MANUFACTURING SIMULATION USING BSP TIME WARP WITH VARIABLE NUMBERS OF PROCESSORS

Malcolm Yoke Hean Low
Programming Research Group, Computing Laboratory, University of Oxford
Wolfson Building, Parks Road, Oxford, OX1 3QD, UK
E-mail: mlow@comlab.ox.ac.uk

ABSTRACT

The performance of an optimistic parallel simulation run depends on many factors. These factors range from the inherent complex nature of the simulation workload to the overhead in using the available parallel computing resources. Very often, using the maximum number of available processors for a parallel simulation run will not yield the best performance achievable. In this paper, we describe a new approach for executing BSP Time Warp optimistic parallel simulations using variable number of processors. Processors are automatically added or removed during the simulation runtime based on a performance cost model. Experiment results using this new approach on a real-world semi-conductor factory model is presented.

1 INTRODUCTION

The performance of an optimistic parallel simulation protocol [Jefferson 1985] such as the BSP Time Warp (BSP-TW) [Marín 1998] depends on different numbers of factors, both internal and external. In order to maximize the performance of a long running parallel simulation, the simulation protocol needs to adapt to its surrounding environment and make the necessary changes when required. These changes can range from throttling the event-limit between supersteps [Low 2001] to dynamic load-balancing via migration of simulation objects between processors [Low 2002]. Depending on the amount of overhead and the granularity of events of the simulation model, using the maximum number of processors available in a parallel system throughout the whole duration of a simulation run may not always yield the best performance. The simulation protocol must make decision whether to make use of additional available processors or discard some of the processors by migrating simulation objects out of them.

In this paper, we propose a new approach for executing BSP-TW simulation by varying the number of processors during the runtime of a simulation. The new algorithm automatically adds or removes processors from the parallel simulation environment based on a performance cost model. We carried out experiments on a real-world semi-conductor wafer fabrication model using this new algorithm and compared its performance with the BSP-TW DLB$_{ccl}$ algorithm described in [Low 2002].

The rest of this paper is organized as follows. We first describe the BSP model and the BSP Time Warp optimistic protocol in section 2. The performance cost model that is used for determining whether to add or remove processors from a simulation run is described in section 3. Based on the performance model, the new algorithm for automatically adding or removing processors during runtime is described in section 4. Section 5 describes the Sematech wafer fabrication model used in the experiments as well as presents the experiment results comparing the performance of the new BSP-TW DLB$_{accl}$ algorithm and the BSP-TW DLB$_{ccl}$ algorithm. Some related work are described in section 6. Section 7 summarizes the paper and outlines future research directions.

2 BSP TIME WARP

The bulk synchronous parallel (BSP) model [Valiant 1990] is developed to be a general purpose approach to parallel computing. It has features such as simple programming interfaces, scalable performance and a simple cost model for performance prediction. The BSP model allows the separation of concerns between the computation, communication and synchronization costs when designing a parallel algorithm. A BSP programming model consists of P processors linked by an inter-connecting network and each with its own pool of memory. BSP processors communicate with one another by exchanging messages using the inter-connecting network.

The BSP-TW algorithm [Marín 1998] is designed to be an efficient realization of an optimistic synchronization protocol on the BSP model. The algorithm for the original BSP-TW is shown in Figure 1. Each processor manages a group of logical processes (LPs) in the system. In BSP-TW, LPs are also referred to as simulation objects and the two terms are used interchangeably in this paper. LPs in the same processor share a common event-list. The BSP-TW algorithm proceeds in a series of supersteps as indicated by the outer `while` loop and the `bsp_sync()` statement at the end of the loop.

```
bsp_begin();
[A] Initialization
while GVT < SimEndTime do
    [B] Receive external events and process rollback;
    [C] Compute new GVT, perform fossil collection and
        compute new event limit $n_e$ every $n_g$ supersteps;
    [D] Execute $n_e$ events;
    bsp_sync();
endwhile
bsp_end();
```

Figure 1: Algorithm for BSP Time Warp

The BSP-TW algorithm uses event-rollback to correct any violations in time-order relationship between events on LPs located in different processors. The global virtual time (GVT) measures the progress of a simulation run. An estimate of GVT is computed after every n_g supersteps; n_g is also known as the GVT update interval. Memory for events or states in an LP with time-stamps smaller than GVT are reclaimed after each GVT computation (fossil collection). The body of the loop is executed till the processor's GVT value is greater than the simulation end time.

