International Journal of Engineering & Technology, 7 (4) (2018) 2446-2452

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET
doi: 10.14419/ijet.v7i4.14748
Research paper

A task-level parallelism approach for process discovery

Muktikanta Sahu[1]*, Rupjit Chakraborty[2] and Gopal Krishna Nayak[3]

[1]*International Institute of Information Technology Bhubaneswar, Odisha, India - 751003*
[2]*International Institute of Information Technology Bhubaneswar, Odisha, India - 751003*
[3]*International Institute of Information Technology Bhubaneswar, Odisha, India - 751003*
**Email: muktikanta@iiit-bh.ac.in*

Abstract

Building process models from the available data in the event logs is the primary objective of Process discovery. Alpha algorithm is one of the popular algorithms accessible for ascertaining a process model from the event logs in process mining. The steps involved in the Alpha algorithm are computationally rigorous and this problem further manifolds with the exponentially increasing event log data. In this work, we have exploited task parallelism in the Alpha algorithm for process discovery by using MPI programming model. The proposed work is based on distributed memory parallelism available in MPI programming for performance improvement. Independent and computationally intensive steps in the Alpha algorithm are identified and task parallelism is exploited. The execution time of serial as well as parallel implementation of Alpha algorithm are measured and used for calculating the extent of speedup achieved. The maximum and minimum speedups obtained are 3.97x and 3.88x respectively with an average speedup of 3.94x.

Keywords: *Alpha algorithm; MPI; Process Discovery; Process Mining; Speedup.*

1. Introduction

Process mining is a novel discipline which is derived from process modeling and analysis along with data mining. A range of tools are available in process mining which can not only be used for process improvement but also for getting data-driven insights. Both structured as well as unstructured processes can be analyzed by using process mining techniques. The possible benefits are considerable; just by examining the discovered model, important perception can be obtained. Information systems play an important role in executing business processes in many organizations.Organizations use a number of IT based business solutions to run their operational processes. Examples of such solution includes Work flow Management Systems (WMS), Customer Relationship Management (CRM) Systems and Enterprise Resource Planning (ERP). In many cases these information systems may not be aware of the process model in which these are a part and in such cases process discovery becomes important [1], [2]. Process discovery, conformance checking and further improvements of real processes are the primary tasks that are carried out by process mining techniques by extracting knowledge from the available event logs in information system [3].

Improving performance of computationally exhaustive process mining algorithms is challenging as it requires efficient processing of the event log data which is increasing exponentially. It becomes computationally intensive and time consuming task for process discovery algorithms to work on ever increasing event logs. Thus, there is a need to make the process discovery algorithms efficient enough to handle the rapidly growing event logs through parallelism.

The Alpha algorithm [4] is one of the important algorithms in Process Mining which explores process models from event logs. The proposed research work is motivated by the need to investigate the efficiency and scalability of Alpha algorithm in a parallel computing environment by using Message Passing Interface programming [5]. The objective is to accelerate Alpha algorithm with the help of different parallelisms provided by MPI.

The remainder of this paper is organized as follows: A brief general idea of process mining along with the works that have been done in the area of process discovery has been presented in the literature review section. We then present a general description of MPI constructs for parallel processing. In the following section a parallel execution framework for the Alpha algorithm has been proposed by using MPI implementations with a meticulous depiction of the required parallel programming paradigm constructs. Execution results of the experiments are presented for both the serial version as well as the parallel version of the Alpha algorithm to demonstrate performance and scalability. Result discussion and analysis of the same is presented before the conclusion with a glance to further scope of improvement.

2. Literature Review

Process discovery involves a combination of machine learning and data mining techniques. Description of process discovery algorithms along with their respective notations can be found in [4], [6], [7], [8], [9], [10],[11].

Process discovery algorithms may be categorized as *local* or *global* approaches [12]. *local* approaches involve building process models from the relations that exist between activities in event logs, e.g., Alpha [4], Alpha++ [7], and Heuristic Miner [6]. But, *global* approaches begin with a full model and try to improve those models in successive phases later on. Genetic [13], [14] and region mining[9],

[10] are examples of *global* approaches.

Methods proposed by Greco [15], Song [16] and Bose [17] use clustering and abstraction at the level of traces to mine complex process having noise. Similarly, the methods presented by Dongen [18] and Gunther [19],[20] also mine complex and noisy processes by using clustering and abstraction at the level of activities.

Goedertier et. al.[21] described a robust process discovery method which is based on generating negative examples artificially to enhance learning. Work proposed by Ferreira and Gillblad [11] deals with discovering process models from event logs where case IDs are unavailable.

As a supportive tool, process mining has also been used for detecting anomalous event behavior [22] and distributed work flow execution [23].

van der Aalst in his work in [24], [25], [26], and [27] applied the divide and conquer strategy for event log decomposition to tackle with the problem of large event logs. To decompose large event logs the concept of passages is proposed in [24]. Passages are more suitable for distributed conformance checking rather than process discovery as prior knowledge about the process model is required to identify passages. In [28] a heuristic based streaming process discovery algorithm has been proposed where the mining algorithm updates a finite queue of events with new events to periodically rediscover the process model rather than updating the discovered process model.

Discovering state based process models by identifying different facets, or perspectives rather than focusing on events or activities that are executed in a particular process has been proposed by van Eck in [29].

