Importance of SIMD Computation Reconsidered

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

Abstract

In this paper, SIMD and MIMD solutions for the real-time database management problem of air traffic control are compared. A real-time database system is highly constrained in a multiprocessor and access to the common database must be made to a limited number of data elements at a time. This MIMD database access is contrasted with the comparable SIMD common database access, which can be several hundred times greater. This is true because the SIMD can simultaneously access thousands of pertinent records instead of the limited number in the MIMD. A relatively simple example is given of a problem that has a polynomial time solution using a SIMD but for which a polynomial time solution using a MIMD is normally impossible. The fact that SIMDs can support a polynomial time solution for the Air Traffic Control problem but this problem is normally considered to be intractable for multiprocessors argues against the common belief that MIMDs have greater power than SIMDs. SIMDs are more efficient and powerful for some critically important application areas.

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 design [21]. An associative processor (AP) is a SIMD machine with additional hardware enhancements that support constant time broadcasting, associative searches, and AND/OR and maximum/minimum reductions (assuming word length is a constant) [11, 25]. Since it is easy to see that a SIMD can simulate an AP in a low order polynomial time using its interconnection network, in this paper we use associative processor (AP) or SIMD interchangeably. Likewise, multiple processors (MP) and MIMD will be used interchangeably. APs have a single instruction-processing unit that broadcasts commands to the set of processing elements [22]. The commands are executed simultaneously on the set of processors. On the other hand, with the MP, each of the processors can independently execute programs using its individual instruction processing unit. MP processors exchange information using shared memory or asynchronous message passing. Because they can use off-the-shelf microprocessors, MPs are considered less expensive and more "state of the art" than APs. MPs dominate today's market. As a result of the assumed advantages of MPs, APs have all but vanished, and many people in the field are pessimistic about the future of such machines [1,5,13,14].

However, as we investigate real-time database problems, in particular for some problems in command and control (C&C) such as air traffic control (ATC), we observe that claims that MPs are more powerful than APs are untrue and need to be re-evaluated. For many real-time problems, efficient polynomial time AP solutions can be obtained, while extraordinary efforts to date have been unable to design a polynomial time MP solution for the same problems. MP systems require dynamic scheduling and are unable to support a static schedule for problems like Air Traffic Control. However, because thousands of person years have been devoted to searching for an efficient MP solution to problems like ATC, it is generally believed that these problems are truly NP-hard.

A close look at the ATC problem reveals no reason why a sequential solution to the problem should not execute in polynomial time. Nevertheless, Stankovic in discussing distributed systems states that the problem of determining an optimal schedule even in a multi-processor is known to be NP-hard [19]. Virtually all real-time problems today include a software solution to one or more dynamic scheduling problems. Since real-time scheduling problems using MPs cannot normally be solved using a static scheduling algorithm [19], it is doubtful that a static scheduling algorithm exists for MPs for C&C problems like ATC.

Predictability of system performance is the key to success. If performance can be statically predicted, a polynomial time solution is assured. With the AP a static off-line schedule can be developed for the ATC problem,

which allows the entire problem to be solved in polynomial time, and within system deadline time [25, 26] . Similar performance is expected for other real-time database problems. Until system performance can be reliably predicted, as in the AP solution for the worst-case environment, an adequate solution for problems like ATC is improbable [26]. MPs have inherent weaknesses because they have to deal with difficult problems such as synchronization costs, mutual exclusion of access to shared resources, data concurrency management, difficulty in achieving serializability, load balancing, etc [7,19,20]. These same problems are normally non-existent in AP systems. The full impact of these inherent MP weaknesses becomes evident when their abilities to manage real-time databases are considered.

In this paper we will compare APs and MPs using a few real-time problems. An example is given of a problem with a polynomial time solution using an AP but for which MP solutions are not polynomial. (In this paper, we assume $P \neq NP$; hence, no NP-complete problem has a polynomial time solution.) We will discuss reasons that real-time scheduling problems for MPs are NP-hard.

