Real-Time Scheduling in Command and Control

Will. C. Meilander
Kent State Univ.
willcm@mcs.kent.edu

Jerry L. Potter
Kent State Univ.
potter@mcs.kent.edu

Kathy J. Liszka
The Univ. of Akron
liszka@computer.org

Johnnie W. Baker
Kent State Univ.
jbaker@mcs.kent.edu

ABSTRACT

Real-time tasks for command and control systems are too large or too complex for one processor to handle. Simply adding more CPUs does not result in a linear increase in performance. Current comparative analysis of parallel algorithms does not accurately reflect the increased cost of scheduling when more processors are added. A case is made that associative processors effectively handle real-time command and control type problems and avoid most of the difficulties introduced by multiprocessors. These results suggest that when comparing different architectures, comparative analysis should consider the ALU and control unit {CU} separately.

Keywords

Real-time scheduling, command and control, parallel processing, associative processing, air traffic control.

1. INTRODUCTION.

Scheduling complexity analysis is a very general problem that is studied in many contexts. It may be utilized to provide assurance of a system's ability to complete a set of tasks within a time deadline for a worst case set of input conditions under a set of predefined constraints. The inability to complete a set of tasks in a given time frame forces us to alternative architectures in seeking a solution. In many real-time systems the task is too large or too complex for one processor to handle. When you run out of "real time", you go to multiple processors (MPs) resulting in systems that substantially increase the size and complexity of the task. (*We regard MPs as parallel architectures with multiple CPUs, e.g. MIMD, SMPs, NUMAs, Clusters, etc.*) Indeed, when the number of current processors is in the 16-32 range, adding even one more processor can add more complexity to the system than the additional computing power contributes [6]. Finding solutions to the general problem is not trivial for practical situations. Specifically, in command and control systems, most real-time, MP scheduling is considered to be a NP-hard problem [6]. Often those components of MP systems that add complexity have not been thoroughly addressed or studied in a practical sense. These factors must be looked at seriously to truly analyze the performance of real-time command and control systems.

We claim that the current state of comparative analysis does not accurately reflect the impact on scheduling of adding additional instruction streams (IS) when additional processors are added. We observe that single instruction stream, multiple data stream associative processors (APs) avoid most of the problems introduced by MPs. This paper will discuss the needs of real-time command and control systems in Section 2 using air traffic control, a real-time relational database management problem, as a practical example. Section 3 describes the associative processor architecture. We propose it as a practical solution to static scheduling in these types of systems which yields a current optimal schedule. We use air traffic control as a real world example to demonstrate the difference between MPs and APs as solutions for typical command and control problems in Section 4. We consider AP advantages/disadvantages in Section 5 and state our conclusions in the last section.

2. REAL-TIME COMMAND AND CONTROL REQUIREMENTS

In general applications for real-time systems need to be fast, predictable, reliable and adaptable [7]. Air traffic control systems, like DABS/IPC, AAS, and STARS have been unable to meet their requirements in FAA's $41 billion modernization program[10]. In fact, the DABS/IPC and AAS programs have been cancelled without completion. STARS is having problems and delays in attempts to put it into the field. We will address the four points that impact the success, or lack thereof, in these systems.

2.1 Fast

CPU and memory speeds have been rapidly increasing since the development of the first command and control systems. CPU speeds have increased from one megahertz to one gigahertz. Memory speeds have increased from two microsecond access time to the current ten nanosecond access time. Yet the tremendous increase in stated computer capability has not brought about an expected and needed increase in performance. Specifications are still not met. Developers are trying to explain why increased computer speed is not the sole solution to their problems. G. Pfister states the real problem succinctly in the penultimate paragraph of his book [9]: "Once a processor is fast enough to continuously saturate a memory system, what difference does the architecture make? For computer performance in a wide and increasingly broad class of applications, 'It's the memory stupid!' "

2.2 Predictable

The AAS program, initiated as it became obvious that DABS/IPC was failing, started with two nearly half billion dollar "proof" studies to establish predictability. The studies spent the money, but proof of predictability was never shown. Nevertheless, an implementation was started which continued, for about ten years, until June 1994 when the program was officially "terminated" We later show examples of

predictability using an AP and discuss why the AP performance can be reliably predicted. Predictability is needed for in all software. For example one is advised in [11] to "…develop high-confidence systems with predictable properties at a predictable cost."

2.3 Reliable

Command and control systems operate under severe reliability requirements. If the workload increases due to an unusual or unexpected load, the system may crash. Many air traffic controllers have seen their screens go blank because of computer overload in the ATC system. We read "The President's airplane is missing for one minute."[15] This is the result of computer overload. Primary radar data would have followed the President's plane but the data was necessarily abandoned to prevent computer overload. Missing data of this kind is expected and accepted by air traffic controllers daily.
During periods of overload, each system is configured to ignore aircraft in defined sectors, while using only selected information from other sectors. Further, the ATC system also ignores certain classes of aircraft flights to help alleviate system overload. Failures such as these can result in loss of life, which can be prevented if adequate computational capacity is available.