In [30] van der Aalst et al. proposed a generic process discovery approach based on localized event logs. Events are localized by assigning a non-empty set of regions to each event and it is assumed that regions can only interact through shared events.

A novel reverse engineering technique for obtaining real-life event logs from distributed systems has been proposed by Leemans and van der Aalst in [31] where they have been able to analyze the operational processes of software systems under real-life conditions, and use process mining techniques to obtain precise and formal models.

The existing literature has attempted to discover processes under the presence of noise in the event log data, presence of loops in the process model, using clusters, divide and conquer strategies. However, the strategy of exploiting parallelism in the algorithm has not been sufficiently exploited. The present paper proposes to exploit parallelism to improve efficiency in executing the Alpha algorithm.

3. Event logs and process mining

The concept of process mining is explained in this section by taking an example from [4] and this has been used as a continuing example in the remaining sections of this paper.

Event logs are a collection of events citing to cases (process instances), the event types (activities) and the respective time stamps of each event type. Events correspond to the implementation of activity instances or tasks. In a work flow management system this can be represented by the completion of a task or activity. Thus, event types are synonymous to tasks or activities. The example event log consisting of 22 events over 3 cases are shown in Table 1. The execution of activity X indicates beginning of each case, whereas execution of activity Z indicates ending of each case. The execution of activity M is followed by the execution of activity N and the vice-versa is also true. A finite set of activities available for each individual case constitute a trace. A trace σ can be defined as a set of finite events or activities for a single case (process instance)in an event log where events follow a temporally ordered sequence, e.g., (Trace) $\sigma = t_1 \ldots t_n$ The event log which is taken as an example in Table 1 contains the following six traces:

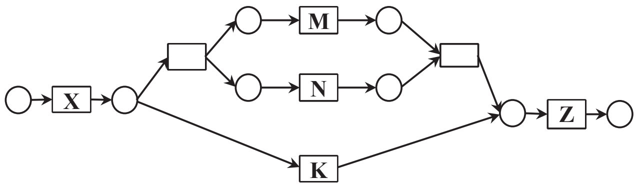

Figure 1: A work flow net representing the event log in Table.1

σ_1 = σ_6= $XMNZ$, σ_2 = σ_5 = $XNMZ$, σ_3 =σ_4 = XKZ

Figure.1 shows the work flow net [32] pertaining to a process model that can result in the event log shown in Table.1

Table 1: An Example of Event Log

CaseID	Activity	Time Stamp
1	*X*	1
1	*M*	2
2	*X*	3
2	*N*	4
3	*X*	5
3	*K*	6
1	*N*	7
1	*Z*	8
2	*M*	9
2	*Z*	10
3	*Z*	11
4	*X*	12
4	*K*	13
5	*X*	14
5	*N*	15
6	*X*	16
6	*M*	17
5	*M*	18
6	*N*	19
4	*Z*	20
5	*Z*	21
6	*Z*	22

Filtering out noise is an important task in process mining. Noise can be defined as “outliers” [1], i.e. remarkably uncommon traces, but not errors in the log. As the algorithm may result in a massively complex process model by attempting to measure also for uncommon traces, the “noisy” data are not taken into account in the Alpha algorithm [4].

Any process mining approach begins with establishing the ordering relationships among activities present in a trace in the event log. Different process mining techniques apply different procedures to determine the ordering relations based on event logs, however. An activity does or does not succeed another activity forms the basis of this ordering relationship. In a recorded event log, all such ordering relations are fetched out and accumulated for further processing. It is significant to know in which manner and of what frequency two events pursue each other rather than retaining each process instance or trace. Once the computations of these ordering relations are completed, a relatively small data set need to be operated on by the process mining algorithms to generate the process model.

4. The MPI model

Message Passing Interface (MPI) is one of the many paradigms available for writing parallel programs. It can be employed on a single processing node as well as on multiple connected nodes for parallel processing. The primary objective of MPI is to provide efficient, portable and flexible standard for programs that utilize message passing for computation. MPI is also responsible for providing a suitable programming environment for the tasks or computations having regular communication patterns.

Several subroutines constitute the MPI library which is being used to identify and implement explicit parallelism in programs by using special constructs. Most of the distributed memory architecture use MPI constructs to speed up the computation. Message passing paradigm is based on dividing a single computational task into several independent processes and running these processes on different computing nodes concurrently. This scheme involves creating many concurrent processes and distributing the required data among these processes for computation. There are several data distribution schemes. MPI does not have the concept of shared data. Data required by one process, if held by another process, must be sent to the former process by initiating a send operation by the later process. The internal methods and policies that are required for message delivery are described in an MPI message passing protocol.

Eager and rendezvous [33] are the two commonly used message passing protocols. Eager is an asynchronous protocol where a send operation is allowed to complete without getting an acknowledgment from a matching receive operation. On the other hand, rendezvous is a synchronous protocol where an acknowledgment is required from a matching receive operation for a send operation.

Writing parallel programs using MPI is a risky task as MPI allows programmers to have control over data distribution as well as process synchronization. Hence, correctness of a parallel program is affected by the data distribution and synchronization requirements.

