International Journal of Electrical and Computer Engineering (IJECE)
Vol. 14, No. 4, August 2024, pp. 4214~4227
ISSN: 2088-8708, DOI: 10.11591/ijece.v14i4.pp4214-4227

Accelerating real-time deterministic discovery through single instruction multiple data graphical processor unit for executing distributed event logs

Hermawan Fauzan[1,2], Riyanarto Sarno[3], Ahmad Saikhu[3]
[1]Department of Informatics, Faculty of Engineering, Universitas Trunojoyo Madura, Bangkalan, Indonesia
[2]Doctoral Student at Department of Informatics, Faculty of Intelligent Electrical and Informatics Technology, Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia
[3]Department of Informatics, Faculty of Intelligent Electrical and Informatics Technology, Institut Teknologi Sepuluh Nopember, Surabaya, Indonesia

Article Info

Article history:

Received Feb 9, 2024
Revised Mar 8, 2024
Accepted Mar 15, 2024

Keywords:

General programming
Graphical processing units
Graphical processor unit
Multi instruction multiple data
Multi instruction single data
Process discovery
Single instruction multiple data

ABSTRACT

With the rapid expansion of process mining implementation in global enterprises distributed across numerous branches, there is a critical requirement to develop an application qualified for real-time operation with fast and precise data integration. To address this challenge, computational parallelism emerges as a feasible solution to accelerate data analytics, with graphical processor unit (GPU) computing currently trending for achieving parallelism acceleration. In this study, we developed a process mining application to optimize parallel and distributed process discovery through a combination of central processing unit (CPU) and GPU computing. The use of this computing combination is leveraged for executing multi-windowing threads within multi-instruction, multiple data (MIMD) in the CPU for streaming distributed event logs, using multi-instruction, single data (MISD) within the CPU to deploy a large footprint pipeline to the GPU, and then utilizing single instruction, multiple data (SIMD) to execute global thread discovery within the GPU. This method significantly accelerates performance in real-time distributed discovery. By reducing branch divergence in SIMD on the global thread GPU parallelism, it outperformed local-thread CPU execution in deterministic discovery, speeding up from 10 to 40 times under specific conditions using a novel min-max flag algorithm implemented within the main steps of the process discovery.

Corresponding Author:

Riyanarto Sarno
Department of Informatics, Faculty of Intelligent Electrical and Informatics Technology, Institut Teknologi Sepuluh Nopember
Surabaya, East Java, Indonesia
Email: riyanarto@if.its.ac.id

1. INTRODUCTION

The In recent years, global business corporations have experienced significant growth, marked by a proliferation of interconnected branches online. This trend is observable across various sectors, including retail, industry, banking, and government services. For instance, in Indonesia, businesses in these sectors have established branches spanning urban, rural, and international locations. Moreover, government service networks have adopted integrated and distributed data operations seamlessly operating across different departments. To ensure effective management and supervision of global business processes across multiple branches, the support of business intelligence (BI) is essential. This support should facilitate the seamless

integration of data tabulation, evaluation, and real-time transaction monitoring, considering the distributed information systems within interconnected branches. In this context, process mining serves as a complementary technique to data mining [1]. Process mining intersects data science and process science [2], particularly in the context of process-aware information systems (PAISs) for BI purposes [3], [4]. It helps uncover and analyze underlying process flows and patterns within organizations, providing valuable insights [5] for business process analysis [6], business process monitoring [7], business process simulation [8], and workflow management systems [9], leading to decision-making systems [10].

The adoption of graphical processor unit (GPU) parallelism in data mining and machine learning aims to improve the performance of business intelligence (BI) [11] with efficient and low-cost high-performance computing (HPC) [12]. This trend is prevalent in the field of deep learning for business analytics, where GPUs extensively accelerate the training and testing processes of complex neural networks [13]. They are widely implemented in fields such as internet of things (IoT) [14], autonomous vehicles [15], robotics [16], and exascale computing [17]. The use of GPUs has resulted in notable advancements in both accuracy and speed across diverse BI applications, highlighting their significant impact and versatility in GPU-accelerated computing, driving progress in computational research and data-driven disciplines [14].

In the realm of GPU implementation for fundamental computational tasks like matrix multiplication, GPU acceleration performance can achieve a speed-up twice as fast compared to using central processing units (CPUs) alone [18]. This heightened performance has been observed in diverse implementations of data mining algorithms operating on vectors, such as k-nearest neighbor (KNN) [19], association rule (AR) [20], and naïve Bayes [21], showcasing noteworthy speed-up enhancements. Current trends in machine learning, especially those rooted in deep learning [22], involve the widespread utilization of GPU-based libraries like TensorFlow [23], MATLAB [24], and Google Colab's cloud [25]. These libraries have substantiated their ability to amplify speed-up during the training and testing processes, establishing them as pivotal tools in the domains of data mining and image processing.

Furthermore, the implementation of general-purpose GPU (GPGPU) computing offers numerous advantages, including fast parallel calculations, high data throughput, and extensive memory bandwidth. Additionally, GPGPU is supported by programming interfaces in multiple programming languages [12], [26]. However, when it comes to process mining, specific limitations hinder its efficient implementation due to the unique characteristics of process mining and GPU computing. These limitations include challenges such as branch divergence caused by loop truncation in branching conditions, difficulties in managing memory access patterns (especially with sparse matrices), and issues related to thread synchronization occupancy and dependencies on synchronous work in cluster stream processors [26].

The limitations of GPU parallelism in process mining, particularly in process discovery, are largely impacted by the constrained size of matrices and concurrency patterns [27]. The effectiveness of GPU parallelism is most evident when dealing with a significant number of event activities, often necessitating the use of dummy event logs containing thousands to millions of activities [28], [29]. However, finding such extensive event logs in real-life business processes can be challenging since business processes are tailored to specific scopes and goals.

Considering the less effective process mining in managing real and static event log-ins [30], its impact on fully automated process mining for business activity monitoring from the value stream in the information system [31] is noteworthy. Enhancing GPU acceleration efficiency in real-time process mining within a multi-branch distributed system [32] provides advantages in generating a massive amount of diverse data with very high speed, termed as big data [33]. This approach allows for the effective aggregation of a large matrix for parallel CPU and GPU thread execution using multiple instruction multiple data (MIMD) and multi instruction single data (SIMD) strategy [34]. Additionally, algorithm reconfiguration is required to minimize conditional branching, memory latency, and thread synchronization [16], [35].

Parallelism characteristics in process mining utilizing GPUs emphasize three main categories through the incorporation of big data features, specifically volume, variety, and velocity. Volume denotes the capacity to tabulate data sources concurrently; variety encompasses the multitude of variations in trace event logs that can undergo parallel testing, and velocity quantifies the speed at which data flow can be processed in a parallel stream. These characteristics assess the effectiveness of executing MIMD and multiple instructions single data (MISD) using CPUs in comparison to single instruction multiple data (SIMD) using GPUs and various combinations of these architectures.

Research endeavors focused on the advancement of parallel deterministic algorithms have showcased proficiency in executing multi-thread parallelism within a multi-windowing model through GPU acceleration [36]. The utilization of GPGPU parallel computation is integral in deterministic discovery, encompassing pre-discovery processes such as footprint arrangement, basic footprint causality pattern analysis, parallel reduction, and the determination of maximum paths [37]. The assessment of the optimal method for algorithm implementation involves two scenarios: independence and aggregation testing. In the independent scenario, employing the MIMD execution strategy, all stages of the mining process are executed

within local threads on both the CPU and GPU [38]. In contrast, the aggregation of data transformation scenario combines the use of MIMD for streaming data and MISD for constructing footprints in local CPU threads, while also finally leveraging SIMD in global GPU threads. The dual-scenario testing aims to evaluate the parallelism performance of both the CPU and GPU, with a specific focus on the discovery process within deterministic algorithms [39]–[41].