2.4 Adaptable

All programmable real-time systems are subject to change. The first installation rarely satisfies all requirements: glitches seem to arise out of nowhere, the environment may be slightly or substantially different than anticipated, new needed functions are added, the expected load is significantly greater than planned, etc. It is essential that the system can be adapted to react to new situations.

A primary goal of any modernization effort should be the ability to easily expand to handle new tasks as they are needed. This was demonstrated in the first AP installation in the Knoxville terminal in 1971 [4] (and it is even more important in 1999 [11]). The initial requirements were for automatic track initiation, conflict detection, and conflict resolution. That program was developed and demonstrated in about six months. A program for automatic voice advisory (AVA) of approaching traffic for VFR flights was added[8]. AVA was completed in about three months. Approximately 75 pilots participated in AVA. Another requirement, that of terrain avoidance, was added and completed in about four months. The total AP programming and support effort was about four man-years and yielded 2,200 instructions. It was clear that additional tasks could be added easily in the AP. Current ATC systems cannot do today what was done using an AP at Knoxville in 1971[1].

3. ASSOCIATIVE PROCESSORS

Associative processors access their data associatively, that is, they access objects in the local memory of each processing element (PE) by content rather than by address. The term *associative* means that an entire record is selected by matching one or more of its fields. Associative access is accomplished by broadcasting an item and having all active PEs search a specified field in their local memory. An associative processor, such as the ASPRO built by Lockheed-Martin, consists of an array of cells, each conceptually containing a PE and a local memory. Cell memory holds information such as track data (ex. ground speed, heading, altitude). A control processor stores the program and broadcasts program instructions to all active cells. In this tabular data structure with many related records, associative searches may find multiple records for a given search query. The idea is to perform the operations in constant time by performing pattern matches in parallel. For example, it is possible to select a group of tracks in response to a request from a specific air traffic controller with the execution of a single instruction string. Those track records that match the request successfully are called responders, while those track records that do not produce a match are called non-responders. The basic idea of accessing data associatively is not new [1,2,3], but advances in technology in the past two decades have made this a viable technique for developing new approaches to efficient data parallel programming [5]. In addition to database management the AP can support a wide range of applications including string matching, convex hull, 2D knapsack etc [19].

4. MPs VS APs FOR REAL-TIME COMMAND AND CONTROL

Single CPU systems suffer from the classic von Neumann bottleneck, that is, moving instructions and data across a limited bandwidth path from memory to CPU. It is widely recognized that the key to increased capacity is the delivery of more data per time unit from memory to CPU. The standard approach to high capacity real-time command and control applications has been to use MPs. Yet the MPs suffer from the same von Neumann malady, but on a grander scale. When applications are what is called embarrassingly parallel, MPs should provide an increase in performance. But usually data must be shared among processors, requiring some kind of communication network and an additional level of algorithmic complexity to coordinate this process. Clever use of a communication network and a good parallel algorithm may reduce some of the inefficiency introduced, but the von Neumann bottleneck still exists. We present a different solution that greatly increases memory bandwidth while reducing software and hardware complexity.

We use the air traffic control application as an example to show the significant difference in effectiveness of MPs and APs. The data for this application is a set of large relational databases such as operational flight plans, aircraft characteristics, weather and so forth. In the operational flight plan table, each row represents the plan for a single flight with fields such as (but not limited to) flight identity, next checkpoint, expected arrival at next checkpoint, aircraft type, heading, speed, and altitude [4]. Another table contains track data about the flight in progress with a common key field such as flight identity to link the two tables. Thus, the relational database structure can be used to provide a rigorous mathematical model for system definition and data processing in the ATC system.

Using an MP to solve the demands of the relational approach adds a great deal of complexity that is often ignored in theoretical analysis. For example, each of the four basic database functions, enter/read/modify/delete must be preceded by a search. In order for this to be feasible, the database must be sorted and indexes must be maintained on many fields of the database. In sharp contrast, an AP is a content addressable system, eliminating the need for any type of sorting/indexing

[1] An explanation for why this solution was not adopted can be found in appendices C and D of [12].

software. In an MP, task-scheduling software is required. Stankovic et al state in [6] "…complexity results show that most real-time multiprocessing scheduling is NP-hard." An AP is a set processor. It has a single instruction stream and is statically scheduled, eliminating the need for dynamic scheduling software. With an AP, data distribution is inherent in the data structure, while an MP requires data distribution software. Record or table locking software must be used in the MP environment to provide concurrency, consistency and data integrity. AP tables need never be locked since only one instruction executes at any time. Finally, for an MP to handle a relational model the data pointers which are used in profusion to link data parameters must be continuously maintained and updated. Data pointers are unnecessary in APs because, unlike other architectures, the conceptual and physical representations of data stored in AP can be identical.