The proposed work focuses on the parallel programs developed by MPI using MPICH2 [34] which is an open source implementation of MPI. In this case to get the final executables, an MPI source program is first compiled and then linked with the MPI libraries. A user can place a copy of the executable program on a processor by issuing a directive to the operating system. The user directive must contain the number of processes. This implementation is compatible with a variety of hardware platforms and also can be integrated with programming languages such as C/C++ and FORTRAN. This paper is about MPI implementation of the Alpha algorithm [4] using C.

5. Redefining the Alpha algorithm using the MPI API

The MPI implementation of the Alpha algorithm is described in this section. The algorithm is being described by highlighting the ordering relations available among activities in an event log.

5.1. The Alpha Algorithm

The objective of the Alpha algorithm is to discover a process model and represent it in the form of a work flow net [32] from a recorded event log assuming zero noise in the log data. The first step of the Alpha algorithm is to trace out the *causal* relationship [4] which can be derived from the ordering relationships existing between the activities in an event log. As an example, we can say that, activity X is causally related to activity Y when Y always succeeds X but not the vice versa in an event log. Further, there may be four types of *causal* relationships possible between any two activities depending on the ordering relations that may be present between any two activities. These four types of ordering relations can be defined as follows:

Let A be a set of activities and P be an event log over A. Let $x, y \in A$:

- $x >_P y$ iff there is a trace $\sigma = t_1 t_2 t_3 \ldots t_{n-1}$ in P such that $\sigma \in P$ and $t_i = x$ and $t_{i+1} = y$ for $i \in \{1, \ldots, n-2\}$
- $x \rightarrow_P y$ iff $x >_P y$ and $y \not>_P x$
- $x \parallel_P y$ iff $x >_P y$ and $y >_P x$
- $x \sharp_P y$ iff $x \not>_P y$ and $y \not>_P x$

The $>_P$ represents the basic temporal ordering relationship between two activities from which other relations are derived. A possible causal ordering relationship between two activities is represented by $\rightarrow_P$. The $\parallel_P$ represents the parallel relationship between two activities in the event log. The $\sharp_P$ represents unrelated activities in the event log, i.e., those activities that never follow each other directly.

For the event log shown in Table 1 the pair of activities having the basic temporal relationship is shown in Table 2. Table 3 represents the pair of activities having causal relations. The parallel relations computed from the event log is shown in Table 4. Similarly, Table 5 shows those activities which are unrelated to each other.

Table 2: Basic Temporal Ordering Relation

Activity x	Activity y
X	M
M	N
N	Z
X	N
N	M
M	Z
X	K
K	Z

Table 3: Causal Relations

Activity x	Activity y
X	M
X	N
M	Z
N	Z
X	K
K	Z

Table 4: Parallel Relations

Activity x	Activity y
M	N
N	M

The detailed steps involved in the Alpha algorithm are as follows:

(i) $A_P = \{t \in A | \exists_{\sigma \in P} t \in \sigma\}$
Determine the set of unique activities present in the event log. In the given example event log P, A_P from A is $\{X, M, N, Z, K\}$.

(ii) $A_S = \{t \in A | \exists_{\sigma \in P} t = first(\sigma)\}$
From the set of all activities A, determine the set of activities which do not have immediate predecessor anywhere in any of the trace in the log. In the given example event log P, A_S is $\{X\}$.

(iii) $A_E = \{t \in A | \exists_{\sigma \in P} t = last(\sigma)\}$
From the set of all activities A, determine the set of activities which do not have immediate successor anywhere in any of the trace in the log. In the given example event log P, A_E is $\{Z\}$.

Table 5: Unrelated Activities

Activity x	Activity y
X	Z
M	K
N	K
X	X
M	M
Z	Z
N	N
K	K
Z	X
K	M
K	N

(iv) Determine the above mentioned relations $(>, \rightarrow, \|, \sharp)$ between all activities and represent them in the form of a matrix called footprint by scanning through the traces present in the event log. The generated footprint matrix for P is shown in Table 6.

(v) $R_P = \{(X,Y) | X \subseteq A_p \wedge X \neq \varnothing \wedge Y \subseteq A_P \wedge Y \neq \varnothing \wedge \forall_x \in X \forall_y \in Y x \rightarrow_P y \wedge \forall_{x1,x2} \in X x1 \sharp_P x2 \wedge \forall_{y1,y2} \in Y y1 \sharp_P y2\}$
The set R_P is generated by using the footprint matrix and can be given as:
$R_P = \{(\{X\},\{M\}),(\{X\},\{N\}),(\{X\},\{K\}),(\{M\},\{Z\}),(\{N\},\{Z\}),(\{K\},\{Z\}),(\{X\},\{M,K\}),(\{X\},\{N,K\}),(\{M,K\},\{Z\}),(\{N,K\},\{Z\})\}$ by considering the example event log.

(vi) $Q_P = \{(X,Y) \in R_P \mid \forall_{(X',Y') \in} R_P X \subseteq X' \wedge Y \subseteq Y' \Rightarrow (X,Y) = (X',Y')\}$
The set Q_P represents the maximal set pairs which can be stated as: R_P, if for a set pair (X,Y), all activities in X are a subset of activities in set X' and all activities in set Y are a subset of activities in set Y' and (X',Y') set pair is present in R_P, then set pair (X,Y) in R_P is considered to be same as set (X',Y'). In the given example event log:
$Q_P = \{(\{X\},\{M,K\}),(\{X\},\{N,K\}),(\{M,K\},\{Z\}),(\{N,K\},\{Z\})\}$.