By employing Java Aparapi as an open GPGPU library, the testing results revealed substantial performance enhancement when utilizing the global thread GPU operating in SIMD compared to MIMD. The SIMD implementation demonstrated acceleration ranging from 2 to 40 times, contingent on the number of threads and the volume of footprints. Notably, higher thread counts and larger footprint volumes led to greater speedup. Specifically, the combination proved particularly effective in achieving acceleration when executing over 100 threads for more than 100 events of footprints. These findings underscore the effectiveness of employing SIMD implementation for real-time process mining within a multi-branch distributed system. Through harnessing the capabilities of the global thread GPU and optimizing thread and volume configurations, significant acceleration can be realized in the discovery process.

Through the utilization of the global thread GPU in SIMD, we have achieved optimal parallel execution of footprints across multiple threads, leveraging the potent capabilities of the GPU stream processor. This parallelism has notably improved efficiency in balancing the workload between the CPU and GPU, resulting in accelerated computations compared to relying solely on the CPU. These findings underscore the substantial potential of GPGPU computing, particularly in the realm of deterministic discovery involving large-scale datasets.

2. PROPOSED METHOD

Through a systematic review of 450 papers spanning a 14-year research period, key techniques for enhancing GPU parallelism performance are highlighted. These techniques include memory coalesced access, the use of dedicated memories, reducing branch divergence, and autotuning. Memory coalesced access optimizes the utilization of global memory bandwidth by organizing vectorized matrix indices. Dedicated memories aim to synchronize random access memory (RAM), registers, local memory, and global memory to optimize the kernel's lifecycle and enable optimal operation between the CPU and GPU. Reducing branch divergence involves modifying algorithms to ensure convergence in program branching paths, facilitating the simultaneous execution of parallel threads. Autotuning involves hardware and software tuning during GPU installation and firmware configuration [26].

In the field of process mining, limited research has been conducted on GPU acceleration to enhance performance, specifically in the discovery process. Kundra conducted a study utilizing the implicit parallelism provided by the parallel computing toolbox (PCT) in MATLAB to execute deterministic discovery stages such as tuple formation, footprint assembly, and maximum path determination. The test results demonstrated that GPU parallelism accelerated the discovery process 39 times faster than CPU parallelism, achieving a maximum acceleration of only 10 times [28]. Another study by Santos [29] explored the application of GPU-accelerated control-flow algorithms, resulting in acceleration up to 8 times faster when testing large event logs containing 10,000,000 stream data. However, for smaller datasets with fewer than 10,000 event logs, no acceleration improvement was observed, and parallel GPU threads tended to be slower compared to the CPU [18].

Improving process discovery performance relies not only on parallel computing acceleration mechanisms but also on the event log streaming process. Typically, a discovery resource is performed on static event logs. For minimizing time and space complexity, event log execution can be managed using streaming windowing methods, as exemplified in Burattin's work [30]. Therefore, in this research, we contribute to improving the parallelization performance of process mining, starting from the preprocessing stage of event-log streaming from distributed resources to the primary process in the discovery phase.

After analyzing the studies conducted by Kundra *et al.* [28] and Santos [29] on GPU acceleration in process mining, notable differences in their results become evident. Santos's study exhibits realistic outcomes and a clear methodology, although it does not demonstrate excessively high acceleration in the discovery process. On the other hand, Kundra's study showcases significant acceleration results without providing specific details on the methods employed. There is considerable skepticism regarding the direct parallelization of a sequential algorithm with concurrency content in the process discovery to achieve a substantial increase in performance speed-up. Nevertheless, the execution of parallel programs on GPUs is particularly prone to various limitations, especially concerning branch divergence.

Both studies, however, did not identify the dominant factors influencing the performance of GPU parallelism in process discovery. Therefore, the objective of this research is to explore optimization techniques that can enhance GPU parallelism performance, with a specific focus on three key factors: the

influence of threading, memory coalesced access, and branch divergence. By addressing these issues, we aim to improve the efficiency and effectiveness of GPU acceleration in process mining, particularly in the discovery process.

Based on previous research, it has been observed that achieving performance improvement in process discovery on GPU requires a notably large execution footprint, a challenge often encountered in practical field implementations [28], [29]. In this study, simulations were conducted on a distributed system using a real-live event-log from BPI to illustrate the effectiveness of accessing substantial event-log footprints that align with real-world field requirements. The implementation on the distributed system involves threading on the CPU using MIMD mechanisms, facilitating the streaming of data from the distributed network, and forming event-log data windows to generate large footprints through MISD processes. Subsequently, the data is directed for execution in SIMD GPU parallelization as figured in Figure 1. To obtain the best threading mechanism, we used three combinations of threading scenarios in the testing, namely: i) combination of MIMD CPU threading and MIMD GPU for sequential algorithm, ii) combination of MIMD CPU threading and MIMD GPU for parallel algorithm, iii) MIMD CPU threading and SIMD GPU for sequential algorithm, and iv) MIMD CPU threading and SIMD GPU for parallel algorithm.

Through these three threading combinations, speed measurements for the discovery process were conducted to assess the performance acceleration of parallel computing in a distributed system. Regarding the parallelization algorithm for discovery on SIMD GPU, there are three optimized stages in the discovery process: footprint construction, parallel reduction, and maximum path optimization. In the testing scenario, the data utilized consists of both real-live and artificial event logs collected by the Business Process Institute, serving as the material for annual testing and contests. From the various event log variants used, they are classified based on two perspectives:

- Event-log file volume: Indicates the number of event traces within the event log. This perspective is employed to measure the performance in the formation of causality matrix footprints.
- Number of activities: Represents the size of the footprint matrix dimension, implying the complexity of concurrency in causality tuples. This perspective is utilized to measure the performance in parallel reduction and maximum path aspects.

