The Power of SIMDs vs. MIMDs in Real-time Scheduling

Mingxian Jin, Johnnie W. Baker, and Will C. Meilander
Department of Computer Science
Kent State University, Kent, Ohio 44242-0001
Phone: 330-672-2430 Fax: (330) 672-7824
{mjin, jbaker, willcm}@cs.kent.edu

Abstract

SIMDs and MIMDs are the most important categories of computer systems for parallel computing in Flynn's classification scheme. Due to their higher flexibility in allowing processors to execute independently and their ability to use off-the-shelf microprocessors, the MIMD systems are generally favored and considered to be more powerful. In comparison, the SIMD systems are considered outdated. However, we observe that many intrinsic weaknesses of the MIMD systems are not fully recognized until they are compared while solving real-time scheduling problems. The SIMD systems have inherent advantages that MIMDs lack. In this paper, we compare SIMDs and MIMDs in real-time scheduling, e.g., scheduling for air traffic control. Two abstract parallel computation models, the ASC and BSP models that represent SIMDs and MIMDs respectively, are used in our discussion and analysis. We argue that the common belief that MIMDs have greater power than SIMDs is false. Our research shows that SIMDs are not outdated, as they offer tractable solutions for problems considered intractable with MIMDs. Rather, SIMDs are more efficient and powerful in some important application fields. They deserve more attention and considerations than they currently receive.

1. Introduction

Flynn's taxonomy of parallel computers, based on the numbers of instruction streams and data streams, has been widely used in the research literature of parallel processing since it appeared in 1972 [6]. SIMDs and MIMDs are the most important categories in Flynn's classification of computer. Most early parallel computers had a SIMD-style design. They had a central controller that broadcasts commands to multiple processing elements. The commands are executed synchronously on the set of processors. On the other hand, with the MIMD architectures, each of processors executes programs at their own pace. Processors exchange information using shared memory or messages passing. Due to their high flexibility in using off-the-shelf micro-processors, MIMD computers are considered cheaper to build and more modern architecturally than SIMD computers. They dominate today's market and SIMD computers have all but vanished. Many people in the field are pessimistic about the future of SIMD computers [1,5,13,14].

However, as we investigated real-time scheduling problems, in particular for some problems in air traffic control, we observed that claims that MIMDs are more powerful than SIMDs are false. For many real-time scheduling problems, efficient polynomial time SIMD solutions can be obtained, while it appears impossible to obtain a polynomial time MIMD solution. This suggests that SIMDs are not outdated since MIMDs have inherent weaknesses such as synchronization costs, communication limitations, mutual exclusion to access shared resources, serializability, data concurrency, etc. which do not occur with SIMDs. The impact of all these problems was not fully recognized until MIMDs were used to try to solve real-time problems where these weaknesses become critical. In this paper, we will compare SIMDs and MIMDs for real-time problems. An example is given of a problem with a polynomial time solution using a SIMD but for which there is no polynomial time solution using MIMDs. (In this paper, we assume $P \neq NP$; hence, no NP-complete problem has a polynomial time solution.) We will discuss the currently accepted superior power of asynchronous models, based on a sequence of simulations connecting the ASC and BSP models. We will discuss the theoretical and practical reasons for NP-hardness when MIMDs are used to solve real-time scheduling problems.

This paper is organized as follows. Section 2 introduces real-time scheduling terms and the parallel models BSP and ASC used here for MIMD and SIMD, respectively. Section 3 presents a real-time scheduling example that is fundamental to this paper. Section 4 combines existing simulations and one new simulation to produce a simulation between the BSP and ASC models in order to better compare these models. Section 5 describes advantages of SIMDs over MIMDs in real-time

0-7695-1573-8/02/$17.00 (C) 2002 IEEE

scheduling. Section 6 analyzes the reasons that prohibit an efficient MIMD solution. Section 7 is the conclusion.

2. Real-time scheduling and parallel models

2.1. Real-time scheduling

Real-time scheduling differs from classic scheduling in that tasks must meet specified timing requirements. A real-time system executes tasks to ensure not only their logical correctness but also their temporal correctness. If the timing deadlines are not met, the system fails, no matter how accurately the tasks are executed.