(vii) In L_P, a place is generated for each distinct pair of set (X,Y). Along with it, an input place S_P and output place E_P are generated. In the given example event log:
$L_P = \{S_P, E_P, L(\{X\},\{M,K\}), L(\{X\},\{N,K\}), L(\{M,K\},\{Z\}), L(\{N,K\},\{Z\})\}$

(viii) $F_P = \{(x, L_{(X,Y)}) | (X,Y) \in Q_P \wedge x \in X\} \cup \{(L_{(X,Y)}, y) \mid (X,Y) \in Q_P \wedge y \in Y\} \cup \{(S_P, t) \mid t \in A_S\} \cup \{(t, E_P) \mid t \in A_E\}$
The flow relation F_W is generated as: For each set pair (X,Y) in Q_P, arcs are connected from every activity present in set X to a place generated for the set pair (X,Y) and arcs are also connected from the place to every activity present in set Y. Activities in A_S are connected to the input place S_P and activities in A_E are connected to the output place E_P. In the given example event log:
$F_P = \{(S_P, X), (X, L_{(\{X\},\{M,K\})}), (L_{(\{X\},\{M,K\})}, Y), \ldots, (Z, E_P)\}$

(ix) Finally, $Alpha(P) = (L_P, A_P, F_P)$ The generated work flow net of the Alpha algorithm is represented by L_P, A_P and F_P is shown in Figure 1.

Table 6: Footprint matrix of the given example event log

	X	M	N	Z	K
X	$\sharp$	$\rightarrow$	$\rightarrow$	$\sharp$	$\rightarrow$
M	$\leftarrow$	$\sharp$	$\|$	$\rightarrow$	$\sharp$
N	$\leftarrow$	$\|$	$\sharp$	$\rightarrow$	$\sharp$
Z	$\sharp$	$\leftarrow$	$\leftarrow$	$\sharp$	$\leftarrow$
K	$\leftarrow$	$\sharp$	$\sharp$	$\rightarrow$	$\sharp$

5.2. The MPI implementation of the Alpha algorithm

To parallelize the Alpha algorithm it is important to identify the discrete and independent tasks involved in it which can be executed concurrently. Specifically, a thorough investigation is required to exploit task-parallelism [35]. In task-parallelism, the various tasks carried out in solving a problem are partitioned among the computing processors or cores. From a thorough analysis of the Alpha algorithm it is found that step (i) through step (iii) are independent tasks, and hence, these can be run in parallel to take advantage of task-parallelism. Task-parallelism can again be exploited in step (iv) of the Alpha algorithm where it involves deriving the four types of ordering relations among various activities recorded in the event log and building the footprint matrix. Once the footprint matrix is generated, step (v) and step (vi) can be executed in parallel using task-parallelism. It may be noted that the size of the event log, where the log-based ordering relations occurs, is significantly large in comparison to the size of the data set that is being used for the subsequent computations of L_P, F_P, and generation of the work flow net in Step (vi) through step (ix) respectively. The tasks involved in computing L_P, F_P, and generation of the work flow net are dependent on the number of distinct or unique activities fetched out from the recorded event log rather than the size of the event log itself. Thus, it is significant to employ the distributed memory multiprocessing from the MPI API for the execution of step (i) through step (vi) while deriving the ordering relations available in the event log that can benefit from the parallel computation.
Using the message passing constructs available in the MPI API the Alpha algorithm can be redefined as follows to take advantage of parallel computation:

```
 Step 1 : Read event log data
 Step 2 : Initialize MPI by calling MPI_Init()
 Step 3 : Create new processes and assign ranks to these processes
          to carry out the tasks mentioned in the step (i) through
          (iii) of the Alpha algorithm
 Step 4 : Assign tasks to the newly created processes
          if( process rank == 0)    { Compute $A_P$;}
          if( process rank == 1)    { Compute $A_S$;}
          if( process rank == 2)    { Compute $A_E$;}
 Step 5 : Synchronize all processes by calling MPI_Barrier()
 Step 6 : Assign tasks to the newly created processes
          if( process rank == 0)    { Compute $x >_P y$;}
          if( process rank == 1)    { Compute $x \rightarrow_P y$;}
          if( process rank == 2)    { Compute $x \|_P y$;}
          if( process rank == 3)    { Compute $x \sharp_P y$;}
 Step 7 : Synchronize all processes by calling MPI_Barrier()
 Step 8 : Create new processes and assign ranks to these processes
          to carry out the tasks mentioned in the step (v) and (vi)
          of the Alpha algorithm
 Step 9 : Assign tasks to the newly created processes
          if( process rank == 0)    { Compute $R_P$;}
          if( process rank == 1)    { Compute $Q_P$;}
Step 10 : Synchronize all processes by calling MPI_Barrier()
Step 11 : Compute $L_P$;
Step 12 : Draw $Alpha(P)$;
```

A schematic representation of the above mentioned redefined version of the Alpha algorithm is given in Figure 2.