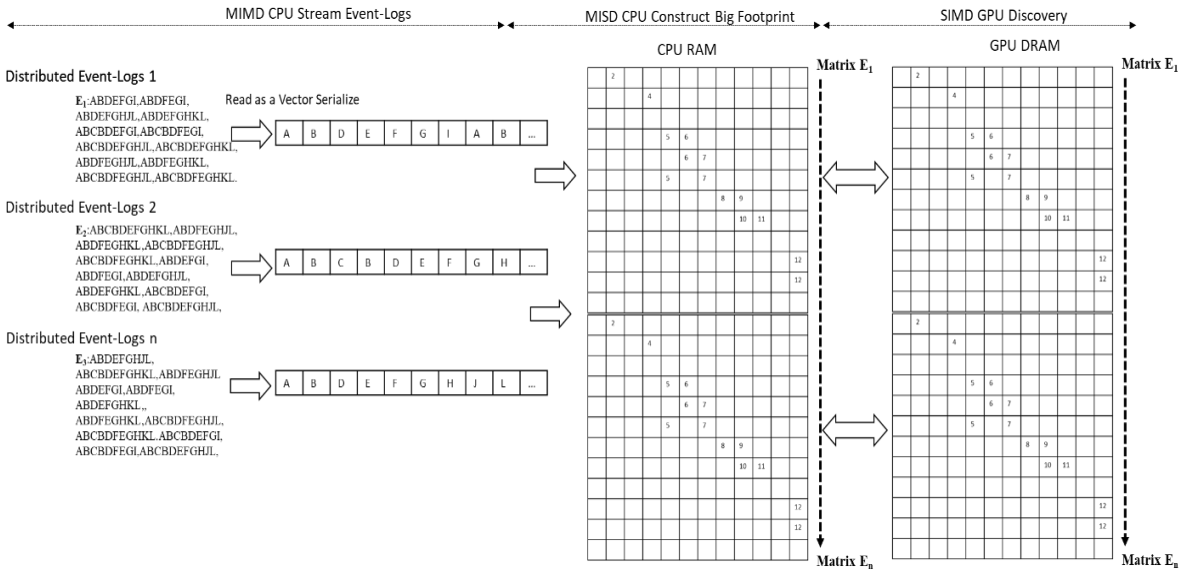


Figure 1. Data transformation scenario with combination of MIMD, MISD, and SIMD

3. METHOD

In facilitating the development of real-time discovery processing, our method has specifically focused on the pre-discovery and discovery stages. We have devised multiple algorithmic approaches that utilize multi-windowing threading to compare the efficient execution of streaming footprints on both the CPU and GPU. During the pre-discovery phase, our approach involves reading tuples from a distributed event log stream. This initial step leads us to construct a footprint that serves as a foundation for subsequent

processes. In the discovery phase, we undertake several essential steps to extract valuable insights from the data. These steps include assessing the causality of the footprint, performing parallel reduction, and determining the maximum path. By combining the computing power of the CPU and GPU using the MIMD and SIMD strategy, we aim to optimize the efficiency and speed of the real-time discovery process in a distributed system.

3.1. Contextual model

In our real-time multi-window system design, we have developed a model for the real-time discovery system, as illustrated in Figure 2 of the data flow diagram (DFD). The primary data source for the process discovery is the event-log, which consists of activity transactions recorded in the information system's data logger. To conduct experimentation and testing, we obtained event-logs from real-data collections provided by the business process institute. Additionally, we created dummy event-logs by duplicating the existing data and distributing them across multiple client computers connected to the local network. Accessing the event-logs in various formats, such as MXML, XES, and CSV, was made possible through the HTTP port. It was observed that the CSV format had approximately 75% smaller file size compared to the other formats, primarily due to the replacement of various XML tags with punctuation. This reduction in size makes CSV files more efficient for streaming event-logs.

By utilizing CSV files and buffer streaming, sequential data reading to form the footprint matrix does not require a large memory consumption. It operates through a read-release operation, where the memory is released to the Garbage Collector after being read, making it available for reuse. This approach allows for continuous real-time data streaming. For the multi-windowing process, parallel CPU thread execution is executed using MIMD local threads. To maintain data consistency and the functionality of local threads, objects are constructed as represented in the class diagram shown in Figure 3. Class *Schedule* is used to schedule the streaming process, executing the *Stream* class, thereby ensuring that the *Ithread* loop thread runs in a scheduled manner. Class *Ithread* collects the aggregation of footprints based on the principles of the factory design pattern.

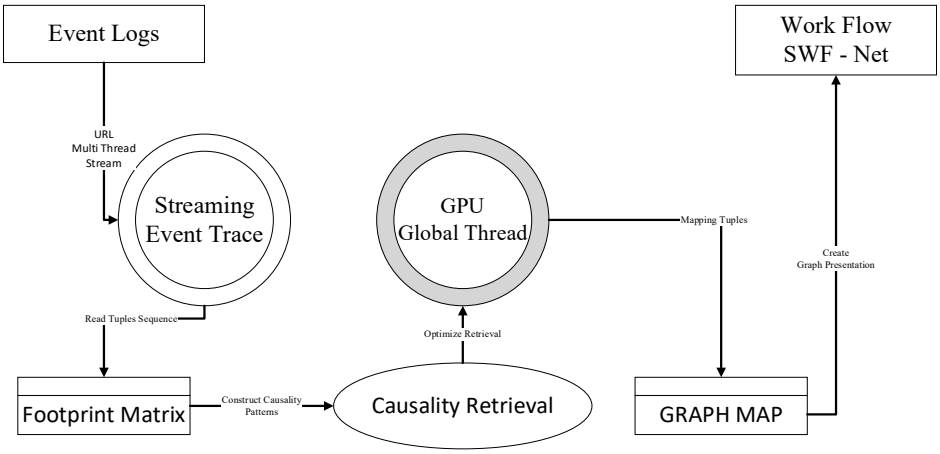


Figure 2. Contextual diagram model for real time discovery stream

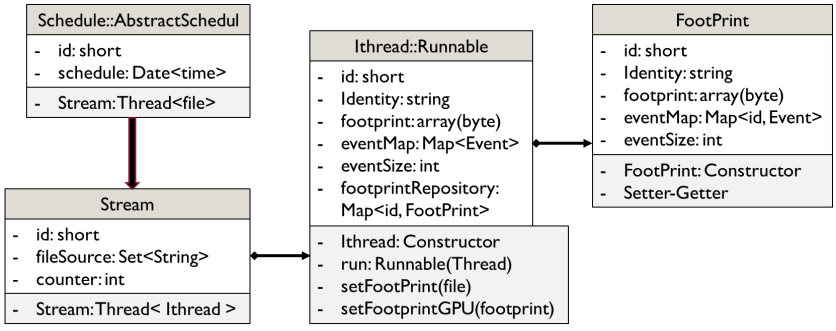
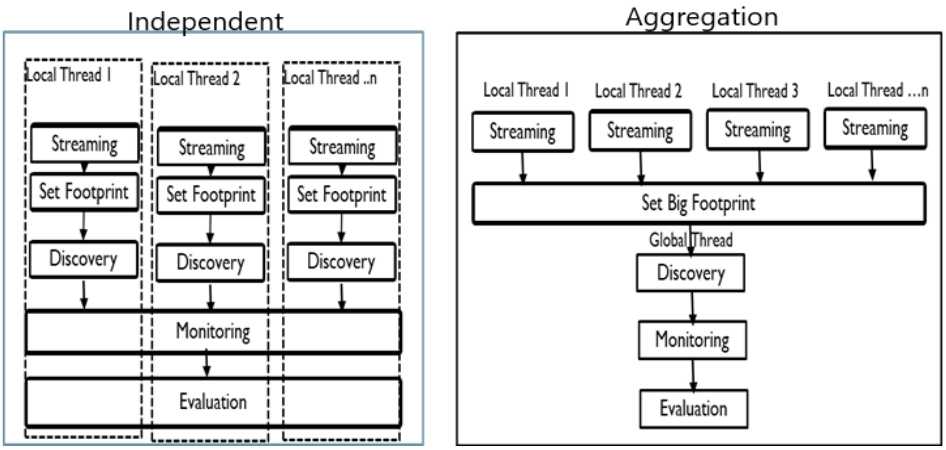


Figure 3. Class diagram model for real time discovery stream

3.2. Parallelism strategy

Two strategies are deployed for their research: independent and aggregation. As shown in Figure 4(a), by employing the independent multi-window strategy, the entire process, including pre-discovery, discovery, post-discovery, and monitoring and evaluation, is performed in a MIMD strategy approach. On the other hand, aggregation is utilized to optimize the effective use of the GPU for processing large datasets by tabulating footprints within a big-footprint matrix and then executing the discovery process using global threads through a SIMD approach as shown in Figure 4(b).