The algorithm provides an automatic means of throttling the number of events, n_e, being simulated per superstep based on statistics from fossil collected events. The aim of the algorithm is to complete the simulation in the least number of supersteps possible. The BSP cost model for a BSP-TW algorithm S can be expressed as

$$\text{cost}(S) = \sum_{i=1}^{n_s} \Big(w(i) + gh(i) + l \Big) \qquad (1)$$

where n_s is the total number of supersteps; $w(i)$ is the computation cost for superstep i; and $h(i)$ is the maximum number of messages sent or received respectively by any processor in superstep i. The architecture dependent parameters g and l represent the communication and synchronization costs respectively. The values of g and l for some commonly used parallel systems are listed in [Skillicorn et al. 1997].

Although the cost model is relatively simple, we can see that the performance of a BSP-TW algorithm relies on three factors: a) computation balance; b) communication balance; and c) n_s, the total number of supersteps.

3 PERFORMANCE COST MODEL

In order to decide the number of processors needed for a parallel simulation, the following factors have to be considered:

- Computation and communication load-imbalance
- Communication and synchronization overhead
- Event rollback rate
- Event granularity/overhead.

Let the cost of executing an event using the sequential simulation engine be c_e and the total number of events executed in the sequential execution be n_e. The total cost of sequential execution, C_{seq}, is given by

$$C_{seq} = n_e c_e. \qquad (2)$$

Let P be the number of processors used in the parallel execution of the simulation.

Let $k_e \geq 1$ be the event overhead in the parallel execution. The event overhead accounts for the additional cost of state-saving and fossil collection for each event in the parallel execution. The cost of executing an event in the parallel execution will be $k_e c_e$. Let $k_r \geq 0$ be the event rollback ratio for the parallel execution such that the total number of events executed in the parallel execution is $(1 + k_r)n_e$.

Let k_b ($0 \leq k_b \leq P$) be the load-imbalance factor for the parallel execution of the simulation such that

$$k_b = \frac{n_{max} - \dfrac{(1+k_r)n_e}{P}}{\dfrac{(1+k_r)n_e}{P}}, \qquad (3)$$

$$n_{max} = \frac{(1+k_b)(1+k_r)n_e}{P} \qquad (4)$$

where n_{max} is the maximum number of events executed by any of the P processors.

From the BSP cost equation, the total cost of parallel execution, C_{par}, is given by

$$C_{par} = W_{par} + gH + n_s l \qquad (5)$$

where g and l are the BSP communication and synchronization parameters. W_{par} is the accumulated total computation cost for every superstep. H is the accumulated total of the maximum bytes of data sent or received by any processor in every superstep.

The cost $gH + n_s l$ constitutes the overhead for communication and barrier synchronization. We can define k_o to be the overhead ratio such that

$$k_o = \frac{gH + n_s l}{W_{par}}. \tag{6}$$

The cost of parallel execution can be rewritten as

$$C_{par} = (1 + k_o)W_{par}. \tag{7}$$

Using equation 4, we can rewrite W_{par} as

$$W_{par} = \frac{(1 + k_b)(1 + k_r)k_e n_e c_e}{P}. \tag{8}$$

The total cost for parallel execution can be expressed as

$$C_{par} = \frac{(1 + k_o)(1 + k_b)(1 + k_r)k_e n_e c_e}{P}. \tag{9}$$

The achievable speedup, S, is given by

$$S = \frac{n_e c_e}{\frac{(1 + k_o)(1 + k_b)(1 + k_r)k_e n_e c_e}{P}} \tag{10}$$

$$= \frac{P}{(1 + k_o)(1 + k_b)(1 + k_r)k_e}. \tag{11}$$

4 BSP-TW DLB$_{accl}$ ALGORITHM

Using the DLB cost equation for BSP-TW, we propose a new BSP-TW algorithm that automatically determines the required number of processors for a parallel simulation execution.