As defined in [19], a real-time *task* is an executable entity of work that, at minimum, is characterized by a worst-case execution time and a time constraint. A *job* is defined as an instance of a task. A real-time task can be *periodic,* which is activated (released) regularly at a fixed rate (period); a*periodic*, which is activated irregularly at some unknown and possibly unbounded rate; or *sporadic*, which is activated irregularly with some known bounded rate. Typically, real-time scheduling can be static or dynamic. For *static scheduling,* the scheduling algorithm has complete knowledge a priori about all incoming tasks and their constraints such as deadlines, computation times, shared resource access, and future release times. In contrast, in *dynamic scheduling*, the scheduling algorithm only has knowledge about the currently active tasks, but it does not have knowledge about future tasks prior to their arrival. The event-driven schedule produced by a dynamic scheduling algorithm therefore changes over time.

Real-time scheduling can be executed on a uniprocessor or a multiprocessor. (Since a multiprocessor is usually regarded as a parallel computer that has multiple processors with multiple instruction streams, we will use the words "multiprocessor" and "MIMD" interchangeably.) It has been shown that there exist optimal scheduling algorithms for a uniprocessor system. An *optimal* scheduling algorithm is one that may fail to meet a deadline only if no other scheduling algorithm can meet it [20]. For example, the *earliest deadline first* (EDF) algorithm is optimal for scheduling a set of independent real-time tasks on a uniprocessor [19]. As real-time systems become larger and tasks become more sophisticated, real-time scheduling has become much more dependent on parallel systems. Unfortunately, with the multiprocessor system, optimal scheduling algorithms have not been found for most problems. Complexity results have established that almost all real-time scheduling problems on multiprocessors are NP-hard [7,19,20]. Even though there are some heuristic scheduling algorithms for multiprocessors, they assume restricted conditions and work only under special circumstances (see examples in [18]).

Our studies are based mainly on typical real-time scheduling problems, and in particular, on real-time task scheduling for air traffic control. The tasks have a common feature that virtually all data in the problem must be stored in a shared database. Each task is executed on data from the shared tables. Our observations may be applied to a large number of other real-time systems, as most real-time tasks conceptually involve database type operations on data from a shared database source.

2.2. Parallel computational models

Parallel computation models range from very abstract to very concrete. Most models (e.g., PRAM, 2D Mesh) have two versions: a synchronous version and an asynchronous version. An important goal of this paper is to show that probably the most important property of a parallel model is whether it is synchronous or asynchronous.

Both SIMDs and MIMDs have their particular characteristics and advantages. All processors of a SIMD are controlled by a central unit and operate in lockstep or synchronously. The SIMD model has advantages of being easily programmed, cost-effective, highly scaleable, and especially good for massive fine-grain parallelism [13,15]. On the other hand, each of processors of a MIMD has its own program and executes independently at its own pace; i.e., asynchronously. The MIMD model has the advantages of high flexibility in exploiting various forms of parallelism, ease in using current high-speed off-the-shelf microprocessors, and being good for coarse-grain parallelism [1,14]. To make our comparison more clear and specific, we use two abstract models, ASC and BSP, which identify the essential properties we assume for SIMDs and MIMDs, respectively.

The ASC model can be characterized as an associative SIMD. It consists of an array of processing elements (PEs) connected by a bus and an instruction stream processor (IS) that broadcasts commands and data to all of the PEs on the bus. Each PE has its own individual memory and can perform all the usual local operations of a sequential processor other than issuing instructions. Additionally, each PE can only access its own memory. Each PE can become active or inactive, based on the result of a data test. An active PE executes the instructions issued by the IS, while an inactive PE listens to but does not execute the instructions. Assuming that the word length is a constant, ASC supports several important constant time operations, namely broadcasting, global OR/AND, maximum/minimum, and associative search which identifies the PEs whose data values match the search pattern (called responders) or do not match (called non-responders) [11]. Two parallel architectures that fully support the ASC model (in hardware) are the STARAN and the ASPRO [2,3]. The ASC model is a

restriction of the MASC model, a multiple SIMD model, to one instruction stream. MASC supports both data and control parallelism. A detailed description of MASC is given in [16] and further information about the properties of ASC can be inferred from [4,11].