6. Experimental set up and results

Scalability and effectiveness of the redefined version of the Alpha algorithm were evaluated through an experimental study. A small tool was developed using C programming language to generate the synthetic event logs. In each event log file the following three fields are present: case id, activity name and time stamp of each activity. This tool has been used to generate event logs based on

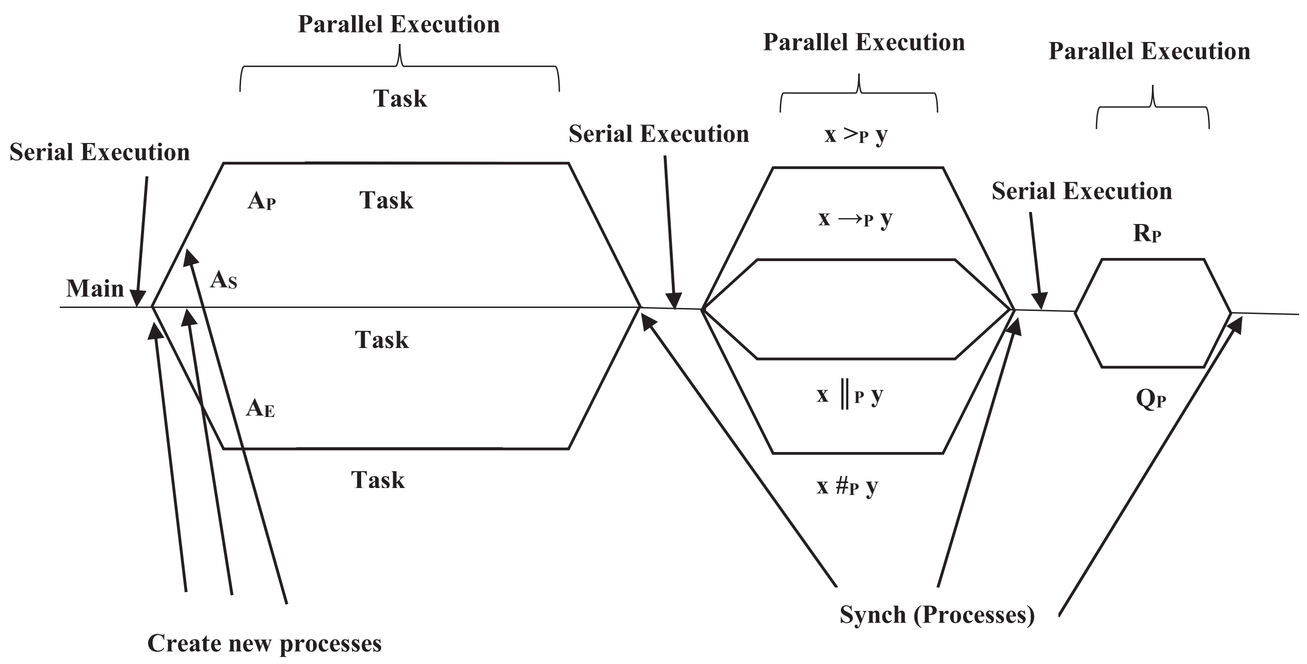

Figure 2: Schematic representation of the redefined Alpha algorithm

different process models by varying the following two parameters: number of activities and proportion of splits (like AND splits and XOR splits). Activity names were represented through the English alphabets from 'A' to 'Y'. To check the impact of number of unique activities on the over all execution time of the Alpha algorithm, four process models were created with 10, 15, 20 and 25 unique activities respectively. The number of cases were set to 8, 13, 18 and 23 respectively for the above said process models. Based on these four process models, event logs of different sizes consisting of event traces ranging from 3×10^6 to 3×10^7 were generated. The size of event log files generated were varied from approximately 90 MB to 1.08 GB where the use of parallel programming becomes effective. Both the serial version as well as the parallel version of the Alpha algorithm were developed using C programming language with GCC 4.4.6. While developing the parallel version of the Alpha algorithm the Open MPI 1.8.8 API was used and hyper threading was set to on. Table 7 shows the details of machine hardware and software configuration used for the experimental set up.

Table 7: Hardware and software configurations used for experiments

Parameters	Values
CPU (2 Numbers) per node of a 13 node High Performance Computing Cluster	E5-2650 Intel Xeon @ 2.0GHz
Physical Cores per node	16
Operating System	Red Hat Linux
GCC Compiler	Version 4.4.6
OpenMPI	Version 1.8.8

To get the accurate execution timings of programs, the experiments were performed in isolation. Table 8 to 11 shows the average execution time of 100 runs of serial as well as parallel version of the Alpha algorithm for 10, 15, 20 and 25 activity sets respectively. The number of traces were varied from 3×10^6 to 3×10^7 for each activity set to generate separate event logs. The performance comparison between the serial execution time and the parallel execution time recorded in seconds for each individual set of activities are shown in Figure 3 through 6.