By employing the MIMD strategy, local-thread processes are executed on the static local memory of the GPU's processor registers. The maximum number of parallel threads is heavily influenced by the number of processors, clock speed, and memory bandwidth according to the specifications of the GPU being used. Table 1 is a memory and processor specifications of the NVIDIA GPU utilized for the testing phase. The hardware specifications exclusively employ low to medium specifications for testing, aiming to demonstrate that the algorithm used in the balanced specifications can effectively compare multithread performance on both CPU and GPU in a proportional manner for executing different strategy on MIMD, MISD, and SIMD approach.



(a) (b)

Figure 4. Two strategies parallelism MIMD and SIMD: (a) independent local threads windowing in MIMD strategy for overall stages and (b) aggregation local thread MIMD to MISD use CPU and SIMD use GPU

Table 1. CPU-GPU specification for testing

Tipe	Cuda core	Memori	Clock speed	Bandwidth
CPU Core I7	4 core 8 threads	16 Gb	3800 MHz	DDR3 2800 MHz
GTX 960	1024	2 Gb	1300 Mhz	7 Gbps
GTX 1080	2800	8 Gb	2000 Mhz	10 Gbps

3.3. Pre-discovery

In the independent multi-window pattern as Figure 4(a), each thread operates locally, optimizing the performance of the multi-threaded CPU using the multi-instruction multi data (MIMD) mechanism, the thread loop algorithm as shown in Algorithm 1. The results of each process are stored in a Map Footprint, allowing them to be independently utilized in subsequent stages of the process. MIMD data streamer is applied to create a collection of multiple footprint matrices $\forall M_l$ has size t^2, which are executed using pseudocode within Algorithm 2. Afterwards, the file parsing is performed using pseudocode Algorithm 3 to generate a collection of footprint matrices $F_{\forall M_L}$.

$$F_{\forall M_l} = \sum_{L=0}^{n-1} \sum_{t=0}^{i-1} \forall_L M_{[t_a,t_b]} \leftarrow b + 1$$

where $F_{\forall M_l}$ is collection map of matrix footprint from event log; M is footprint matrix; L is event log; n is numbers of event log; t is task activity, encode $\{A...L\} \rightarrow \{0...11\}$; i is numbers of activity; and a, b is tuple of task, follow tuple $a \rightarrow b$.

On the other hand, Figure 4(b) shows the aggregation pattern use the big footprint $\wedge M_l$ has same updated within the Algorithm 3. This aggregation pattern address for efficient processing in large volumes of data. In this case, data tabulation in the large matrix follows the multi-instruction single data (MISD) mechanism, which enables multiple thread instructions to be executed on the single footprint simultaneously by local thread in Algorithm 2.

As for the big-footprint $F_{\wedge M_L}$ has size $n * t^2$, where:

$$F_{\wedge M_l} = \sum_{L=0}^{n-1} \sum_{t=0}^{i-1} M_{[L*i+t_a][\,t_b]} \leftarrow b + 1$$

Algorithm 1. Stream thread

```
Input
     M_l: matrix of byte [Number of event logs*Number of tasks] [Number of tasks]
Void Stream
     R : Thread local
     Uri: Set<String> of URL
     For uri: Uri
            R= new Ithread(counterId, uri, M_l)
           Start Thread Local (R)
             counterId++
```

Algorithm 2. Discovery thread

```
Class Ithread implementation of runnable thread
     R_local: create new thread local
     F: footprint
      Counter : int of thread counter
constructor Ithread (id, uri, footprint)
     Set F
Void run thread
     stream footprint
     f = create Footprint (id, uri, footprint)
     R_local set thread setFootprint
     Uri: Set<String> of URL
     If R_local<>Null
            Counter++
            Discovery MIMD for f ∈ F_∀M_l
            Update F as big foot print
      If Counter% Size of Uri
             Discovery SIMD for f ∈ F_∧M_l
```

By directly working with large-scale data, the computation of the discovery process can be significantly accelerated within parallelism, especially using GPU, through footprint vectorization. In various discovery process methods, especially deterministic approaches, the majority of computational resources are consumed during the pre-discovery stage, which involves streaming and constructing footprints. This is because the retrieval of causality patterns during the discovery phase involves low computational complexity and executes limited matrix dimension.

Algorithm 3. Footprint writer

```
Void setFootprint (id, url)
     a: String as predecessor task
     b: String as successor tak
     τ: String of event log trace
     reader: input stream buffer
     buffer: tokenize reader from event log
     While line of reader <> NUll
          Set of tasks ← tokenize line
          If length of task > 0
             τ = task [0]
             a = task [1]
             If buffer = τ
                row = id* length of task+code number of a
                col  = code number of b
                Set Footprint F[row][col] = a+1
                a = b
                buffer = τ
      close reader
```

3.4. Discovery

The discovery process is performing within the graphical processor unit (GPU). As an initial step, the construction of the basic causal logic matrix is performed, consisting of the Input (In), Output (Out), Oneloop ($\diamondsuit$), Twoloop (Δ), and Parallel ($||$) matrices. All these features serve to construct a pattern of deterministic for achieving the SWF-network gateway boundary. The execution of MIMD on CPU and CPU-GPU forms a multithreaded local single windowing that directly executes the discovery on the collection matrix of footprints $F_{\forall M_L}$.

Since the size of the footprint matrix is determined by the limited number of tasks, sequential kernels can be executed directly. In this kernel, the determination of the maximum path is done by subtracting the content of tuple $Out\ [i][j]$ that has a smaller index from the maximum index $In[i][k]$, and vice versa. As shown in Algorithm 4, this kernel has a computational complexity on $O(n^3)$ with branch divergence caused by one of the loops inside a branching condition.

In CPU MIMD parallelism, where the computing units of the processor can perform independent multithreading, the impact of branch divergence is minimal. However, in GPU MIMD parallelism, which is formed by a cluster of stream processors with limited logic capabilities, the synchronization of warp threads on the stream processor causes significant delays for synchronization. The determination of the maximum path uses MIMD as shown in pseudocode Algorithm 4.

Algorithm 4. Find maximum path MIMD

```
Input
     In, Out: matrix footprint reduction
     row: matrix row dimension
     col: matrix column dimension

Void MaxPathLocal
     For index of i<row
        For index of j<column
             If In[i][j]>0
                for index k=j+1, k<column, k++
                    If In[i][k]>0
                        if Out[j][i]>0)
                            Out[j][i]←0
                        if Out[k][i]>0)
                            Out[k][i] ←0
         Do reverse reduce Out to In
```

As for the SIMD strategy that optimizes the Global thread in single-windowing of GPU, the optimization is performed through two steps:

- Memory coalesced access, by configuring the tuple vectors into a unidirectional pattern for the *Out*, *Parallel*, and Δ footprint features by using matrix reflection transpose. In Algorithm 5, the matrix In is initialized to be equal to the $F_{\wedge M_l}$ footprint, which is copied from CPU memory to GPU memory, while Out is the transposed matrix of In. To overcome II, inverse row-column relationship checking is performed. The parallel global threads run in two-dimensional multi-windowing matrix for constructing the *Out, Parallel*, and Δ within the global GPU memory.

Algorithm 5. Transpose of matrix output

```
Input
     In: Footprint Matrix
Output

void setOutTranspose (number of columns, Matrix In, Matrix Out)
       row: GlobalId (0)
       col: GlobalId (1)
                xrow = (row/column) * column + col
                xcol = row%column;
                If In[xcol] [xrow]>0
                     Out[xrow][xcol]← xcol+1
        In[xcol][xrow]← 0
```

- Reduce branch divergence, by configuring the loop structure and branching, the loops executed within the global threads can be parallelized in asynchronous works without any loops waiting for conditional requirements. To optimize memory access speed, vectorization is applied by converting the matrix data dimensions into a one-dimensional stream vector using division and modulus operations, as shown in Algorithm 6.