The BSP (Bulk-Synchronous Parallel) model was introduced in [21] to overcome the shortcomings of the classic PRAM model and to make a bridge between abstract algorithms and realistic architectures for general-purpose parallel computation. A BSP program consists of a sequence of parallel *supersteps*. A superstep is a combination of local computation steps and message transmissions. Each superstep is followed by a global check (barrier synchronization) to wait for all processors to finish the current superstep before proceeding to the next superstep.

The BSP model is an abstract MIMD model since the processors can execute different instructions concurrently. It is loosely synchronous at the superstep level, in contrast to the tight synchrony in a SIMD model. The processor interaction mechanism in the BSP model is not specific and allows either shared variables or message passing. BSP therefore is a reasonable model for most current MIMD machines. Clusters and SMPs [14] are currently two of the most popular architectural variants of MIMDs and are captured well by the BSP model.

As stated in [22], the BSP model is expected to have universal efficiency for general-purpose parallel computation over special-purpose models. It is claim that "special-purpose machines have no major advantage since general-purpose machines can perform the same functions as fast" and additionally can use "high-level programming languages and transportable software". Conceptually, since BSP is considered to be a bridging model, an efficient optimal BSP simulation can obtained for any special-purpose model and used to transfer its algorithms to cost efficient ones on a general-purpose model.

3. A real-time scheduling example

The motivation for this section is the results reported in [12,17] on real-time scheduling for the air traffic control (ATC) problem using an associative SIMD computer. A polynomial time algorithm is given in [12] for the ATC scheduling problem using the ASC model and static scheduling. In contrast, the air traffic control scheduling problem on a multiprocessor and many other similar scheduling problems are either known or believed to be NP-hard. In particular, it is shown in [7] that a set of real-time tasks that have varied computation times or shared resources cannot be scheduled on a multiprocessor in polynomial time. In order to study this phenomenon in greater detail without getting involved in the various aspects of the ATC scheduling problem, we present a simple real-time scheduling problem. This allows us to compare the power of the SIMD model and the MIMD model using real-time scheduling problems.

Consider an array A of size of $n \times m$. Let a_{ij} denote the element in the ith row and jth column where $1 \le i \le n$ and $1 \le j \le m$. All values in A are updated every D time units. The problem is to complete a set of periodic tasks $T_1, T_2, ..., T_k$ within a deadline D, where each T_i performs the same (polynomial time) operation on some or all rows of A. Such a task is a set operation and includes multiple jobs, each of which is to perform the operation specified by the task on the specified rows. The deadline D for completing all tasks is the same as the period for updating the data in the array A.

In sequential processing, the computational costs of the tasks are $C_1, C_2, ..., C_k$. Each C_i is the product of the number of the rows r_i that task T_i involves and the time c_i required to perform each job (i.e., the specific operation on each row). For example, T_1 could be a task adding the values in column 1 to column 2 for those rows with an even row number. T_2 could be a task finding all medians of individual rows for all rows. So T_1 consists of $n/2$ jobs of adding a_{i1} to a_{i2} ($1 \le i \le n$ with i even). T_2 consists of n jobs each of which is finding the median of $a_{i1}, a_{i2}, ..., a_{im}$ ($1 \le i \le n$), respectively. If the time for adding two numbers is c_1 and the time for finding the median of m elements in a row is c_2, the computation costs C_1 of T_1 is $n/2 \times c_1$ and C_2 of T_2 is $n \times c_2$. According to scheduling theory [19], to schedule such a set of tasks on a uniprocessor using non-preemption, the following condition must be met:

$$\frac{r_1 c_1 + r_2 c_2 + ... + r_k c_k}{D} \le 1 \qquad (1)$$

In practice, n and r_i could be very large compared to D and c_i. When this is the case, it could be impossible to meet condition (1) and finish the tasks within the deadline using a uniprocessor. Parallel processing seems to provide the only solution.