This algorithm is an extension of the dynamic load-balancing (DLB) algorithm BSP-TW DLB$_{ccl}$ described in [Low 2002]. The BSP-TW DLB$_{ccl}$ algorithm is enhanced with a module to automatically add or remove processors by comparing the current performance and the expected performance using the cost model developed in section 3.

Figure 2 shows the new BSP-TW DLB$_{accl}$ algorithm. The difference between BSP-TW$_{accl}$ and BSP-TW DLB$_{ccl}$ is the addition of module D0. From part D, the load-balancing algorithm is performed at each migration interval. A migration interval consists of λ GVT computations. Modules D1 to D3 were previously introduced in [Low 2002]. These three modules provide support for balancing computation and communication workload, as well as optimizing lookahead for the BSP Time Warp runtime system. A detailed description of these three modules is beyond the scope of this paper, readers are referred to [Low 2002] for more details.

```
bsp_begin();
[A] Initialization;
while GVT < SimEndTime do
    [B] Receive external events and process rollback;
    [C] Compute new GVT, perform fossil collection and
        compute new event limit $n_e$ every $n_g$ supersteps;
    [D] After each $\lambda$ GVT computation:
        D0 add_remove_cpu();
        D1 balance_computation();
        D2 balance_communication();
        D3 optimize_lookahead();
    [E] Execute $n_e$ events;
    bsp_sync();
endwhile
bsp_end();
```

Figure 2: Algorithm for BSP-TW DLB$_{accl}$

Module D0 is added to provide functionality for adding or removing processors based on predicted performance derived from the cost equation. The pseudo-code for the function `add_remove_cpu()` is shown in Figure 3. In order for the algorithm to decide whether to add/remove processor or to maintain the current number of processors used, the following terms for speedup based on equation 11 are computed:

- S_{cur} : the achievable speedup for the current migration interval.
- S_{add} : the achievable speedup if processors are added.
- S_{rem} : the achievable speedup if processors are removed.

To obtain k_o for the respective speedup terms, we consider equation 6. Since we are considering the achievable speedup for the current migration interval, n_s is set to λn_g for all three cases. Table 1 shows the values of BSP parameters g and l for different numbers of processors used in the experiments in section 5.

In the experiments, the number of processors is doubled or

Table 1: BSP Parameters for a Cluster of Sun UltraSparc Workstations Connected via a 100Mbits TCP/IP Network

P	g (μs/byte)	l (μs)
1	0.0025	2.332
2	0.3275	1210.143
4	0.5225	2069.116
8	0.7235	3186.775
16	1.2695	8287.531

```
add_remove_cpu()
    let $P_i$: the set of current active processor, (0< $i < n$-1);
    compute $S_{cur}$, $S_{add}$ and $S_{rem}$;
    if $S_{add} > S_{cur}$ and $S_{add} > S_{rem}$ then
        // adding $n$ processors
        let $P'_i$: a set of inactive processor, (0< $i < n$-1);
        foreach $P_i$ (0< $i < n$-1) do
            migrate half of $P_i$'s simulation objects to $P'_i$;
            set $P'_i$ as active;
        endfor
    else if $S_{rem} > S_{cur}$ and $S_{rem} > S_{add}$ then
        // removing $\frac{n}{2}$ processors
        sort $P_i$ in descending order by computation workload
        foreach $P_i$ (0< $i < \frac{n}{2}$-1) do
            migrate all of $P_i$'s simulation objects to $P_{i+\frac{n}{2}}$;
            set $P_i$ as inactive;
        endfor
    endif
```

Figure 3: Pseudo Code for `add_remove_cpu()` Procedure

halved each time a decision is made to add or remove processors from the parallel simulation system. We make the assumption that the event overhead, k_e, is 1 no matter how many processors are used. In reality, k_e should be greater than 1 and should remain constant regardless of the number of processors used. We also set the load-balance ratio, k_b, to 0 and use the average computation workload among the active processors as W_{par}. Load-balancing among the set of active processors will be handled by modules D1 to D3 in Figure 2.