Algorithm 6. Maximum path SIMD

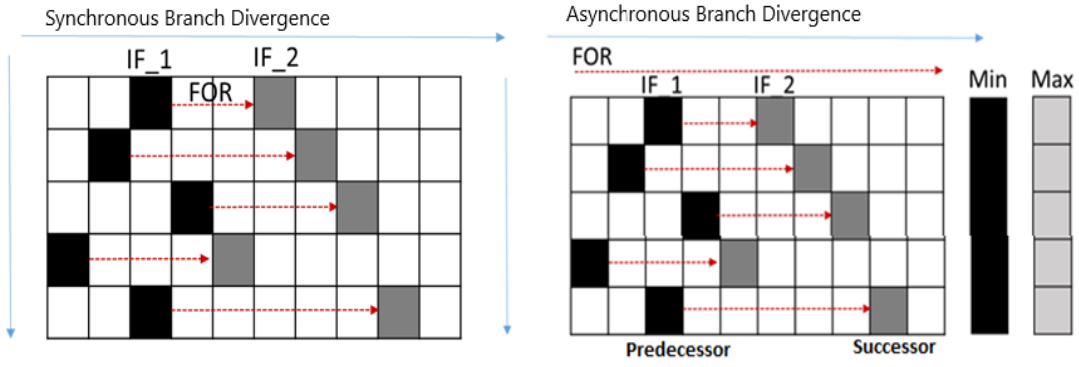
```
Input
     Min: vector array of predecessor flags
     Max: vector array of successor flags

void MaxpathOutGlobal (number row, number col, matrix In, Matrix Out)
       row: GlobalId (0)
       int m = row/col;
       int n = row % col;
       For counter i <col
                  If In[i*m+n] [i]>0
                         Max [i*m+n] = In[i*m+n] [i];
                         If Min [i*m+n] <> 0
                         If Max [i*m+n] <> Min [i*m+n]
                               If Out [Min [i*m+n]-1] [i*m+n] <> Min [i*m+n]
                                     Out [Min [i*m+n]-1] [i*m+n] ← 0
                               If Out [Max [i*m+n]-1] [i*m+n] <> Max [i*m+n]
                                     Out [Max [i*m+n]-1] [i*m+n] ← 0
                         Min [i*m+n] = Max [i*m+n];
       Do reverse reduce Out to In
```

The modification of the algorithm to reduce branch divergence in parallelism GPU over replacing the logic pattern from FOR-IF-FOR loops to FOR-FOR-IF, both in parallel reduction and maximum path determination. In the case of the pseudocode in Algorithm 6 for determining maximum-paths, when a loop is started by an IF statement, the FOR loop contained inside the IF statement cannot execute global memory warp in parallel asynchronously. So, each FOR loop will be executed on a sequential synchronously depend on IF condition, as shown in Illustration Figure 5(a).

The inefficiency of GPGPU parallelism arises from the failure of parallel global thread execution, resulting in slower execution speed on the GPU stream core compared to the CPU core for execution sequencial thread. Consequently, GPGPU utilization becomes inefficient, causing to higher time complexity of approximately $O(n^3.IF) \approx O(n^4)$. Moreover, the computational load addition with library load, data transfer between the CPU and GPU, also the allocation of matrices in GPU memory.

For reducing the high complexity caused by FOR-IF-FOR statements above, algorithm modification is necessary to shift the FOR loop within the IF statement. This adjustment reconfigures the tuple reading sequence in the matrix to become FOR-FOR-IF, allowing for asynchronous parallelism. Figure 5(b) illustrates the execution of the MIN-MAX algorithm in the parallel reduction and maximum path discovery processes.



(a) (b)

Figure 5. Pattern for reduction branch divergence (a) Synchronous Sequencial loop influenced by branch *For-If-For-If*, and (b) Asynchronous parallel loop after modification branch to *For-For-If-If*

During the execution of the processing loop $FOR(1) - FOR(2) - IF$, as $FOR(1)$ represents a global thread on the stream processor that operates asynchronously within the global memory GlobalId (0), resulting in a time complexity of $O(1)$. Then $FOR(2)$ is a vector loop that iterates through each row index, it is achieved a time complexity in $O(n)$ since there are no nested loops within the IF statement. Also, for *IF* statement evaluates the number of non-zero values in the sparse matrix, resulting in a time complexity of $O(n)$. Through the reduction of branch divergence, the algorithm effectively overcome low complexity in $O(n^2)$.

The algorithm operates in a sparse footprint matrix by iterating through each vector $\overrightarrow{V_{in[n]}} \in In_{[m,n]}$ matrice. For the initial instruction, state of flag vectors $\overrightarrow{Min_{[m]}}, \overrightarrow{Max_{[m]}} = 0$. If value of $\overrightarrow{V_{in[n]}} > 0$ that lead vector successor flag updated $\overrightarrow{Max_{[m]}} \leftarrow \overrightarrow{V_{in[n]}}$ and then reduce $Out_{[Min_{[m]},n]}$ tuple by set to zero if certain conditions is fulfilled. With the last looping order is always updating predecessor $\overrightarrow{Min_{[n]}} \leftarrow \overrightarrow{Max_{[n]}}$, after loop is incremented then find again value $\overrightarrow{V_{in[n]}} > 0$ so updated $\overrightarrow{Max_{[m]}} \leftarrow \overrightarrow{V_{in[n]}}$ that made value of successor $Max_{[m]} \neq Min_{[m]}$. The decrepancy value of successor and predecessor drives to set task tuples $Out_{[Min_{[m]},n]} = 0$ that ensuring a smaller output index is reduced. Because $Out_{[m,n]} = Transpose(In_{[m,n]})$ reducing the smaller index lead no redundancy relation between In/Out matrices also select the maximum path tuples that construct the optimum relation in split and join XOR control flow.

3.5. Post-discovery

Following the completion of the discovery process, GPGPU was employed to conduct conformance similarity testing of the discovery results using cosine similarity. Cosine similarity involves performing vector dot products and is well-suited for leveraging GPU parallelism. By utilizing the standardize operation procedure (SOP) matrix as a reference, the similarity of the discovery matrix is measured to quantify the variations in business processes. This information can then be utilized for purposes such as classification or clustering. This paper session does not delve into the post-discovery section in detail, as it is the main focus of our future paper.

4. RESULTS AND DISCUSSION

To examine the performance of SIMD and MIMD in the CPU-GPU combination, we have conducted the testing use many combinations of event logs and threads scenarios. The event logs included of real-life BPI event logs such as: credit application (12 activities), hospital billing (45 activities), BPI11 (640 activities), and a dummy event log (1,000 activities). The thread counts ranged from single-threaded to a local multi-thread count of 1,000,000. As shown in Table 2, even on short task, SIMD demonstrated significant performance acceleration when executed with numerous threads.