Due to the obvious data-intensive nature of this computation, a SIMD can be significantly more efficient than a MIMD when the array size is large. With the massive parallelism supported by SIMDs, we may have thousands of PEs available with each row residing in one PE. Since the same operation is performed on all the involved rows, we process all the rows simultaneously. When using ASC for this example, all jobs for each of the k tasks are processed in parallel, no matter how many rows are involved. Therefore, condition (1) now reduces to the following:

$$\frac{\sum_{i=1}^{k} c_i}{D} \le 1 \qquad (2)$$

Compared with (1), condition (2) is much easier to satisfy. In fact, problems similar to this example occur in the air traffic control problem frequently and can be solved efficiently on the ASC model [12,17].

Next, we consider solving this problem with a MIMD. Unfortunately, since all tasks have varied computation times, there is no known polynomial algorithm using a multiprocessor to schedule this set of tasks. This is shown in the following theorem of Gary and Johnson [7]. Let T be a set of tasks, $m \in Z^+$ (the set of positive integers), length $l(t) \in Z^+$ for each $t \in T$, and a deadline $D \in Z^+$. The problem of whether there is an m-processor schedule for T that meets the overall deadline D is NP-complete for $m \ge 2$, assuming not all tasks have the same length. (Here, m-processor means a MIMD with m processors). This theorem applies directly to our example.

The example presented in this section is more difficult than would be the case if all the tasks were independent and did not share resources. Since all tasks share the same data source (a data file or a database), the scheduling problem for this set of tasks with the added requirement that is performed on a multiprocessor is NP-hard. Either varied task computation times or resource sharing causes this scheduling problem to be NP-hard on a multiprocessor, as shown in [7,19,20].

4. Advantages of associative SIMDs

The ASC model (or an associative SIMD) has certain features that make it possible to efficiently solve real-time problems such as the air traffic control. These features can be used by associative SIMD systems to solve other similar real-time scheduling problems since they can share the same features that we discuss below.

a) Eliminate the expensive synchronization costs in MIMDs

It has been generally believed that an asynchronous MIMD model is more powerful than a synchronous SIMD model because MIMDs have the flexibility to allow every processor to compute at its full speed and without waiting for others to proceed. This seems to imply that, if a problem can be solved efficiently in a synchronous model, then it should be possible to efficiently execute the same algorithm in an otherwise identical asynchronous model. However, our example shows that this is not true, at least for real-time scheduling problems. Synchronization costs are extremely expensive and can cause significant time delays.

b) Locate data by content rather than by address

Since data is accessed by content rather than address, sorting and indexing are normally eliminated in ASC algorithms. The complexity of the software is significantly reduced and the real-time deadline is much more easily met with SIMD algorithms than with MIMD algorithms. Normally, with large MIMD software systems and even for some MIMD algorithms, a considerable amount of time is used for sorting and indexing the data.

c) Use wide memory bandwidth to access a large amount of data simultaneously

The traditional memory access bottleneck does not exist in the associative SIMDs. Since all data are stored in the local memory of individual PEs, data is loaded into the PEs in parallel for computations. A data item can be broadcast to one or more PEs in one step. This is in stark contrast to the shared-memory MIMD systems that have to distribute shared data to each of the processors. A SIMD also eliminates the network congestion prevalent in distributed memory MIMD systems due to the continual massive data movement required to update local copies of shared data in different processors or to provide needed data to the different processors. While it might appear that the same cost would be incurred in the moving data to different PEs in the ASC model, this is not true. Normally, most of the data needed by different processors is broadcast by the IS to all processors. Additionally, if a massive data relocation is required (e.g., in a sort or FFT algorithm), then the synchronous interconnection network, such as 2D mesh, of the associative SIMD would be used. Unlike the "store and forward" types algorithms for asynchronous networks, the algorithm for this data relocation will move the data using a fixed, worst case number of synchronous data movements that avoids congestion and has predictable time requirements. Moreover, on most ASC applications, the use of the interconnection network is minimal or non-existent. The usual use of sorting to keep the data organized and easily accessible, and make items easy to locate is unnecessary in ASC, since desired data are normally located using a content-addressable associative search and then broadcast to all PEs that need the data.

d) Use massive parallelism for intensive data-parallel computations

SIMDs support massive parallelism and data-parallelism more efficiently than MIMDs. In SIMDs, the broadcast feature minimizes (and often eliminates) any data movement between PEs, which are handled using network or shared memory in MIMDs. Typically, any large data relocations that occur in SIMDs are programmed to move synchronously, avoiding the problems associated with procedures such as store-and-forward, pipelines, etc.

