1] K. A. Smith-Miles, "Cross-Disciplinary Perspectives on Meta-Learning for Algorithm Selection," *ACM Comput. Surv.*, vol. 41, no. 1, Dec. 2008

[2] M. Gagliolo and J. Schmidhuber, "Dynamic Algorithm Portfolios," in *Proceedings of the 9th International Symposium on Artificial Intelligence and Mathematics*, 2006.

[3] W. Jin, J. Dong, K. Lu, and Y. Li, "The Study of Hierarchical Branch Prediction Architecture," in *2011 IEEE 14th International Conference on Computational Science and Engineering (CSE)*, 2011, pp. 16–20.

[4] M. Mohammadi, S. Han, T. M. Aamodt, and W. J. Dally, "On-Demand Dynamic Branch Prediction," *IEEE Comput. Archit. Lett.*, vol. 14, no. 1, pp. 50–53, Jan. 2015.

[5] O. Trachsel and T. R. Gross, "Variant-based competitive parallel execution of sequential programs," in *Proceedings of the 7th ACM international conference on Computing frontiers*, Bertinoro, Italy, 2010.

[6] O. Trachsel, "Application-level multi-variant speculation with competitive parallel execution," Doctoral Thesis, Department of Computer Science, ETH Zurich, Switzerland, 2010.

[7] S. Tavakoli, A. Mousavi, and S. Poslad, "Input variable selection in time-critical knowledge integration applications: A review, analysis, and recommendation paper," *Adv. Eng. Inform.*, vol. 27, no. 4, pp. 519–536, Oct. 2013.

[8] T. Tsumura, Y. Shibata, K. Kamimura, T. Tsumura, and Y. Nakashima, "Hinting for Auto-Memoization Processor Based on Static Binary Analysis," in *2014 Second International Symposium on Computing and Networking (CANDAR)*, 2014, pp. 426–432.

[9] A. Moreno and T. Balch, "Speeding up Large-Scale Financial Recomputation with Memoization," in *2014 Seventh Workshop on High Performance Computational Finance (WHPCF)*, 2014, pp. 17–22.

[10] J. W. S. Liu, K. J. Lin, W. K. Shih, A. C. s Yu, J. Y. Chung, and W. Zhao, "Algorithms for scheduling imprecise computations," *Computer*, vol. 24, no. 5, pp. 58–68, May 1991.

[11] W.-K. Shih and J. W. S. Liu, "Algorithms for scheduling imprecise computations with timing constraints to minimize maximum error," *IEEE Trans. Comput.*, vol. 44, no. 3, pp. 466–471, Mar. 1995

[12] J. K. Dey, J. F. Kurose, D. Towsley, C. M. Krishna, and M. Girkar, "Efficient on-line processor scheduling for a class of IRIS (increasing reward with increasing service) real-time tasks," in *SIGMETRICS '93 Proceedings of the 1993 ACM SIGMETRICS conference on Measurement and modeling of computer systems*, 1993, pp. 217–228.

[13] J. K. Dey, J. Kurose, and D. Towsley, "On-line scheduling policies for a class of IRIS (increasing reward with increasing service) real-time tasks," *IEEE Trans. Comput.*, vol. 45, no. 7, pp. 802–813, Jul. 1996

[14] Phillip A. Laplante and Seppo J. Ovaska, *Real-Time Systems Design and Analysis: Tools for the Practitioner, Fourth Edition*. Hoboken, New Jersey: Wiley-IEEE Press, 2011.

[15] S. Tavakoli, A. Mousavi, and P. Broomhead, "Event Tracking for Real-Time Unaware Sensitivity Analysis (EventTracker)," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 2, pp. 348–359, Feb. 2013.

[16] M. J. Schulte and E. E. Swartzlander, "A family of variable-precision interval arithmetic processors," *IEEE Trans. Comput.*, vol. 49, no. 5, pp. 387–397, May 2000.