Table 2. Comparison of execution times for a real time of distributed discovery for credit application (12 activities)

Processor Type	1	10	100	1000	10000	50000	100000	1000000
CPU MIMD I	1	6	30	40	124	450	1400	18000
GPU 1 960 MIMD II	130	260	600	1200	∞	∞	∞	∞
GPU 1 960 MIMD III	150	200	400	800	∞	∞	∞	∞
GPU 1 960 SIMD	100	110	130	140	250	520	820	2700
GPU 2 1080 MIMD I	80	180	320	800	∞	∞	∞	∞
GPU 2 1080 MIMD II	90	120	210	520	∞	∞	∞	∞
GPU 2 1080 SIMD	70	80	100	110	140	180	270	1100

The single and low thread testing results showed that CPU discovery on MIMD I without data transfer between PC memory and GPU memory achieved the best result, with a mere 1 ms execution time. This can be attributed to the simplicity of the deterministic discovery algorithm, despite its time complexity on the CPU. However, when utilizing the GPU, there was a minimal initialization time required to load the class model and Java Native Interface (JNI) when using the OpenCL library, resulting in an execution time close to 100 ms.

For the CPU-GPU combination, thread-local multi-windowing was controlled by the multi-thread CPU using MIMD. Three combinations were used for the GPU that are: conventional MIMD II without branch divergence reduction, MIMD III with branch divergence reduction, and SIMD with branch divergence reduction. The testing revealed that MIMD II without branch divergence reduction performed the worst and quickly encountered faults, as highlighted in Table 2. The GPU stream processors have struggled with synchronization due to the thread differentiation on stream cores. In this case, serialization multi thread CPU without branch divergence reduction has become the best result for short activities compared to GPU parallelization as shown as Figure 6(a) dan 6(b). Moreover, MIMD III outcome poor performance due to the GPU stream cores had significantly lower performance compared to the CPU cores on single works, they quickly reached saturation and fault outputs were observed from the OpenCL compiler.

In contrast, SIMD demonstrated remarkably high performance due to the division of executing stages, leveraging the optimal specifications of the CPU processor cores to handle numerous threads in the

pre-discovery process, while the GPU stream processor specifically handled the discovery. The highest performance was achieved with SIMD due to optimal workload distribution and the significant impact of branch divergence reduction on large data clusters. This enabled the stream processor to maintain asynchronous parallelism. Table 3 shows that the workload approach in SIMD, as opposed to MIMD in GPU, can accelerate the discovery speed by 10 to 40 times for event-logs containing over 45 to 500 activities and thread counts ranging from 100 to 1,000. This shows that the implementation of GPU parallelism in real-live discovery is more realistic because it does not depend on the size of the activity which must be large as in previous studies [28], [29], where the result as shown as Figure 6(c) and 6(d) by using 100 activities as event-log task input, the performance of SIMD GPU can be performed to achieve significant speedup. By utilizing real-world event logs with a limited composition of event occurrences from distributed sources, it demonstrates the effectiveness of the parallel computing performance employed.

Table 3. Comparison of execution times for a real time of distributed discovery for BPI challenge 18 (100 activities)

Processor Type	1	10	100	1000	10000	50000	100000	1000000
CPU MIMD I	6	8	300	800	3200	20000	80000	∞
GPU 1 960 MIMD II	370	820	∞	∞	∞	∞	∞	∞
GPU 1 960 MIMD III	400	640	1200	∞	∞	∞	∞	∞
GPU 1 960 SIMD	170	180	200	380	620	2400	8800	∞
GPU 2 1080 MIMD I	240	570	2600	∞	∞	∞	∞	∞
GPU 2 1080 MIMD II	300	440	950	∞	∞	∞	∞	∞
GPU 2 1080 SIMD	120	140	160	190	240	780	2600	∞

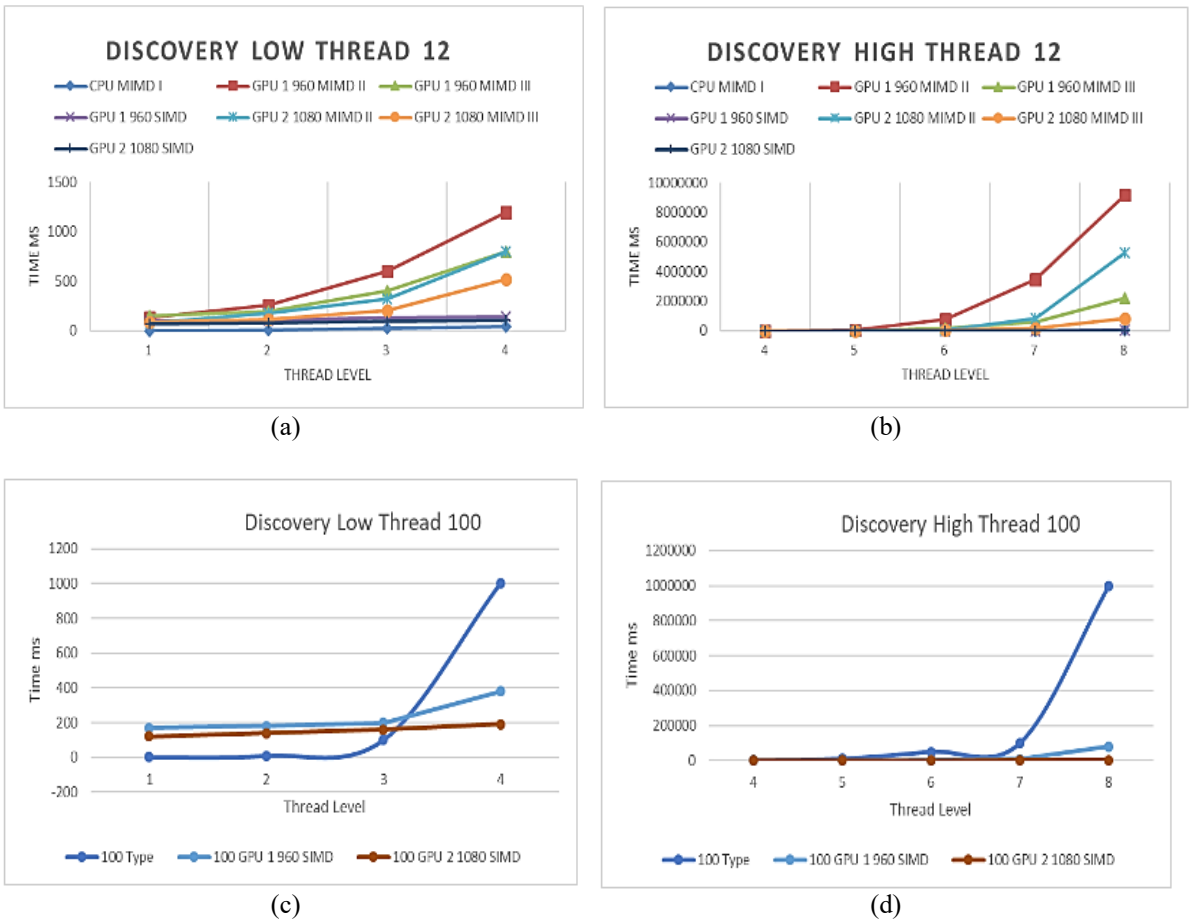


Figure 6. Comparison of discovery results between MIMD and SIMD architectures: (a) performing short tasks with low thread count, (b) performing short tasks with high thread count, (c) executing long tasks with low thread count, and (d) executing long tasks with high thread count

Regarding hardware specifications, the NVIDIA GTX 960 is old NVIDIA architecture which has a low-specification GPU with 16 compute units of stream multi-processors (SMs), experiences a bottleneck with limited acceleration power, averaging under 10 times. The number of SMs and bandwidth speed has the most significant influence on determining GPGPU performance as they coordinate the synchronization of parallelism between GPU and CPU threads. On the other hand, the NVIDIA GTX 1080, a high-performance GPU with 20 SMs, achieves significantly higher performance compared to the GTX 1080 for activation thread clusters.