5. Simulations between BSP and ASC

Asynchronous parallel models have continued to gain status over the years and an asynchronous model is now commonly regarded to be more powerful than the resulting model when restricted to being synchronous. For example, in Flynn's classification scheme, MIMDs are generally regarded as the most general and powerful, as indicated by the following statement: "Theoretically, any parallel algorithm can be executed efficiently on the MIMD model" [1]. The abstract BSP model is a reasonable model for most current MIMD architectures including SMPs and clusters, which are the two most common MIMD architectures in today's market. In this section, we consider a simulation between the BSP and ASC models to further analyze the power of MIMDs. Our investigation shows that the claim that BSP is a "bridging model" for all parallel computation does not seem to be justified for the associative SIMD computer (or the ASC model).

Theoretically, not only is simulation between parallel models an efficient way to transfer algorithms from one model to another model, but also a good method to examine the comparative power of the models. If one model can simulate a second model efficiently, any algorithm on the simulated (second) model can be transferred to an algorithm on the simulating model using the simulation algorithm. The running time of a transferred algorithm is bounded by the product of the algorithm time on the simulated model and the simulation time.

We next recall a series of previously established simulations from BSP to PRAM, and then from PRAM to ASC. We use $M(p)$ to denote a model M with p processors.

- BSP → Asynchronous PRAM.

It is shown in [22] that there are simulations between BSP and PRAM either with exclusive memory access (EPRAM) or with concurrent memory access (CPRAM). The simulations have expected optimal efficiency. When a simulation has *optimal efficiency*, execution of an algorithm on the simulated model or through the simulation of the algorithm on the simulating model will result in the asymptotically same amount of work. If the simulation is randomized, the efficiency is said to be *expected efficiency*. Giving a good randomization function, Valiant [22] has proven that BSP(p) can simulate a EPRAM(v) with expected optimal efficiency if $v \geq p\log p$. A BSP(p) can simulate a CPRAM(v) with expected optimal efficiency if $v \geq p^{1+\varepsilon}$ (ε is a positive constant). Clearly, any polynomial algorithm running on asynchronous PRAM can be transferred to BSP in expected polynomial time.

- Asynchronous PRAM → synchronous PRAM

The PRAM model referred in [22] is assumed to be an asynchronous PRAM, in which each processor executes its own program. It supports the power of asynchronous MIMD machines. Asynchronous PRAM variants have been studied by many researchers in the past (see [1,8]). The best known is Gibbons' asynchronous PRAM [8]. Gibbons has shown that a time t algorithm on EPRAM(p) can be simulated by an asynchronous PRAM(p/B) running in time $O(Bt)$, where B is the time required to synchronize all the processors used in the algorithm and is a function of the number of the processors. Usually $B(p) \leq p$ and mostly $B(p) = O(\log p)$. This simulation is a work-preserve simulation, which means that the amount of work through the simulation remains asymptotically the same. We notice that the costs assigned to the operations for this asynchronous PRAM model do not seem to allow the exorbitant costs resulting from synchronization, shared memory contention, etc., which have been observed in real time scheduling with MIMDs. These observations raise some questions concerning the validity of the costs assigned to this model.

- EPRAM → CPRAM

It has been long known that an EPRAM(p) can simulate any variant of a CRCW PRAM(p) or CPRAM(p) in $O(\log p)$ [9]. The simulation is either randomized or deterministic.