It is generally believed that a MIMD can efficiently simulate a SIMD and hence provide a MIMD solution to a problem that is as efficient as a SIMD [1]. However, the usual MIMD simulation of a SIMD cannot provide efficient, constant time execution of broadcasting, associative searches, and AND/OR and maximum/minimum reductions. These operations can realistically be executed in constant time in an AP [11]. The use of these constant time AP operations is ubiquitous and pervasive in the AP solution of the ATC problem [25, 26]. However, the continuous use of these "AP constant time operations" in the AP solution would be much less efficient for a MIMD. While the MIMD may be able to execute the polynomial AP solution for the ATC, it is unreasonable to expect that this solution would be able to meet real-time deadlines. Moreover, the usual distributed storage of a database in a MIMD for real-time problems prevent MIMDs from directly simulating AP solutions. For real-time problems with deadlines, MIMD solutions normally use a task scheduling algorithm to ensure deadlines are met. As noted earlier, most scheduling problems for MPs are NP-hard. While additional hardware could be added to a MIMD to support constant time execution of the AP constant time operations so that it could simulate SIMD solutions more efficiently, it would seem more practical to use an AP to execute this solution instead.

Many may consider the AP to be a machine that has not been actively used in the real-time environment. We here dispel that thought. The first associative processor was installed in the Knoxville ATC terminal in 1969 where it performed the functions of automatic radar tracking with secondary radar backup, conflict detection and resolution, terrain avoidance and automatic voice advisory to uncontrolled aircraft. The processor had 128 PEs, and was programmed with less than 2,800 instructions. An editorial "Back to the Future" by R. Smith in Dec. 1995 "Flying Magazine" discussed the systems performance from a pilot's point of view, and asked what happened to the system. Reference [17] discusses the rationale for stopping the program. Several STARAN machines were delivered with 1,024 or 1536 PEs. The U S Navy used ASPRO in the E2C Airborne Early Warning system (part of the Navy's ATC system). ASPRO, developed in 1977, had 2,000 PEs in .42 cu ft of space, and improved tracking from a capability of 200 tracks to 2,000 tracks. The tracking program required only 0.76 seconds per 10 second period. The Navy installed more than 130 ASPROs, and found many other uses for ASPRO.

This paper is organized as follows. Section 2 introduces the parallel MP and AP architectures. Section 3 introduces concepts of real-time computing. Section 4 discusses several real-time scheduling problems and shows some examples. Section 5 examines one of the most critical aspects of real-time data processing, namely concurrency management and its limits on performance. Section 6 considers data bandwidth and its effect on system performance in both the MP and AP architectures. Section 7 outlines advantages of the AP and the reasons that make an efficient MP solution to real-time problems difficult or impossible. Section 8 summarizes the conclusions reached.

2. Parallel computation architectures

APs and MPs each have their particular characteristics and advantages. All processors of an AP operate synchronously, and are controlled by a single instruction stream. The AP acts on a set of data with each instruction and has advantages of being easily programmed, cost-effective, highly scaleable, and especially good for massive fine-grain parallelism [13,15]. Each of the processors making up the MP has its own program and instruction processor and executes independently or asynchronously at its own rate. The MP has the advantages of high flexibility in exploiting various forms of parallelism, availability by using current high-speed off-the-shelf microprocessors, and being good for coarse-grain parallelism [1,14].

An AP consists of an array of processing elements (PEs) connected by a bus and an instruction stream processor (IS) that can broadcast commands and data to all of the PEs on the bus. Each PE, essentially an ALU, uses its own individual data memory and can perform all the usual local operations of a sequential processor other than issuing instructions. Each PE can become active or inactive, based on the program or data. An active PE executes the instructions issued by the IS, while an inactive

PE receives but does not execute the instructions. Assuming that the word length is a constant, the AP supports several important constant time operations, namely broadcasting, global OR/AND, maximum/minimum, and associative search. A search instruction, executed once, identifies all PEs whose data values match the search pattern (called responders) or do not match (called non-responders) [11]. Similarly a MAX/MIN search locates the global maximum/minimum in constant time. Search results can be logically joined in a single step. A detailed description of the properties that we assume for the AP can be found in [4, 16], ([Also see [25]). Two AP architectures that have been developed and used in a real-time environment are STARAN and ASPRO [2,3]. The CM-2 Connection Machine was a SIMD computer with 64k processors. The AP is easily capable of a similar number of PEs today.