We can simplify equation 11 to

$$S = \frac{P}{(1+k_o)(1+k_r)}. \qquad (12)$$

Assuming a well balanced communication workload across all processors, the current communication workload H will be the average of the maximum amount of data sent or received by all processors in the current migration interval. We make estimates to the values of H used to compute S_{add} and S_{rem}. The value of H for S_{add} will be twice the current H value. For S_{rem}, half the current H value is used.

The current value of k_r is obtained using the ratio between the total number of events rolled back and the total number of events committed in the current migration interval. We assign twice the value of k_r for S_{add}; and half the value of k_r for S_{rem}.

Again assuming a well balanced computation workload between all processors, the computation workload W_{par} is taken to be the average of the total computation workload on all processors. We use half the value of W_{par} for S_{add} and twice the value of W_{par} for S_{rem}.

The assumptions for the values of H and k_r will be verified in section 5. Using the values of S_{cur}, S_{add} and S_{rem}, processors are added or removed based on the following conditions:

- If the estimated value of S_{add} exceeds both S_{cur} and S_{rem}, then the algorithm allocates a set of processors from the inactive processor pool. For each of the current active processor, half the simulation objects on the active processor are migrated to one of the inactive processors and the status of the inactive processor is set to active.
- If the estimated value of S_{rem} exceeds both S_{cur} and S_{add}, then the algorithm merges the simulation objects between each pair of active processors and set the status of one of the processors to inactive. When selecting the pair of processors, processor with the highest computation workload is grouped with the processor with the lowest computation workload.
- If the value of S_{cur} exceeds both S_{add} and S_{rem}, then the number of processors is kept unchanged.

Note that processors that are removed from computation are flagged as inactive and modules D1 to D3 in Figure 2 only act on the set of active processors.

5 EXPERIMENT RESULTS

5.1 Simulation Model

In this section, we present experiment results comparing the new BSP-TW DLB$_{accl}$ and the BSP-TW DLB$_{ccl}$ algorithms. The simulation model used to benchmark the two algorithms is a manufacturing process of a wafer fabrication plant. The data model is based on the Sematech Modeling Data Standard (MDS) project [Sematech 1997]. The aim of the project

Table 2: Statistics on Sematech Data-sets

	Data-set					
	1	2	3	4	5	6
No. of products	2	7	11	7	177	9
No. of process flows	2	6	11	1	21	9
No. of process steps	455	1606	4138	111	4176	2541
No. of machines	83	97	73	35	85	104
Steps/machines ratio	5.48	16.56	56.68	3.17	49.13	24.43

is to "develop a set of standard that will enable the seamless exchange, sharing and re-use of data among modeling applications and Manufacturing Execution Systems (MES)".

The MDS data models are realistic examples from real-world applications. The MDS uses several files to define the manufacturing processes. These files define the process flow, rework, tool set, operator set and volume release.

For example, the process flow file defines the workflow of products and contains the information for the different processing steps that wafer lots need to flow through. Each step specifies attributes such as the machine and operator sets that are needed as well as the processing time, load time and unload time etc.

Table 2 shows some statistics for the six Sematech data-sets used in the experiments. The steps/machines ratio measures the average number of processing steps sharing a given machine.

The simulation models used in the experiments are strip-down versions of the detailed model described by Sematech. Some features such as operators, reworks and machine down time are not modelled. The models also use only first-come-first-served wafer-lot processing and do not split wafer-lots on batch-processing machines. These simplifications do not reduce the complexity of the simulation models in terms of the amount of sharing of machines between different processing steps.

5.2 Experiments

For all the experiments, the GVT computation interval, n_g, is fixed at 50 supersteps. The migration interval λ is set to 5. The experiments are conducted on a cluster of 16 350MHz Sun UltraSparc workstations connected via a 100Mbits TCP/IP network. All execution times shown are the average of three runs. A fixed simulation run length of one year (525600 time units) is used for all simulation runs.

The experiments are executed using different spin-loop values to artificially increase the event granularity. We experimented with event granularities of 0, 10, 100 and 1000 μs. The typical event granularities in a detailed simulation model [Jain et al. 1999] are in the range of 10-100 μs.