- CPRAM → ASC

A CRCW PRAM(p) or CPRAM(p) can deterministically simulate ASC(p) in $O(\log p)$. This is because a processor of a synchronous PRAM can simulate each of the PEs of ASC. One of the PRAM processors also plays the role of the IS. It can be shown that the upper bound for the constant time operations of ASC (e.g., broadcast and reduction operations) in PRAM is $O(\log p)$.

Combining the above sequence of simulations, we obtain a polynomial time simulation of ASC by BSP with a high probabilistic expectation. Therefore, this simulation provides a polynomial time transformation of ASC algorithms to BSP algorithms with a high probabilistic expectation. Theoretically, this should also be applicable to a real-time scheduling algorithm. Considering the fact that the BSP model has been claimed to be a bridging model, it should be able to execute any real-time scheduling algorithm implemented on ASC efficiently. However, from our analysis of the real-time scheduling example in Section 3, this is not true, at least, using worst case timings. The fact that there can be no polynomial time worst-case BSP simulation of ASC is surprising, in view of the existence of a polynomial simulation of ASC by BSP. This suggests the possible need for a closer reexamination of each of the links in the preceding sequence of simulations. Average case timings are not applicable for mission critical real-time scheduling problems with hard deadlines. It appears questionable to determine the power of a model using only average case or randomized analysis. In this sense, the power of SIMDs has been significantly underestimated.

6. Difficulty of an efficient MIMD solution

In this section, we consider why solving a real-time scheduling problem using a MIMD is usually NP-hard, thus showing why a polynomial time solution for many real-time scheduling problems using a MIMD is intractable.

Given a relatively small environment with a small data size, real-time scheduling is solvable on a uniprocessor. Some optimal scheduling algorithms such as the EDF algorithm have been developed. Whether a set of real-time tasks is schedulable or not can be predicted using a uniprocessor. However, a real-time scheduling problem may become NP-hard when one of the following conditions is added: multiprocessors, shared resources, non-preemption, varied release times, precedence constraints, etc. [7,20].

We now discuss why some difficulties occur whenever a MIMD system is used for scheduling problems. This will lead to intractability of most scheduling problems requiring a solution using a MIMD.

- Multiprocessors

When multiprocessors are used, tasks must be partitioned and assigned to individual processors. Since the PARTITION problem of a finite set is a basic NP-complete problem, so is the problem of assigning a set of real-time tasks with deadlines to a multiprocessor (The detailed proof can be found in [7]). MIMDs have multiple instruction streams; therefore, tasks must be partitioned to allow concurrent executions. Consequently, most scheduling problems for multiprocessors are NP-hard [20].

On the other hand, even though SIMDs also have multiple processing elements, they all execute using a single instruction stream, and task partitioning is unnecessary. When ASC is applied to schedule a set of real-time tasks, for example in air traffic control, it can simply make use of a static sequential scheduling algorithm due to its single instruction stream. Based on the worst-case time of each task, the algorithm reserves a time slice for each task. Each task may consist of a multiplicity of jobs that execute the same instruction on a set of data. Thus, a statically designed scheduling algorithm can be developed on ASC to accommodate one or many jobs within a task. Moreover, the task takes polynomial time.

- Shared resources

Shared resources usually cause NP-hardness in real-time scheduling problems [7,20]. In some real-time systems, for example air traffic control, a real-time database is required as a common resource accessible by all tasks. Generally, resource sharing can be realized by mutual exclusion constraints. Unfortunately, there is no optimal solution for scheduling a set of real-time tasks with mutual exclusion constraints because mutually exclusive scheduling blocks have different computational times that cause NP-hardness in the same way as shown earlier using the partition problem [20].

In particular, MIMDs either with shared memory (e.g., SMPs) or distributed memory (e.g., clusters) cannot efficiently handle mutual exclusion for shared resources. Additionally, SMPs suffer from limited bus bandwidth for transmission of data between the shared memory and individual processors. SMPs have to handle memory and cache coherency to maintain data consistency. Clusters also develop significant costs for data transmission as well as for management of data coherence. However, because of its single instruction stream, the ASC model can avoid accessing shared resources. This fact eliminates the need for mutual exclusive access to shared resources. When data is stored in the ASC memory, simultaneous access is possible since each PE is only accessing its private memory, thus effectively avoiding the shared resource problem.