3. Real-time scheduling – static vs. dynamic

Real-time scheduling differs from classic scheduling in that tasks must meet specified timing constraints. A real-time system executes tasks and must ensure not only their logical correctness but also their timeliness. If the timing deadlines are not met, the system fails, no matter how accurately the tasks are executed. Similarly if timing constraints are met at the expense of accuracy, the database may become inconsistent.

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 [19]. A *job* is defined as an instance of a task. A transaction may be considered a job. Next we introduce a new term called jobset. An AP is a set processor, i.e. it can process a set of data with one instruction. As a result, the AP can execute a set of jobs involving the same basic operations on different data simultaneously. This will be called a *jobset*. 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 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. It has been shown that there exists an optimal *earliest deadline first* (EDF) scheduling algorithm for a uniprocessor system [19]. As real-time systems become larger and tasks become more sophisticated, real-time processing has become much more dependent on parallel systems. Unfortunately, with the multiprocessor system, optimal scheduling algorithms have not been found for most problems. Stankovic et.al. [19], when discussing distributed scheduling, state that "The problem of determining an optimal schedule even in a multi-processor system is known to be NP-hard." Heuristic scheduling algorithms for multiprocessors assume restricted conditions and work only under special circumstances (see examples in [18]). A dynamic scheduling algorithm is inherently unpredictable. A static scheduling algorithm is preferable due to its simplicity and predictability.

4. Real-time processing examples

The motivation for this section is to compare the performance of Aps and MPs on some real-time database problems. A real-time static schedule for the air traffic control (ATC) problem has been given using an AP. A polynomial time algorithm is described for ATC in [25] for the AP. In contrast, MP solutions to the ATC problem use dynamic scheduling. The ATC scheduling problem and many other similar scheduling problems on a MP have been shown 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. Next, four examples will be given which show some of the advantages of the AP over the MP for the ATC problem in particular for real-time database systems with hard deadlines in general. The examples shown demonstrate the speed of the jobset approach to real-time automation.

Example 1 -- ATC conflict detection

For the first example, we look at the problem of detecting conflicts between aircraft paths in a dense environment. Earlier we stated that the AP is much more efficient than the MP because of both the simultaneous and constant time operations. We consider an ATC environment that has 12,000 aircraft tracks of which 4,000 are controlled IFR (instrument flight rules) flights. The other 8,000 are adjacent center IFR and uncontrolled VFR (visual flight rules), we designate these 8.000 VFR. We want to determine the possibility of a future conflict between any pairs of aircraft within a twenty-minute period. To assure timely evaluation we let the detection cycle be 8 seconds. A near approach within three miles or 2,000 feet in altitude will be called a conflict.

The process is essentially a recursive join on the track table where the best estimate of each flights future position is projected as an envelope into future time. The envelope

has some uncertainty in its future position which is a function of the track state, and is modified (to provide a 3 mile minimum miss distance) by adding 1.5 miles to each x, y edge of the future position and 1000 feet in the altitude. Then an intersection of each of the future space envelopes is attempted with every other space envelope in the environment. This means that every eight seconds we must evaluate each of the 4,000 IFR flights against the other 11,999 flights in our environment. It is readily seen that this process is $O(n^2)$ and would require many operations in a MP. As a test of each flight is completed, it is removed from the list. The total number of operations is *IFR*(IFR-1)/2 + IFR*VFR* or 4,000*(3,999)/2 + 4,000*8,000 which yields $39.998*10^6$ operations. It will be noted this number does not include any of the data management functions such as scheduling jobs to be executed, selecting processors to execute the jobs, distributing data to the selected processors, assuring data serializability, maintaining data integrity, concurrency management managing locking of data, or any of the other functions that are often associated with NP-hard problems.

In the AP this same process is O(n) and requires only 4,000 operations. The reason for this is as follows. First, all future flight envelopes are generated simultaneously – an O(1) job. Then in the AP, the first "trial envelope", of the 4,000 controlled future flight envelopes, is compared for possible conflict with all the other 11,999 flight envelopes for a look-ahead period of 20 minutes. Thus the equivalent of 11,999 jobs are completed simultaneously in this jobset. This occurs because each of the 12,000 records in the database is simultaneously available to each of the 12,000 PEs that are active in the AP. The next operation selects the second trial envelope and repeats the conflict tests against the remaining 11,998 tracks. As in the MP when a trial envelope has been tested it is marked "done" in the AP, and future trial envelopes will exclude all prior trial tracks. When the last of the 4,000 trial envelopes is tested the AP will have completed 39.998×10^6 jobs just as the MP did. But due to the simultaneous execution of all jobs in each jobset, the AP will complete the entire set of jobs in 4,000 steps. Thus it is O(n).