Table 8: Average execution time of 100 runs of serial as well as parallel version of the Alpha algorithm for 10 unique activities

No. of Traces	T_{old} (in Seconds)	T_{new} (in Seconds)
3000000	0.3730782	0.0944132
6000000	0.7457835	0.191517889
9000000	1.127482667	0.286959667
12000000	1.513307333	0.38105
15000000	1.92095875	0.4893884
18000000	2.242879714	0.565480167
21000000	2.62120375	0.662925667
24000000	3.0105085	0.7608428
27000000	3.396431857	0.8697522
30000000	3.8610816	0.97478725

Table 9: Average execution time of 100 runs of serial as well as parallel version of the Alpha algorithm for 15 unique activities

No. of Traces	T_{old} (in Seconds)	T_{new} (in Seconds)
3000000	0.466754099	0.118119333
6000000	0.936497601	0.241112625
9000000	1.33079863	0.338792571
12000000	1.791727937	0.4518388
15000000	2.256376384	0.5739615
18000000	2.687997151	0.677909333
21000000	3.098169096	0.7835148
24000000	3.483099485	0.880258
27000000	4.011759718	1.019492167
30000000	4.493973526	1.137441833

The speedup (S) equation can be given as:

$$S = T_{old}/T_{new} \quad (1)$$

where T_{old} is the execution time without any improvement or serial execution time and T_{new} is the new execution time with

Table 10: Average execution time of 100 runs of serial as well as parallel version of the Alpha algorithm for 20 unique activities

No. of Traces	T_{old} (in Seconds)	T_{new} (in Seconds)
3000000	0.618671211	0.156168571
6000000	1.206095859	0.308301429
9000000	1.84865367	0.4699095
12000000	2.431231725	0.613604333
15000000	3.022988887	0.76624025
18000000	3.648282832	0.919373333
21000000	4.250329563	1.073778286
24000000	4.897456803	1.237426778
27000000	5.466739782	1.385718857
30000000	6.032665968	1.538624333

Table 11: Average execution time of 100 runs of serial as well as parallel version of the Alpha algorithm for 25 unique activities

No. of Traces	T_{old} (in Seconds)	T_{new} (in Seconds)
3000000	0.71672409	0.18089225
6000000	1.383316825	0.3528266
9000000	2.11859639	0.53893725
12000000	2.754495407	0.695379333
15000000	3.422744406	0.867125
18000000	4.131517661	1.042739
21000000	4.839376429	1.223209667
24000000	5.435420378	1.374165
27000000	6.169023575	1.558599
30000000	6.849986317	1.738214333

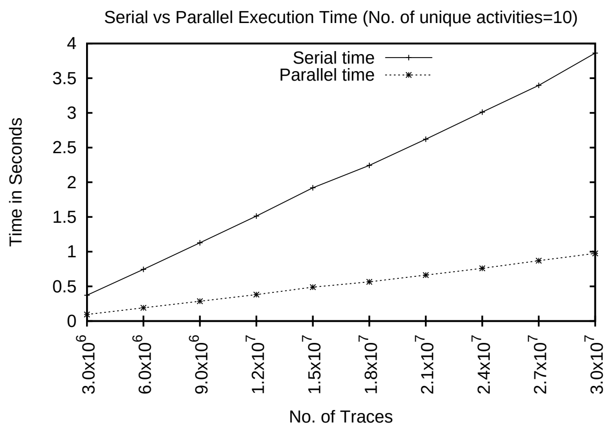

Figure 3: Comparison between Serial and Parallel execution time (No. of unique activities=10)

improvement or parallel execution time [36]. For this experimental set up: T_{old} = Average Serial Execution Time and T_{new}= Average Parallel execution time.

The speedup value was set to 1x for the serial execution time. Table 12 represents the respective speedup achieved for different number of traces. It can be observed from Table 12 that the maximum and the minimum speedup achieved are 3.97x and 3.88x respectively with an average speedup of 3.94x.

7. Conclusion

A series of experiments are conducted to observe the performance of the Alpha algorithm on parallelization. The MPI parallel computing construct is used for computations across multiple nodes of the computing cluster. From experiment it is observed that the speedup is as high as 3.97x and the average speedup is 3.94x with the parallel version of the Alpha algorithm. It is also observed that in a given data set, the parallel execution outperforms the serial execution. Thus, we are able to accelerate the execution of the Alpha Miner algorithm using task based parallelism and achieved better execution time by utilizing the potential of distributed memory computing.
Our proposed parallel approach for computing the different discrete and independent tasks of the Alpha algorithm can now be used on larger size event logs. The proposed parallel approach of building footprint matrix and finding maximal set pairs can enable to work on higher count of activities.

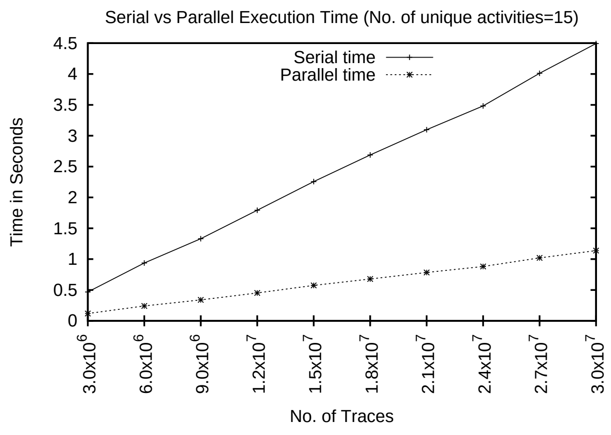

Figure 4: Comparison between Serial and Parallel execution time (No. of unique activities=15)