5. CONCLUSION

The conducted study has determined that integrating parallelism in both the CPU and GPU, by employing local thread multi-windowing with a combination strategy using MIMD in the CPU for reading streaming data, utilizing MISD in the CPU for constructing footprints, and finally executing in global thread SIMD in the GPU within the discovery process, significantly accelerates the speed of real-time distributed discovery. These results indicate that even with a low to medium-specification PC and open-source software, high-performance outcomes can be achieved. Therefore, this approach is highly recommended as an efficient best practice. The observed high-performance results are attributed to effectively managing the convergence of branching logic and memory on the GPGPU, despite potential limitations of the GPU when operating under concurrency and sparse matrix conditions. The method of hiding branch divergence using the Min-Max flag to reduce branch divergence is a novelty in this study, proving to accelerate the performance for parallelism in GPU SIMD.

ACKNOWLEDGEMENTS

The authors gratefully to the *Lembaga Pengelolah Dana Pendidikan* (LPDP) of the Indonesian Ministry of Finance for providing financial support for this study. The authors would also give appreciation to the supervisors, reviewers, and editors for their valuable feedback, insightful comments, and suggestions, which have greatly enhanced the overall presentation of the paper. Their contributions have been instrumental in improving the quality and clarity this research.

REFERENCES

[1] W. van der Aalst, "Data science in action," in *Process Mining*, Springer Berlin Heidelberg, 2016, pp. 3–23, doi: 10.1007/978-3-662-49851-4_1.
[2] W. M. P. van der Aalst, "Process mining: A 360 degree overview," in *Lecture Notes in Business Information Processing*, vol. 448, Springer International Publishing, 2022, pp. 3–34, doi: 10.1007/978-3-031-08848-3_1.
[3] A. Adriansyah, B. F. Van Dongen, and W. M. P. Van Der Aalst, "Towards robust conformance checking," in *Lecture Notes in Business Information Processing*, vol. 66 LNBIP, Springer Berlin Heidelberg, 2011, pp. 122–133, doi: 10.1007/978-3-642-20511-8_11.
[4] A. H. M. Rashed, N. E. El-Attar, D. S. Abdelminaam, and M. Abdelfatah, "Analysis the patients' careflows using process mining," *PLoS ONE*, vol. 18, no. 2 February, p. e0281836, Feb. 2023, doi: 10.1371/journal.pone.0281836.
[5] A. Rozinat and W. M. P. Van Der Aalst, "Conformance checking of processes based on monitoring real behavior," *Information Systems*, vol. 33, no. 1, pp. 64–95, Mar. 2008, doi: 10.1016/j.is.2007.07.001.
[6] M. Jans and M. Laghmouch, "Process mining for detailed process analysis," in *Advanced Digital Auditing*, Springer International Publishing, 2023, pp. 237–256, doi: 10.1007/978-3-031-11089-4_9.
[7] C. Di Francescomarino and C. Ghidini, "Predictive process monitoring," in *Lecture Notes in Business Information Processing*, vol. 448, Springer International Publishing, 2022, pp. 320–346, doi: 10.1007/978-3-031-08848-3_10.
[8] M. Camargo, M. Dumas, and O. González-Rojas, "Automated discovery of business process simulation models from event logs," *Decision Support Systems*, vol. 134, Art. no. 113284, Jul. 2020, doi: 10.1016/j.dss.2020.113284.
[9] A. Adriansyah, N. Sidorova, and B. F. Van Dongen, "Cost-based fitness in conformance checking," in *Proceedings - International Conference on Application of Concurrency to System Design, ACSD*, Jun. 2011, pp. 57–66, doi: 10.1109/ACSD.2011.19.
[10] M. De Leoni and W. M. P. Van Der Aalst, "Data-aware process mining: Discovering decisions in processes using alignments," in *Proceedings of the ACM Symposium on Applied Computing*, Mar. 2013, pp. 1454–1461, doi: 10.1145/2480362.2480633.
[11] A. Cano, "A survey on graphic processing unit computing for large-scale data mining," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 8, no. 1, Nov. 2018, doi: 10.1002/widm.1232.
[12] L. Shi, H. Chen, J. Sun, and K. Li, "VCUDA: GPU-accelerated high-performance computing in virtual machines," *IEEE Transactions on Computers*, vol. 61, no. 6, pp. 804–816, Jun. 2012, doi: 10.1109/TC.2011.112.
[13] M. Kraus, S. Feuerriegel, and A. Oztekin, "Deep learning in business analytics and operations research: Models, applications and managerial implications," *European Journal of Operational Research*, vol. 281, no. 3, pp. 628–641, Mar. 2020, doi: 10.1016/j.ejor.2019.09.018.
[14] A. N. Sisyukov, O. S. Yulmetova, and V. A. Kuznecov, "GPU accelerated industrial data analysis in private cloud environment," in *Proceedings of the 2019 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering, ElConRus 2019*, Jan. 2019, pp. 348–352, doi: 10.1109/EIConRus.2019.8656751.
[15] A. Díaz-Toro, P. Mosquera-Ortega, G. Herrera-Silva, and S. Campaña-Bastidas, "Path planning for assisting blind people in purposeful navigation," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 26, no. 1, pp. 450–461, Apr. 2022, doi: 10.11591/ijeecs.v26.i1.pp450-461.
[16] M. S. Nguyen, T. T. Than, T. N. Do, and H. N. Nguyen, "Design of elderly-assistant mobile servant robot," *Indonesian Journal of*