Example 2 – Radar/track data correlation

A second real world example involves the correlation of radar reports in relation R with the predicted position of established tracks for aircraft under observation in relation T. A track is the best estimate of position and velocity of each aircraft under observation. (This problem is present in many database systems and a major limitation in ATC performance.) A correlated radar report is used to smooth the position and velocity of the track to obtain the next estimate of position and velocity. Height information may be used to produce a better correlation, but we do not include that operation in the following. Given an unordered set of tracks, each report must be evaluated with every track in the system to assure a match (correlation) is not missed. We must treat multiple matches different than unique matches, and any reports that do not match a track are entered as new tentative tracks.

The correlation process proceeds by developing a box around each track to accommodate track uncertainties. A box is developed around each report to accommodate report uncertainties. The two database relations T and R contain information about points in an (x,y) space. Our objective is to determine the join of the two relations as the intersection of two boxes. Each box from R is evaluated for intersection with every box in T until all boxes from R have been compared with all boxes in T. T has columns x, y, x1, y1, j and k. R has columns x, y, r and q. We develop the box around each point (x, y) with the four corner points (x±j, y±j) in T. A similar box is developed about each point (x, y) in R with the four corner points (x±r, y±r). j is based on the uncertainties of each track in T and r in R is based on uncertainties in the radar report in R.

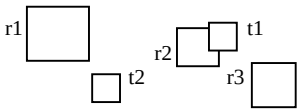

Figure 1 Track/Report intersect

The process proceeds by comparing each report in R with each track in T, as in Fig. 1. If an intersect is found between one report and one track, e.g r2 and t1 in Fig. 1, then the report data is entered into x1 and y1 of the correlated record in T and a correlation flag is set in column k (this record will be excluded from further testing with other reports in this period.). If two or more matches are found, an ambiguity flag is set into the correlated k values to indicate matches are ambiguous, and associated tracks are not updated in this period. If no match is found, a not match flag is set into column q of the correlated record in R.

When all report boxes have been compared with all track boxes, the value of j is doubled (the track box is enlarged) for all tracks that do not have a correlation flag or an ambiguity flag in column k of T, and the process is repeated for all reports in R that have the "not match flag" of column q set. Here we consider the uncertainties that may be caused by track acceleration, turns or by greater noise in the report. Where no intersections are found the process is repeated with $j = 3*j$. New tentative tracks are started to detect arrival of new flights for all reports that remain unmatched. If reports are due to noise, they usually will be dropped in several seconds. (The E2C could manage 2,000 tracks based on primary radar data. This is more than can be done in any MP ATC system today.)

In the AP solution, each report is tested with every track in one operation. That is, the AP time is O(n) where

n is the number of reports in this period. In the MP on the other hand it is seen that the MP process is $O(n^2)$ because it has $r \times t$ processes. In a worst case situation we anticipate 12,000 reports per second against 12,000 tracks. For the AP this is 12,000 operations whereas in the MP it is 144×10^6.

Example 3 – Flight plan conformance

The third example, flight plan conformance, compares each IFR track position with the current position developed from the filed flight plan. In this process each track position is rotated into the flight plan heading and compared with the allowable limits of deviation from the flight plan as shown in Figure 2. This is an O(n) process in the MP, but is O(1) in an AP as shown in [25]. Execution for just one IFR flight requires the same time as for 4,000 IFR flights in the AP.