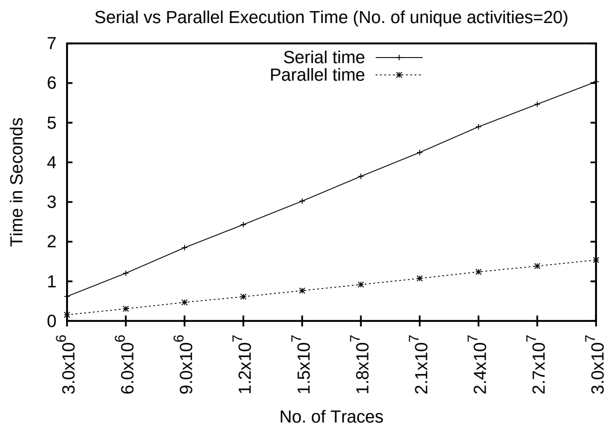

Figure 5: Comparison between Serial and Parallel execution time (No. of unique activities=20)

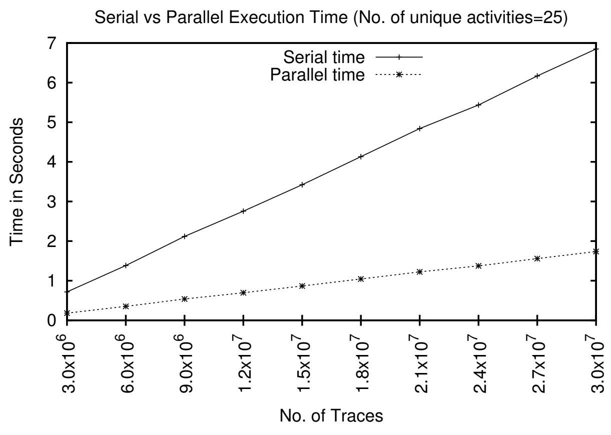

Figure 6: Comparison between Serial and Parallel execution time (No. of unique activities=25)

Table 12: Speedup achieved for different number of traces

No.of traces	**Speedup**			
_	**10 Activities**	**15 Activities**	**20 Activities**	**25 Activities**
3000000	3.9515470294408	3.95154702575234	3.96156030011954	3.96216029155478
6000000	3.89406704456731	3.88406704543157	3.91206704072721	3.92067045115079
9000000	3.92906319827866	3.9280631982925	3.93406319727522	3.93106319891601
12000000	3.97141407426847	3.96541407466557	3.96221407549937	3.96114074186901
15000000	3.92522329912192	3.93123299036608	3.94522329908929	3.94723298947672
18000000	3.9663278128727	3.9651278130434	3.96822781458683	3.96217812990595
21000000	3.95399345730869	3.95419345748159	3.95829345630854	3.95629347899848
24000000	3.95680750346852	3.95690750325473	3.95777502965917	3.95543503000004
27000000	3.90505693115809	3.93505693114364	3.9450569315591	3.95805693125685
30000000	3.96094799147199	3.95094799190492	3.92081799215897	3.94081799174763

Working with MPI model has the following two major benefits as the memory is distributed among multiple nodes: (i) It is a cost effective way to scale the memory bandwidth if the accesses are to the local memory in the node (ii) It reduces the latency for accesses to the local memory. On the other hand, MPI model also suffers from higher communication latency problem as communicating data between processors becomes somewhat more complex. The proposed model exploits only the task level parallelism available in the Alpha algorithm. But, better speedup may be achieved if we could exploit both data-parallelism and task-parallelism available in the Alpha algorithm. Thus, a hybrid approach where both data and task parallelism could be exploited for obtaining better speedup would be our future scope of research.

References

[1] Wil MP van der Aalst, Process Mining, *Springer*, (2011).

[2] Wil MP van der Aalst, “Process mining: Overview and opportunities”, *ACM Transactions on Management Information Systems (TMIS)*, Vol.3, No.2, (2012), p.7.

[3] W. Aalst, A. Adriansyah, A. K. A. Medeiros, F. Arcieri, T. Baier, T. Blickle, J. C. Bose, P. Brand, R. Brandtjen, J. Buijs et al, “Process mining manifesto”, *Business process management workshops*, Springer (2012), pp.169-194.

[4] W. Van der Aalst, T. Weijters, and L. Maruster, “Workflow mining:Discovering process models from event logs”, *IEEE Transactions on Knowledge and Data Engineering*, Vol.16, No.9, (2004), p.1128-1142.

[5] T. Andrews, S. Qadeer, S. K. Rajamani, J. Rehof, and Y. Xie, “Zing: A model checker for concurrent software”, *in International Conference on Computer Aided Verification* , Springer (2004), pp.484-487.

[6] A. Weijters, W. M. van Der Aalst, and A. A. De Medeiros, “Process mining with the heuristics miner-algorithm”, *Technische Universiteit Eindhoven, Tech. Rep. WP*, Vol.166, (2006), pp.1-34.

[7] L. Wen, W. M. van der Aalst, J. Wang, and J. Sun, “Mining process models with non-free-choice constructs”, *Data Mining and Knowledge Discovery*, vol. 15, no.2, (2007), pp. 145–180.

[8] G. Schimm, “Mining exact models of concurrent workflows”, *Computers in Industry*, vol.53, no.3, (2004), pp.265–281.

[9] W. M. Van Der Aalst, V. Rubin, H. M. Verbeek, B. F. van Dongen, E. Kindler, and C. W. G¨unther, “Process mining: a two-step approach to balance between underfitting and overfitting”, *Software and Systems Modeling*, vol.9, no. 1, (2010), pp.87–111.

[10] R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser, “Process mining based on regions of languages”, *in International Conference on Business Process Management.* Springer(2007), pp.375–383.