Electrical Engineering and Computer Science, vol. 26, no. 3, pp. 1338–1350, Jun. 2022, doi: 10.11591/ijeecs.v26.i3.pp1338-1350.
[17] S. J. Kamble and M. R. Kounte, "Application of improved you only look once model in road traffic monitoring system," *International Journal of Electrical and Computer Engineering*, vol. 13, no. 4, pp. 4612–4622, Aug. 2023, doi: 10.11591/ijece.v13i4.pp4612-4622.
[18] R. G. Balagafshe, A. Akoushideh, and A. Shahbahrami, "Matrix-matrix multiplication on graphics processing unit platform using tiling technique," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 28, no. 2, pp. 1012–1019, Nov. 2022, doi: 10.11591/ijeecs.v28.i2.pp1012-1019.
[19] M. Bali, A. S. Pichandi, and J. H. Duraisamy, "Biomedical-named entity recognition using CUDA accelerated KNN algorithm," *Telkomnika (Telecommunication Computing Electronics and Control)*, vol. 21, no. 4, pp. 825–835, Aug. 2023, doi: 10.12928/TELKOMNIKA.v21i4.24065.
[20] N. P. More, V. B. Nikam, and B. Banerjee, "Novel approach of association rule mining for tree canopy assessment," *IAES International Journal of Artificial Intelligence*, vol. 10, no. 3, pp. 771–779, Sep. 2021, doi: 10.11591/ijai.v10.i3.pp771-779.
[21] M. Jaiswal, S. Das, and Khushboo, "Detecting spam e-mails using stop word TF-IDF and stemming algorithm with naïve Bayes classifier on the multicore GPU," *International Journal of Electrical and Computer Engineering*, vol. 11, no. 4, pp. 3168–3175, Aug. 2021, doi: 10.11591/ijece.v11i4.pp3168-3175.
[22] R. D. Darmawan, W. A. Kusuma, and H. Rahmawan, "Deep learning optimization for drug-target interaction prediction in COVID-19 using graphic processing unit," *International Journal of Electrical and Computer Engineering*, vol. 13, no. 3, pp. 3111–3123, Jun. 2023, doi: 10.11591/ijece.v13i3.pp3111-3123.
[23] K. Adam, I. I. Mohd, and Y. Ibrahim, "Analyzing the instructions vulnerability of dense convolutional network on GPUS," *International Journal of Electrical and Computer Engineering*, vol. 11, no. 5, pp. 4481–4488, Oct. 2021, doi: 10.11591/ijece.v11i5.pp4481-4488.
[24] B. K. O. C. Alwawi and A. F. Y. Althabhawee, "Towards more accurate and efficient human iris recognition model using deep learning technology," *Telkomnika (Telecommunication Computing Electronics and Control)*, vol. 20, no. 4, pp. 817–824, Aug. 2022, doi: 10.12928/TELKOMNIKA.v20i4.23759.
[25] H. Kimm, I. Paik, and H. Kimm, "Performance comparision of TPU, GPU, CPU on Google colaboratory over distributed deep learning," in *Proceedings - 2021 IEEE 14th International Symposium on Embedded Multicore/Many-Core Systems-on-Chip, MCSoC 2021*, Dec. 2021, pp. 312–319, doi: 10.1109/MCSoC51149.2021.00053.
[26] P. Hijma, S. Heldens, A. Sclocco, B. Van Werkhoven, and H. E. Bal, "Optimization techniques for GPU programming," *ACM Computing Surveys*, vol. 55, no. 11, pp. 1–81, Mar. 2023, doi: 10.1145/3570638.
[27] P. Xu, M. Y. Sun, Y. J. Gao, T. J. Du, J. M. Hu, and J. J. Zhang, "Influence of data amount, data type and implementation packages in GPU coding," *Array*, vol. 16, p. 100261, Dec. 2022, doi: 10.1016/j.array.2022.100261.
[28] D. Kundra, P. Juneja, and A. Sureka, "Vidushi: Parallel implementation of alpha miner algorithm and performance analysis on CPU and GPU architecture," in *Lecture Notes in Business Information Processing*, vol. 256, Springer International Publishing, 2016, pp. 230–241, doi: 10.1007/978-3-319-42887-1_19.
[29] R. M. M. P. dos Santos, "Parallel computing for process mining," Thesis, Tecnico Lisboa, 2016.
[30] A. Burattin, "Streaming process mining," in *Lecture Notes in Business Information Processing*, vol. 448, Springer International Publishing, 2022, pp. 349–372, doi: 10.1007/978-3-031-08848-3_11.
[31] J. Rudnitckaia, H. S. Venkatachalam, R. Essmann, T. Hruska, and A. W. Colombo, "Screening process mining and value stream techniques on industrial manufacturing processes: Process modelling and bottleneck analysis," *IEEE Access*, vol. 10, pp. 24203–24214, 2022, doi: 10.1109/ACCESS.2022.3152211.
[32] J. Wang, S. Jia, G. Wang, Z. Pan, and X. Yu, "An improved CPU–GPU parallel framework for real-time interactive cutting simulation of deformable objects," *Computers and Graphics (Pergamon)*, vol. 114, pp. 59–72, Aug. 2023, doi: 10.1016/j.cag.2023.05.013.
[33] M. M. Rathore, H. Son, A. Ahmad, A. Paul, and G. Jeon, "Real-time big data stream processing using GPU with spark over hadoop ecosystem," *International Journal of Parallel Programming*, vol. 46, no. 3, pp. 630–646, Jun. 2018, doi: 10.1007/s10766-017-0513-2.
[34] L. C. Sim, G. Leedham, L. C. Jian, and H. Schroder, "Fast solution of large N × N matrix equations in an MIMD-SIMD Hybrid System," *Parallel Computing*, vol. 29, no. 11-12 SPEC.ISS., pp. 1669–1684, Nov. 2003, doi: 10.1016/j.parco.2003.05.011.
[35] E. A. Träff, A. Rydahl, S. Karlsson, O. Sigmund, and N. Aage, "Simple and efficient GPU accelerated topology optimisation: Codes and applications," *Computer Methods in Applied Mechanics and Engineering*, vol. 410, Art. no. 116043, May 2023, doi: 10.1016/j.cma.2023.116043.
[36] X. Limón, A. Guerra-Hernández, N. Cruz-Ramírez, H. G. Acosta-Mesa, and F. Grimaldo, "A Windowing strategy for Distributed Data Mining optimized through GPUs," *Pattern Recognition Letters*, vol. 93, pp. 23–30, Jul. 2017, doi: 10.1016/j.patrec.2016.11.006.
[37] M. De Leoni, W. M. P. Van Der Aalst, and M. Dees, "A general process mining framework for correlating, predicting and clustering dynamic behavior based on event logs," *Information Systems*, vol. 56, pp. 235–257, Mar. 2016, doi: 10.1016/j.is.2015.07.003.
[38] B. Adeleye and S. M. Jiddah, "Analysis of parallel architectures: SIMD, tightly-coupled MIMD, and loosely-coupled MIMD," *International Journal of Computer Trends and Technology*, vol. 53, no. 1, pp. 6–8, Nov. 2017, doi: 10.14445/22312803/ijctt-v53p102.
[39] J. Li, D. Liu, and B. Yang, "Process mining: Extending α-algorithm to mine duplicate tasks in process logs," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 4537 LNCS, Springer Berlin Heidelberg, 2007, pp. 396–407, doi: 10.1007/978-3-540-72909-9_43.
[40] Hermawan and R. Sarno, "A more efficient deterministic algorithm in process model discovery," *International Journal of Innovative Computing, Information and Control*, vol. 14, pp. 971–995, Jun. 2018.
[41] R. Sarno, W. A. Wibowo, Kartini, Y. Amelia, and K. Rossa, "Determining process model using time-based process mining and control-flow pattern," *Telkomnika (Telecommunication Computing Electronics and Control)*, vol. 14, no. 1, pp. 349–359, Mar. 2016, doi: 10.12928/TELKOMNIKA.v14i1.3257.

BIOGRAPHIES OF AUTHORS

Hermawan Fauzan received his B.Eng. degree in electrical engineering from Brawijaya University, Indonesia, in 2002, and his M.Com. degree from Institute Technology Sepuluh Nopember Surabaya, Indonesia, in 2011. He is currently a Ph.D. candidate in Computer Science at Institute Technology Sepuluh Nopember Surabaya, Indonesia. He serves as a lecturer at Universitas Trunojoyo Madura, Indonesia. His research interesting areas, including data structures, web programming, distributed systems, retrieval engineering, internet of things, data mining, and process mining. He can be contacted via email: hermawan@trunojoyo.ac.id.

Riyanarto Sarno professor in Department of Informatics, Institut Teknologi Sepuluh Nopember, Indonesia. Get Master and Ph.D. degree from News Brunswick University, Canada. He is currently a head of the Informatics Management Intelligent Laboratory. His interests include internet of things, business process management, process aware information systems, knowledge engineering, and smart grids. He can be contacted via email: riyanarto@if.its.ac.id, riyanarto@gmail.com.

Ahmad Saikhu received Ph.D. degrees in computer science Institut Teknologi Sepuluh Nopember, Indonesia. He is currently a head office of the Department of Informatics Institut Teknologi Sepuluh Nopember, Indonesia. His interests include data mining, computer vision, and machine learning. He can be contacted via email: saikhu@if.its.ac.id.