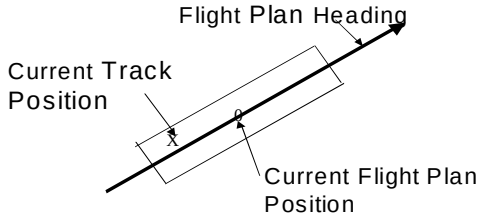

Figure 2 Flight Plan Conformance

We note that each of the preceding examples can be solved in polynomial time in both the MP and AP. However, it should be obvious that the AP solution time will be much faster because of the reduction of one order of magnitude in the number of operations executed. Other important factors must be considered when examining the MP solution. It seems that the performance of today's MP solutions to real-time problems with deadlines are limited by the inclusion of solutions to many hard problems, some of which are NP-hard. These include problems such as scheduling jobs to be executed, selecting processors to execute the jobs, distributing data to the selected processors, assuring data serializability, maintaining data integrity, concurrency management, managing locking of data, load balancing, etc. Some of these problems are discussed further in Section 5.

None of these problems play a role in the AP solution. Scheduling is done statically; i.e. there is no scheduling algorithm at run time. Processors are selected by their data content. Except for data input/output and broadcast to all processors, data is not moved. For example, in ATC, the data for each plane is stored in the processor for that plane. Data serializability, data integrity and concurrency management are achieved by virtue of the single instruction stream. The single instruction stream also eliminates the need for data locking (alternatively, all data is locked at all times). No load balancing is needed.

Example 4 – Multiple task execution

In this example, we consider handling the execution of k tasks. We assume the first three tasks are those given in Example 1-3 and the remaining tasks are similar ATC tasks. Further, we assume that there is a different processor for each plane and that the data for each of the n planes is stored in its processor. Due to the obvious data-intensive nature of these computations, an AP can be significantly more efficient than a MP where a common multi-user real-time database is required. Each record of a relation resides in the memory of one PE. Since the same operation is often performed on all the involved records, we can process all the records simultaneously. When using an AP for this example, all of the *n* jobs in a jobset are processed simultaneously where the number of PEs is n. Therefore, each jobset cost c_i (for task T_i) can be calculated at the instruction level. The following condition establishes overall system feasibility, where D is the deadline:

$$\sum_{i=1}^{k} c_i \leq D \qquad (1)$$

If Equation (1) is satisfied, the jobs can be processed in order of precedence constraints and all deadlines will be met.

Now, we consider solving this problem with a MP. 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 [SS8] from Gary and Johnson [7, page 238]. 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 $m \geq 2$, assuming not all tasks have the same length. (Here, *m*-processor means a MP with m processors). This theorem applies directly to our example.

The examples presented in this section are made more difficult, for a MP 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 common database), the scheduling problem for this set of tasks with the added requirement that they be performed on a MP is NP-hard as observed in [7,19,23]. On the other hand, because of the simplicity of the AP, a static schedule can be developed. This schedule will be incorporated into the table-driven scheduler for this problem.

With regard to the solution of the problem in this example (not just the scheduling of tasks), a polynomial time AP algorithm exists for this problem. In fact, details about a polynomial solution to a more challenging ATC problem are given in [26]. While the MP could conceivably also solve the problem in Example 4 by simulating the AP solution, this AP solution is not the one normally chosen by professionals working in this area. The

reason for this is the inability of the MP to efficiently simulate an AP algorithm, as explained in Section 1. An inefficient simulation of the AP algorithm is likely to result in the MP being unable to meet the required deadlines.

As a result of the preceding discussion, it is apparent that the AP can solve the problem in Example 4 in polynomial time, but that an MP cannot normally solve this problem in polynomial time, especially with real-time problems since their deadlines are very demanding. The preceding comments also apply to the ATC problem. In view of all the attention this problem has received, and expenditure of hundreds of man-years, it is highly unlikely to have a polynomial time MP solution.

5. Real-time transaction management

Another example of the problems in a multi-user database considers the serializable execution of transactions on a common database. It is generally recognized that many real-time systems use a common database on which consistent operations must be performed. Like any database, a real-time database has schemas, queries, transactions, commit protocols, concurrency controls, etc. Most real-time databases are multi-user databases, and as such the maintenance of adequate control over transaction execution becomes extremely critical. We address each of the following issues individually.

Transactions (real-time jobs) must maintain certain *ACID properties*: atomicity, consistency, isolation and durability. *Atomicity* assures that the set of updates of any transaction complete or fail as an isolated unit during execution. *Consistency* assures that data and operations used by a transaction do not violate any integrity constraints established for the database. *Isolation* assures that the execution of any transaction is done completely independently of other transactions that may be concurrently executing, and assures that the result of any transaction is independent of any others. *Durability* assures that data for each transaction that reaches the commit stage is never lost to other activity in the system and that any necessary rollback from an abort is similarly never lost [23]. It has been argued, in a great paper for those interested in real-time databases [8] that, in the interest of timeliness, ACID properties should be relaxed. In the AP solution to the real-time database problem, this relaxation of criteria is unnecessary because all the ACID properties can be supported by the AP.