- Non-preemption

Although non-preemptive scheduling is easier to implement and a lower overhead is incurred, an efficient non-preemptive scheduling algorithm on MIMD systems is almost impossible [7,20]. Most scheduling algorithms on multiprocessors use preemption [18,19,20]. However, when preemption on MIMDs is allowed, one must consider the difficulty of predicting overhead, asynchronous execution of tasks distributed on different processors, synchronization of task execution after preemption, etc. Preemption may also worsen task migration and load balance among processors commonly existing in MIMD systems. Inclusion of these difficulties will substantially increase the complexity of a solution to the scheduling problem.

- Varied release times

If all tasks have the same release time, feasibility of a scheduling algorithm can be evaluated in polynomial time. If tasks have varied release times, the scheduling problem becomes NP-hard [7]. In the MIMD environment, there is no global clock to provide an accurate time stamp and measure elapsed time. If each processor operates independent, it is impractical to continuously synchronize all task executions after every step (or after every few steps). In addition, unpredictable communication delays may also make task time stamps inconsistent. Therefore, the inherent asynchronous nature of MIMDs is unlikely to accurately maintain the same timetable for a set of real-time tasks even though they have the same release time. In contrast, in a SIMD, all processors run synchronously. They have absolute time stamps by sharing a global clock. If release times are given, they will remain unchanged.

- Precedence constraints

The existence of precedence constraints is another reason for NP-hardness in real-time scheduling. Compared to SIMDs, MIMDs have a small number of processors. Precedence constraints may require waiting for higher priority tasks or preemption of lower priority tasks. When tasks have to meet a certain order of execution, MIMDs have to handle problems such as idle processor time, expensive overhead, task migration, load balancing, communication delays, etc. The scheduling problem becomes intractable in most cases[7]. In contrast with MIMDs, the ASC schedules real-time tasks for air traffic control statically [12,17]. It takes advantage of its synchronous data parallel capabilities and uses predefined precedence to assure enough time to complete worst-case time tasks. It statically maintains the order of task execution.

Based on the inherent weakness of MIMDs, most real-time scheduling problems become intractable when the requirement that they be solved on a multiprocessor is added. On the other hand, our example shows if this problem is altered to permit an associative SIMD solution, it may be tractable.

7. Conclusions

In this paper, we have shown that that the common belief that MIMDs (or the abstract BSP model) are more powerful than SIMDs (or the associative ASC model) is unjustified and ignores the many intrinsic weaknesses of MIMDs. In particular, an example is given of a problem that can be solved in polynomial time on an associative SIMD, but cannot be solved in polynomial time on a MIMD. These problems with MIMDs were not fully recognized until their ability to solve real-time scheduling problems were considered. MIMDs have repeatedly failed to meet the Federal Aviation Administration air traffic control requirements. One notable example is the ten-year effort to develop the Automated Air Traffic Control System (AAS). The AAS program was canceled in June 1994 after expenditure of several billion dollars [17]. On the other hand, the ability of associative SIMDs to successfully handle the air traffic control problem have been demonstrated [12,17].

The BSP model was initially proposed for general-purpose parallel computation and was intended to provide a common and efficient model for all parallel systems, However, as we have observed, BSP models SIMDs poorly and is likely to create algorithms run very inefficiently on a SIMD.

Our research has shown that SIMDs are not outdated, as many professionals in parallel computation currently believe. They are efficient and powerful enough to provide efficient solutions to problems that are considered intractable for MIMD systems. Moreover, considering the SIMD's advantages of simple programming styles and simple hardware implementations, it obviously deserves more attention and utilization if we want to solve today's real-time problems.