Table 3: Execution Times (sec.) for Sematech Data-sets using Sequential Simulation Engine

	Event Granularity (μs)			
Data-set	0	10	100	1000
1	10.6	72.3	596.1	5851.4
2	14.7	97.1	850.0	8336.4
3	18.6	137.9	1198.3	11758.6
4	1.2	9.8	87.7	868.2
5	8.6	52.8	440.9	4323.5
6	7.5	54.9	467.5	4594.7

Table 4: Execution Times (sec.) for Sematech Data-sets using BSP-TW DLB$_{ccl}$ on 16 Processors

	Event Granularity (μs)			
Data-set	0	10	100	1000
1	906.5	932.0	1107.9	2784.1
2	809.2	879.3	942.8	2093.8
3	2700.4	2680.5	3055.2	7570.8
4	302.7	301.1	338.2	632.0
5	630.9	600.2	757.3	1742.0
6	519.1	545.8	628.9	1443.2

Table 3 shows the execution times obtained using an optimized sequential simulation engine for the six Sematech data-sets with different event granularities. Data-set 4 has the shortest execution times as it is the smallest model in the Sematech data-sets with only 35 machines and one short wafer-lot process flow.

Table 4 shows the execution times using BSP-TW DLB$_{ccl}$ on 16 processors. Comparing Tables 3 and 4 we can see that the runs with low event granularities suffer from poor performance by using all 16 processors.

Table 5 shows the corresponding percentage of the execution time due to synchronization overhead. We see that the poor performance for runs with low event granularities is due to

Table 6: Execution Times (sec.) for Sematech Data-sets using BSP-TW DLB$_{accl}$ on 16 Processors

	Start with 1 Processor				Start with 16 Processors			
	Event Granularity (μs)				Event Granularity (μs)			
Data-set	0	10	100	1000	0	10	100	1000
1	435.1	510.8	972.1	2574.9	494.6	556.3	1029.5	2437.2
2	400.6	510.3	978.7	2488.2	489.1	581.2	1038.5	2052.4
3	423.0	808.2	1902.7	6189.9	782.4	935.4	2110.2	5946.9
4	122.0	133.1	236.6	655.6	186.8	193.4	283.1	631.3
5	122.8	335.4	601.2	2496.7	249.1	321.9	573.9	1716.7
6	194.4	303.6	558.1	1724.8	285.4	337.2	661.3	1459.3

Table 7: Average Number of Processors Used by BSP-TW DLB$_{accl}$

	Start with 1 Processor				Start with 16 Processors			
	Event Granularity (μs)				Event Granularity (μs)			
Data-set	0	10	100	1000	0	10	100	1000
1	1	1	4	8	1	1	4	8
2	1	1	4	16	1	2	8	16
3	1	1	4	8	1	2	4	8
4	1	1	2	8	1	1	2	8
5	1	2	8	8	1	2	8	16
6	1	1	4	16	1	2	8	16

Table 5: Percentage of Execution Times Spent on Synchronization using BSP-TW DLB$_{ccl}$ on 16 Processors

	Event Granularity (μs)			
Data-set	0	10	100	1000
1	73.7	72.9	63.9	25.0
2	79.7	77.7	69.2	29.6
3	66.4	68.4	56.4	23.0
4	89.9	89.7	78.9	40.1
5	60.6	56.9	51.4	18.2
6	68.5	64.4	59.3	21.4

the high synchronization overhead using all 16 processors. The performance improves with higher event granularities as the synchronization overhead decreases proportionally.

Table 6 shows the execution times using BSP-TW DLB$_{accl}$. The experiments are carried out using two configurations. For the first configuration, 16 processors are allocated at the start of the run but all the simulation objects are initially partitioned onto a single processor. For the second configuration, the simulation objects are uniformly partitioned onto all 16 processors at the start of the run.

Table 7 shows the actual average number of active processors used during the simulation runs. Comparing Tables 4 and 6, we see that the BSP-TW DLB$_{accl}$ protocol is able to automatically select the number of processors to use in order to achieve better performance.

However, we note that the performance for the runs with low event granularities is still very much worse than sequential runs. This is attributed to the fact that the inherent synchronization overhead of 16 processors is still present using BSP-TW DLB$_{accl}$ even though the protocol uses only a small subset of the processors for computation. To verify this, we carried out experiments using BSP-TW DLB$_{ccl}$ with different fixed numbers of processors.