Serializability is the process of controlling concurrent transactions so that they are executed as if they were done one after another, in order to avoid anomalies. When transactions are executed concurrently, a number of anomalies such as lost update, dirty read, inconsistent read, and ghost update may occur if the process is not controlled to assure these anomalies cannot exist. For example, the lost update anomaly can occur when transaction T1 reads a data value X followed by the read of the same value X by transaction T2. If T1 then writes a new value for X followed by the write of a new value for X by T2, the update by T1 is lost. If T1 reads X then updates X before T2 reads X, the operation would occur in serial fashion and the "lost update" would not occur. To assure serializability, a scheduler is maintained to determine if the order of data reads and writes are view equivalent (reads and writes appear in the same order in the transaction schedules) for a schedule of requested transactions. Concurrent schedules can be evaluated for view equivalence by an algorithm that has polynomial time complexity. However, to determine whether a schedule is view serializable (can be serially executed) requires that it be evaluated to determine if it is view-equivalent to any serial schedule. This is an NP-complete problem [23] and occurs because of the necessity of managing the concurrent transactions that must be conducted in a MP.

The *data locking* method of assuring that serializability is maintained works by "locking" the data used by a transaction. Initially each transaction acquires a lock on the data items it needs. This assures serializability because it prevents any other transaction from accessing that data during the ongoing transaction. Sometimes the lock is on the entire database, but it can also be on a table, record, or an item from a record. While locking assures serializability, it can severely reduce timely execution of transactions.

In the AP only one transaction can be executing at any time. Thus, the NP-complete scheduling problem, one of the most difficult database management problems in a MP architecture, is non-existent in the AP. The entire AP database is effectively locked for each transaction.

6. Database memory access comparisons

The importance of the memory bandwidth to computer performance is well-known. Fuller [21] observed that if a single simple parameter of performance had to be selected, that memory bandwidth would be a good choice. In the conclusion of his book, Pfister [14] states that for a wide and increasingly broad class of applications, the best measure for computer performance is the memory (bandwidth). The von Neumann "bottleneck" is another example of bandwidth limitation. Amdahl's law [9] is based on the fact that the running time of a parallel program is bounded by the running time of the sequential parts of that program. We argue that in real-time database problems, the serial part, accessing the real-time database, may often substantially exceed the parallel part when using MPs. This occurs because of the requirement to maintain

the ACID properties (section 5) in the common database of any real-time multi-user system.

For an example, we temporarily assume a rigid constraint that each operand must be taken from or inserted into a common database memory one word at a time, before data can be distributed among the multiprocessors. In ATC, if we wish to update the X-axis of each track by adding ΔX, the distance increment for one second, for 12,000 tracks to show new best estimates of X. Then, with the MP, 36,000 memory accesses are needed to get the data and return the result to the common database. If we consider the memory accesses for data fetch, data distribution, and instruction execution this number will grow substantially.

Next, we relax the rigid constraint made in the preceding paragraph and instead assume that a very limited number of data items in the common database can be accessed at one time. The resulting smaller number of required accesses for the MP will still compare very unfavorably with the number of required memory accesses for the AP that are obtained in the next paragraph.

In the AP, only 96 data memory accesses are needed for all 12,000 operations. This is true because the AP is a set processor, i.e. it can operate on an entire set of data with a single instruction. As mentioned above all operations on the set of 12,000 track operands are executed simultaneously. This process is O(1).

In the MP operations of searching are often dependent on sorting or indexing, which adds to the required memory accesses to establish the sort and increases processing time. On the other hand, in the AP, these operations are never needed. When sorting is examined carefully it is seen that the only reason for the sort is to achieve "content addressability". In the AP content addressability is an inherent hardware feature thus sorting or indexing is not normally needed.