[17] Higinio Mora-Mora, Jerónimo Mora-Pascual, Juan Manuel García-Chamizo, and Antonio Jimeno-Morenilla, "Real-time arithmetic unit," *Real-Time Syst.*, vol. 34, pp. 53–79, Jun. 2006.

[18] A. M. Nielsen and P. Kornerup, "MSB-First Digit Serial Arithmetic," *JUCS J. Univers. Comput. Sci.*, vol. 1, no. 7, pp. 527–547, 1995.

[19] R. E. Moore and C. T. Yang, "Interval Analysis I," Lockheed Aircraft Corporation, Missile and Space Division, Sunnyvale, California, LMSD-285875, Sep. 1959

[20] R. E. Moore, "Interval Arithmetic and Automatic Error Analysis in Digital Computing," Stanford University, Stanford, California, 25, Nov. 1962
[21] Y. Y. Kerlooza, "Unit Aritmetika Waktu-Nyata MFIBVP," Institut Teknologi Bandung, Bandung, 2010.
[22] Yusrila Y. Kerlooza, Yudi S. Gondokaryono, and Agus Mulyana, "MSB-First Interval-Bounded Variable-Precision RealTime Arithmetic Unit," *J. ICT Res. Appl.*, vol. 4, no. 1, 2010
[23] Yusrila Y. Kerlooza, Sarwono Sutikno, Yudi S. Gondokaryono, and Agus Mulyana, "THE MFIBVP REAL-TIME MULTIPLIER," in *Proceedings of the 3rd International Conference on Computing and Informatics, ICOCI 2011*, Bandung, Indonesia, 2011
[24] Sukemi, H. Sudibyo, and A. A. P. Ratna, "Priority based computation a study on paradigm shift on real time computation," in *2012 IEEE International Conference on Computational Intelligence and Cybernetics (CyberneticsCom)*, 2012, pp. 129–132.
[25] S. Sukemi, A. A. P. Ratna, and H. Sudibyo, "Variable Bitspace of Variable Precision Processor," *J. Adv. Inf. Technol.*, vol. 5, no. 2, pp. 65–70, Jan. 2014
[26] Sukemi, A. A. P. Ratna, and H. Sudibyo, "Incorporating Different Bitspaces to Create a Variable Precision Processor," *Adv. Sci. Lett.*, vol. 21, no. 1, pp. 78–82, Jan. 2015.
[27] A. Yulianto and Kuspriyanto, "A New Approach to Reducing Execution Time in Real-Time Computation," in *2017 5rd International Conference on Information and Communication Technology (ICoICT)*, Malacca, Malaysia, 2017
[28] Shah, A. Z. Saidin, I. F. Taha, A. M. Zeki, and Z. Bhatti, "Similarities and Dissimilarities between Character Frequencies of Written Text of Melayu, English, and Indonesian Languages," in *2013 International Conference on Advanced Computer Science Applications and Technologies (ACSAT)*, 2013, pp. 192–194.

Fazmah Arif Yulianto received a bachelor degree in Informatics from Sekolah Tinggi Teknologi Telkom, Bandung in 1998 and master degree in Computer Engineering from Institut Teknologi Bandung in 2004. Currently, he is a doctoral student at the School of Electrical Engineering and Informatics, Institut Teknologi Bandung. email: fazmah.arif@gmail.com

Kuspriyanto is a professor at the School of Electrical Engineering and Informatics, Institut Teknologi Bandung. He received his bachelor degree in Electrical Engineering from Institut Teknologi Bandung in 1974, D.E.A (1979) and Ph.D. (1981) in Automatic System from USTL, France. His field of interest includes Computer System, Computer Architecture, and Real Time Systems. email: kuspriyanto@yahoo.com

Yudi Satria Gondokaryono is a lecturer at the School of Electrical Engineering and Informatics, Institut Teknologi Bandung. He received his bachelor degree in Electrical Engineering from Institut Teknologi Bandung, Magister and Ph.D. in Electrical Engineering from New Mexico State University. His field of interest includes Human Computer Interaction, Real-Time, and Embedded Systems. email: ygondokaryono@gmail.com