References

[1] Selim G. Akl, The Design and Analysis of Parallel Algorithms, Prentice Hall, New Jersey, 1989

[2] K. Batcher, STARAN Parallel Processor System Hardware, *Proc. Of the 1974 National Computer Conference* (1974), pp. 405-410

[3] Loral Defense Systems-Akron, *ASPRO-VME Hardware and Architecture*, June, 1992

[4] J. W. Baker and M. Jin, "Simulation of Enhanced Meshes with MASC, a MSIMD Model", in *Proc. of the 11th International Conference on Parallel and Distributed Computing Systems*, pages 511-516, November 1999 (Unofficial version: http://vlsi.mcs.kent.edu/~parallel/ papers/ baker99b.pdf)

[5] T. Blank, J. Nickolls, "A Grimm Collection of MIMD Fairy Tales", *Proc. of the 4th Symp. on the Frontiers of Massively Parallel Computation*, pp. 448-457, 1992

[6] M. Flynn, "Some computer organizations and their effectiveness." *IEEE Transactions on Computers*, pp. 948-960, Sep., 1972

[7] M. R. Garey and D. S. Johnson, *Computers and Intractability: a Guide to the Theory of NP-completeness*, W.H. Freeman, New York, 1979, pp.65, pp. 238-240

[8] P.B. Gibbons, "A More Practical PRAM Model", *Proc. of 1st ACM Symp. on Parallel Algorithms and Architectures*, pp.158-168, June 1989

[9] T. Harris, "A Survey of PRAM Simulation Techniques", *ACM Computing Surveys*, Vol. 26, No. 2, June 1994, pp. 187-206

[10] K. Jeffay, D. F. Stanat, and C. U. Martel, "On Non-Preemptive Scheduling of Periodic and Sporadic Tasks", *Proc. of the 12th IEEE Real-Time Systems Symposium*, San Antonio, TX, December 1991, pp. 129-139

[11] M. Jin, J. Baker, and K. Batcher, "Timings of Associative Operations on the MASC model", *Proc. of the Workshop in Massively Parallel Processing of IPDPS '01,* San Francisco, CA, April, 2001 (Unofficial version: http://vlsi.mcs.kent.edu/~parallel/papers/jin01.pdf)

[12] W. C. Meilander, J. W. Baker, and J. L. Potter, "Predictability for Real-time Command and Control", *Proc. of the Workshop in Massively Parallel Processing of IPDPS '01,* San Francisco, CA, April, 2001 (Unofficial version: http://vlsi.mcs.kent.edu/~parallel/papers/ meilander01.pdf)

[13] B. Parhami, "SIMD Machines: Do They Have a Significant Future?" *Report on a Panel Discussion at The 5th Symposium on the Frontier of Massively Parallel Computation,* McLean, LA, Feb., 1995

[14] Gregory F. Pfister, *In Search of Clusters*, *2nd Edition,* Prentice Hall, New Jersey, 1998

[15] J. L. Potter, *Associative Computing: A Programming Paradigm for Massively Parallel Computers,* New York; Plenum Press, 1992

[16] J. L. Potter, J. W. Baker, S. Scott, A. Bansal, C. Leangsuksun, C. Asthagiri, "ASC: An Associative-Computing Paradigm", *Computer*, 27(11), 19-25, 1994

[17] L. Qian, *Complexity Analysis of an Air Traffic Control System Using an Associative Processor*, Master's Thesis, Kent State University, 1997

[18] K. Ramamritham, J. A. Stankovic, and W. Zhao, "Distributed Scheduling of Tasks with Deadlines and Resource Requiremnets." *IEEE Trans. On Computers*, Vol. 38, No. 8, August 1989, pp.1110-1123

[19] J. A. Stankovic, M. Spuri, K. Ramamritham and G. C. Buttazzo, *Deadline Scheduling for Real-time Systems*, Kluwer Academic Publishers, 1998

[20] J. A. Stankovic, M. Spuri, M. Di Natale, and G. Buttazzo, "Implications of Classical Scheduling Results for Real-time Systems", *IEEE Computer,* June, 1995

[21] L. G. Valiant, A Bridging Model for Parallel Computation, *Communication Of ACM*, Vol. 33, No. 8, 1990, pp. 103-111

[22] L. G. Valiant, General Purpose Parallel Architectures, J. van Leeuwen ed., *Handbook of Theoretical Computer Science*, Vol. A, The MIT Press/Elsevier, New York, 1990, pp. 943-971