7. APs vs. MPs for real-time problems

In [24,25], we have addressed in detail the advantages of APs and the difficulties of an efficient MP solution for a real-time database problem such as ATC. In this section, we summarize some of the ideas presented. An AP has three major features that make it possible to efficiently solve real-time database problems. First, an AP eliminates synchronization costs in MPs. Synchronization costs are high and can cause significant time delays in worst case environments. The significance of this has been underestimated, as most current studies on synchronization costs are based on an average case, which is not applicable in real-time problems. Second, since an AP locates data by content rather than by address. Sorting and indexing that are normally required in MPs are not needed. Thus, the complexity of the software on an AP is significantly reduced and the real-time deadline is much more easily met. Third, an AP has extra-wide memory bandwidth.

The traditional memory access bottleneck does not exist in the AP. A data item can be broadcast to one or more PEs in one step. This is in stark contrast to either shared-memory MIMD systems or distributed memory MIMD systems that use a much more complicated communication mechanism.

A real-time scheduling problem using a MP is usually NP-hard. Normally, the problem of scheduling a set of tasks is solvable on a uniprocessor in polynomial time. However, when one or more of the MP-type requirements, such as multitasking and shared resources, are involved, there is no known polynomial time algorithm to schedule a set of real-time tasks. This leaves the fact that most real-time problems are not solvable on MPs.

There are difficulties whenever a MP is used for scheduling problems. These include multitasking, shared resources, preemption, varied release times, precedence constraints, etc. All of these factors have to be dealt with in a MP environment when solving real-time problems. They are all involved in one or more NP-complete problems given in [7]. Therefore, for a real-time problem, e.g., air traffic control, with a set of tasks that need to be scheduled, it is not expected that one could find a polynomial time solution using a MP.

8. Conclusions

In this paper, we have shown that that the common belief that MPs are more powerful than APs is unjustified and ignores the many intrinsic weaknesses of MPs. In particular, we cite the ATC problem which can be solved in polynomial time on an AP, but cannot be solved in polynomial time on a MP. The problems with MPs were not fully recognized until their ability to solve real-time scheduling problems were considered. MP solutions of the ATC problem 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]. A USA Today editorial [10] sees the problem this way: “This time, the FAA has only itself to blame. Back in 1995, Congress freed the FAA from cumbersome procurement rules that the FAA claimed were the main cause for the unrelenting delays and cost overruns plaguing the 17-year, $41 billion modernization effort.” On the other hand, the ability of associative SIMDs to successfully handle air traffic control problems has been demonstrated [12,17].

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

satisfactory solutions to problems that are considered intractable for MP systems. Moreover, considering the AP's advantages of simple programming style and simple hardware implementations, it obviously deserves more attention and utilization if we want to solve today's real-time problems.

Acknowledgement

The authors wish to thank an anonymous referee of this paper for their valuable comments.

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] Stankovic, Son, Hansson, "Misconceptions About Real-Time Databases" Computer, June 1999

[9] G. Amdahl, Validity of the single-processor approach to achieving large-scale. *Proceedings of the AFIPS Conference* pages 483-485, 1967.

[10] April 19 USA Today editorial

[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 Requirements." *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] S. H. Fuller, *Introduction to Computer Architecture* (edited by Stone) Science Research Associates, 1980, pp 530

[22] *Proceedings of the 1973 Sagamore Conference on Parallel Processing*, (August, 1973). A general conference on SIMDs.

[23] Atzeni, Ceri, Paraboschi and Torlone, *Database Systems*, McGraw-Hill, 2000

[24] Jin, Baker, Meilander, The Power of SIMDs and MIMDs in Real-time scheduling, I*PDPS 2002, MPP workshop*, IEEE Digital Library, (Unofficial version: http://vlsi.mcs.kent.edu/ ~parallel/papers)

[25] Meilander, Jin, Baker, Tractable Real-Time Control Automation, *Proc. of the 14th IASTED Inte'l Conf. on Parallel and Distributed Systems (PDCS 2002)*, pp. 483-488 (Unofficial version:http://vlsi.mcs.kent.edu/~parallel/papers)

[26] Meilander, Baker, Jin, Predictable Real-Time Scheduling for Air Traffic Control, *Fifteenth International Conference on Systems Engineering*, August 2002, pp 533-539 (Unofficial version:http://vlsi.mcs.kent.edu/~parallel/papers)