Table 8 shows the execution times for the Sematech data-sets for this set of experiments. The numbers in bold show the execution times using the number of processors determined by the BSP-TW DLB$_{accl}$ protocol in Table 7 for the configuration starting with 1 active processor. We see that the runs using the number of processors determined by BSP-TW DLB$_{accl}$ give the best performance achievable in most cases.

Comparing Tables 6 and 8, we can see the additional synchronization overhead in using a subset of processors out of the full 16 processors allocated. For example, the run for data-set 4 with event granularity 0 only takes 8.5 seconds to complete using BSP-TW DLB$_{ccl}$ on one processor (with only one processor allocated), as opposed to 122.0 seconds on BSP-TW DLB$_{accl}$ on 1 processor (16 processors allocated, but all simulation objects are initially mapped onto a single processor).

In order to verify the assumptions on (doubling/halving) the values of rollback ratio k_r and communication workload H when processors are added or removed, the corresponding

Table 8: Execution Times (sec.) for Sematech Data-sets using BSP-TW DLB$_{ccl}$ with P Processors

	Set1				Set2				Set3			
	Event Granularity				Event Granularity				Event Granularity			
P	0	10	100	1000	0	10	100	1000	0	10	100	1000
1	**40.3**	**96.6**	625.2	5913.5	**45.4**	**131.3**	887.1	8367.3	**51.7**	**169.1**	1231.6	11803.5
2	216.9	255.2	601.9	4107.3	208.8	269.5	704.7	5177.4	365.9	443.0	1132.3	8176.9
4	336.7	349.7	**551.9**	2498.6	308.3	330.8	**586.4**	3149.3	631.2	673.5	**1044.6**	5023.2
8	489.7	517.6	638.3	**2087.8**	480.7	517.5	689.3	2404.7	1156.0	1221.2	1487.8	**4832.7**
16	906.5	932.0	1107.9	2784.1	809.2	879.3	942.8	**2093.8**	2700.4	2680.5	3055.2	7570.8

	Set4				Set5				Set6			
	Event Granularity				Event Granularity				Event Granularity			
P	0	10	100	1000	0	10	100	1000	0	10	100	1000
1	**8.5**	**16.8**	95.4	880.7	**17.3**	62.6	450.3	4366.0	**22.2**	**68.8**	481.6	4609.7
2	38.9	43.5	**93.2**	591.5	115.3	**134.5**	362.9	2645.5	156.3	175.6	432.5	3001.3
4	94.3	94.6	127.9	439.0	183.4	194.0	328.9	1680.8	206.9	221.0	**370.0**	1837.3
8	170.5	184.9	198.9	**321.0**	274.4	279.7	**355.9**	**1227.3**	291.7	307.9	391.5	1279.7
16	302.7	301.1	338.2	632.0	630.9	600.2	757.3	1742.0	519.1	545.8	628.9	**1443.2**

values of k_r and H on different numbers of processors are recorded during the simulation runs. Tables 9 and 10 show the percentages of events rolled back and the communication workload for the Sematech data-sets using BSP-TW DLB$_{ccl}$. Although our assumptions do not produce an exact match on the variations of k_r and H for all six data-sets, they do reflect the general trend in the increasing values of k_r and H when more processors are added to the system.

6 RELATED WORK

A study reported in [Iqbal et al. 2002] examined the behaviour of different load-balancing algorithms when the number of processors is dynamically changed during the lifetime of a multi-stage parallel computation. The approach is to add or remove processors at different stages of the parallel computation based on the memory requirement. While the partitioning algorithms used in the study do try to minimize communication cost among processors, the decision to add or remove processors does not take into consideration the communication cost for adding or removing them.

In another related study reported in [Hill et al. 1998], the authors used regular check-pointing to save the states of a BSP computation. The computation is terminated when one or more of the processors used in the computation becomes heavily loaded with jobs from other users. The BSP computation is then restarted on another set of less loaded processors.