[11] D. R. Ferreira and D. Gillblad, “Discovering process models from unlabelled event logs”, *in International Conference on Business Process Management.* Springer(2009), pp.143–158.

[12] W. M. Van der Aalst and A. Weijters, “Process mining: a research agenda”, *Computers in industry*, vol. 53, no.3, (2004), pp.231–244.

[13] A. K. de MEDEIROS, A. J. Weijters, and W. M. van der Aalst, “Genetic process mining: an experimental evaluation”, *Data Mining and Knowledge Discovery*, vol.14, no.2, (2007), pp.245–304.

[14] C. J. Turner, A. Tiwari, and J. Mehnen, “A genetic programming approach to business process mining”, *in Proceedings of the 10th annual conference on Genetic and evolutionary computation.* ACM(2008), pp.1307–1314.

[15] G. Greco, A. Guzzo, L. Pontieri, and D. Sacca, “Discovering expressive process models by clustering log traces”, *IEEE Transactions on Knowledge and Data Engineering*, vol.18, no.8, (2006), pp.1010–1027.

[16] M. Song, C. W. G¨unther, and W. M. Aalst, “Trace clustering in process mining”, *in Business Process Management Workshops.* Springer(2009), pp.109–120.

[17] R. J. C. Bose and W. M. van der Aalst, “Context aware trace clustering:Towards improving process mining results”, *in Proceedings of the 2009 SIAM International Conference on Data Mining.* SIAM(2009), pp.401–412.

[18] B. F. van Dongen and A. Adriansyah, “Process mining: fuzzy clustering and performance visualization”, *in International Conference on Business Process Management.* Springer(2009), pp.158–169.

[19] C. W. G¨unther, A. Rozinat, and W. M. Van Der Aalst, “Activity mining by global trace segmentation”, *in International Conference on Business Process Management.* Springer(2009), pp.128–139.

[20] C. W. G¨unther and W. M. Van Der Aalst, “Fuzzy mining–adaptive process simplification based on multi-perspective metrics”, *in International Conference on Business Process Management.* Springer(2007), pp.328–343.

[21] S. Goedertier, D. Martens, J. Vanthienen, and B. Baesens, “Robust process discovery with artificial negative events”, *Journal of Machine Learning Research*, vol.10, no.Jun, (2009), pp.1305–1340.

[22] L. V. Allen and D. M. Tilbury, “Anomaly detection using model generation for event-based systems without a preexisting formal model”, *IEEE Transactions on Systems, Man, and Cybernetics-Part A:Systems and Humans”*, vol.42, no.3, (2012), pp. 654–668.

[23] S. X. Sun, Q. Zeng, and H.Wang, “Process-mining-based workflow model fragmentation for distributed execution”, *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol.41, no.2, (2011), pp.294–310.

[24] W. M. Van Der Aalst, “Decomposing process mining problems using passages”, *in International Conference on Application and Theory of Petri Nets and Concurrency.* Springer(2012), pp.72–91.

[25] W. M. Van der Aalst, “Decomposing petri nets for process mining: A generic approach”, *Distributed and Parallel Databases*, vol.31, no.4, (2013), pp. 471–507.

[26] ——, “Process mining in the large: a tutorial”, *in Business Intelligence.* Springer(2014), pp. 33–76.

[27] W. M. Van Der Aalst, “A general divide and conquer approach for process mining”, *in Computer Science and Information Systems (FedCSIS)*, Federated Conference on. IEEE(2013), pp.1–10.

[28] A. Burattin, A. Sperduti, and W. M. van der Aalst, “Heuristics miners for streaming event data”, *arXiv preprint arXiv:1212.6383*, (2012).

[29] M. L. van Eck, N. Sidorova, and W. M. van der Aalst, “Discovering and exploring state-based models for multi-perspective processes”, *in International Conference on Business Process Management.* Springer(2016), pp.142–157.

[30] W. M. van der Aalst, A. Kalenkova, V. Rubin, and E. Verbeek, “Process discovery using localized events”, *in International Conference on Applications and Theory of Petri Nets and Concurrency.* Springer(2015), pp.287–308.

[31] M. Leemans andW. M. van der Aalst, “Process mining in software systems: discovering real-life business transactions and process models from distributed systems”, *in Model Driven Engineering Languages and Systems (MODELS)*, 2015 ACM/IEEE 18th International Conference on. IEEE(2015), pp.44–53.

[32] W. M. Van der Aalst, “The application of petri nets to workflow management”, *Journal of circuits, systems, and computers*, vol.8, no.01, (1998), pp.21–66.

[33] R. Brightwell and K. Underwood, “Evaluation of an eager protocol optimization for mpi”, *in European Parallel Virtual Machine/Message Passing Interface Users Group Meeting.* Springer(2003), pp.327–334.

[34] D. Buntinas, G. Mercier, and W. Gropp, “Implementation and evaluation of shared-memory communication and synchronization operations in mpich2 using the nemesis communication subsystem”, *Parallel Computing*, vol.33, no. 9, (2007), pp. 634–644.

[35] P. Pacheco, An introduction to parallel programming. *Elsevier*, (2011).

[36] L. John, Hennessy and david a. patterson, Computer architecture a quantitative approach, *The Morgan Kaufmann Series in Computer Architecture and Design*, (2003).