We note that this is a viable approach for dynamically varying the number of processors in the system without the overhead of synchronization cost due to the presence of inactive processors. In this case, the BSP-TW computation can be check-pointed, terminated, and restarted on a larger or

Table 9: Percentage of Events Rolled Back for Sematech Data-sets using BSP-TW DLB$_{ccl}$ with Fixed Number of Processors

	Number of Processors				
Data-set	1	2	4	8	16
1	0.0	13.4	11.7	17.1	30.6
2	0.0	4.8	7.5	15.5	21.0
3	0.0	16.4	17.9	31.5	58.3
4	0.0	9.7	17.9	30.5	37.6
5	0.0	8.4	9.9	18.4	45.0
6	0.0	11.9	12.5	18.5	34.5

Table 10: Communication Workload (sec.), H, for Sematech Data-sets using BSP-TW DLB$_{ccl}$ with Fixed Number of Processors

	Number of Processors				
Data-set	1	2	4	8	16
1	0.0	9.3	15.7	21.7	49.7
2	0.0	9.0	14.9	17.3	21.0
3	0.0	17.1	31.3	54.6	140.6
4	0.0	1.0	2.4	3.9	7.4
5	0.0	5.8	10.3	12.1	31.7
6	0.0	7.3	10.9	10.4	21.1

smaller set of processors.

7 CONCLUSION

In this paper, we have described a new BSP-TW DLB$_{accl}$ protocol that is capable of automatically varying the number of processors during the simulation runtime. We have demonstrated the effectiveness of this protocol in providing consistent performance on a set of real-world semi-conductor wafer manufacturing simulation models that have different event granularity characteristics.

Our study shows that optimal performance could be achieved if the underlying runtime library support efficient synchronization and communication for a subset of processors. Further work will be carried out in this area.

ACKNOWLEDGEMENT

This work is supported by Singapore Institute of Manufacturing Technology (SIMTech).

AUTHOR BIOGRAPHY

Malcolm Low received his master degree from Nanyang Technological University, Singapore, in 1999. He was an Associate Research Fellow with the Singapore Institute of Manufacturing Technology from 1997 to 1999. He is currently a DPhil student in the Oxford University Computing Laboratory. His research interests are in the areas of adaptive tuning and dynamic load-balancing of parallel discrete event systems.

REFERENCES

[Hill et al. 1998] Hill, J. M., S. R. Donaldson, and T. Lanfear. 1998. "Process Migration and Fault Tolerance of BSPlib Programs Running on Networks of Workstations". In *EuroPar'98*, ed. D. Pritchard and J. Reeve, Volume 1470, 80–91.

[Iqbal et al. 2002] Iqbal, S., G. Carey, M. Padron, J. Suarez, and A. Plaza. 2002. "Load Balancing with Variable Number of Processors on Commodity Clusters". In *Proceeding of the High Performance Computing Symposium 2002*. San Diego, California.

[Jain et al. 1999] Jain, S., C.-C. Lim, B.-P. Gan, and Y.-H. Low. 1999. "Criticality of Detailed Modeling in Semiconductor SupplyChain Simulation". In *1999 Winter Simulation Conference (WSC'99)*: Institute of Electrical and Electronics Engineers, Piscataway, New Jersey.

[Jefferson 1985] Jefferson, D. 1985. "Virtual Time". In *ACM TOPLAS*, Volume 7, 404–425.

[Low 2001] Low, M. Y. H. 2001. "Adaptive BSP Time Warp". In *Proceedings of the Fifth UK Simulation Society Conference (UKSim 2001)*, 14–20. Cambridge, UK.

[Low 2002] Low, M. Y. H. 2002. "Dynamic Load-Balancing for BSP Time Warp". In *Proceeding of the 35th Annual Simulation Symposium*, 267–274. San Diego, California.

[Marín 1998] Marín, M. 1998. "Discrete-Event Simulation on the Bulk-Synchronous Parallel Model". Ph. D. thesis, Oxford University.

[Sematech 1997] Sematech 1997. "Modeling Data Standards, version 1.0". In *Technical report, Sematech Inc*. Austin, TX78741.

[Skillicorn et al. 1997] Skillicorn, D., J. Hill, and W. McColl. 1997. "Questions and answers about BSP". *Journal of Scientific Programming* 6 (3): 249–274.

[Valiant 1990] Valiant, L. G. 1990. "A Bridging Model for Parallel Computation". *Communications of the ACM* 33:103–111.