Many experts agree that the key to faster data processing is to deliver more bits of data to the CPU in the same amount of time. In an AP, this is easily accomplished by simply adding more cells, i.e., PEs plus memory. In an MP, adding more CPUs is the conventional solution. However, in MPs both data and instructions share the memory-CPU path. As more instruction streams are added when processors are added, eventually more software must be added to control them, significantly increasing scheduling and communications and substantially increasing the complexity of the underlying algorithm. Yet, in many situations, a linear improvement in execution time is expected for MP environments. That is, if a task takes ten minutes to execute on a single CPU, it is expected to take only one minute if 10 CPUs are used.

This type of logic is possibly promoted by examples often included in introductory computer science books of embarrassingly parallel problems such as the Manhattan telephone book problem[16]. This example asserts that the time it takes one computer to locate a name in this directory can be reduced by 100 if 100 computers are used. For MPs, this example ignores the costs of synchronization, query broadcasts, resynchronization, reduction of results, etc. Further, observe that if the database is dynamic then many of the data management overhead costs mentioned earlier are also incurred.

Operation	MP	AP
Report to track correlation	$O(n^2)$	O(n)
Conflict detection	$O(n^2)$	O(n)
Conflict resolution	$O(n^2)$	O (n)
Terrain avoidance	$O(n^2)$	O (n)
VFR automatic voice advisory	$O(n^2)$	O (n)
Cockpit situation display	$O(n^2)$	O (n)
Track smooth and predict	O(n)	O (1)
Coordinate transform	O(n)	O (1)
Flight plan update	O(n)	O (1)
Flight plan to track conformance	O(n)	O (1)
Display data selection	O(n)	O (1)

Table 1 – Comparison of some required command and control functions, exclusive of data management overhead software.

The usual definition of speedup is speedup = Ts/Tp where Ts is the time for a sequential algorithm and Tp is the time for a parallel equivalent. A fallacy that is encouraged by the preceding example is that speedup = (number of processors)/k where k is a constant ≥ 1. This is equivalent to the statement that the speedup is linear with the number of processors or that Tp=k*Ts/(number of processors). Unfortunately many computer scientists seem to believe some version of the above fallacy holds for MPs..

A difficulty in devising a metric for comparing the performance of APs and MPs is the current practice of treating CPUs and PEs as equivalent. However each CPU in an MP has both a control unit (CU) that supports an instruction stream (IS) and an ALU that supports a data stream (DS), while the AP has one CU and a multiplicity of ALUs called PEs. In terms of hardware gates, each CPU is roughly equivalent to 800 PEs in an AP. We feel it is essential to consider both the number of ALUs or DSs and the number of ISs in metrics designed to compare the performance of APs and MPs.

The increased complexity of the MP[2] algorithm may basically be described as multiprocessor data *management overhead* software and includes:

- Dynamic real-time task scheduling
- Task assignment to individual processors
- Data assignment and distribution
- Data moves between common memory and processors
- Data moves between processor memories
- Data sorting and indexing
- Re-indexing and re-sorting when changes are made to the data set
- Bus arbitration software or hardware
- Multi-tasking and multi-thread software
- Data pointers
- Pointer update processing
- Table or record locking
- Cache coherency management
- Memory coherency management
- Sequential consistency management
- Iterative data processing loops at individual processors
- Iterative access to individual data items for processing

Table 1 shows our comparative analysis between a sequential solution and an AP solution to processes that are common in command and control**. The order of the tasks shown in Table 1 does not include MP costs due to data management overhead**. In an AP solution to parallel problems, more memory-to-CPU bandwidth can be supplied simply by adding more cells. Since additional instruction streams are not added, the software so essential to a multiprocessor system is unnecessary in an associative processor. In an AP with *n* processors, most tasks which require O(*n*) time in a sequential processor can be executed in O(1) and typically tasks with $O(n^2)$ operations in a sequential processor can be executed in O(*n*) in an AP. Even more significant are the additional data management overhead tasks that must be performed to effectively use MPs. The costs of these tasks are often ignored when evaluating the efficiency and performance of MPs for

[2] We distinguish data management overhead software from data processing software. Data management overhead is the organizing and moving of data without changing the data. Data processing, on the other hand, operates to effect programmed modifications to the data.

real-time systems. Table 2 lists our analysis of these tasks and the number of operations required for n tasks using p processors. Note that "0" for the AP does not indicate constant time but means that these operations simply are not required when an associative processor is used

Operation	MP	AP
Processor assignment	O(p)	0
Program distribution	O(p)	0
Data redistribution	O(n)	0
Data load & Sort	O(n log(n))	0
Virtual memory	O(n)	0
Memory/cache coherency	?	0
Serializability	?	0
Locality of reference	?	0
Bus arbitration	?	0
Multi-tasking multi-thread software	?	0
Data pointer management	?	0
Dynamic scheduling	?	0
Table or record locking	?	0

Table 2 Comparison of operations required for software management tasks.

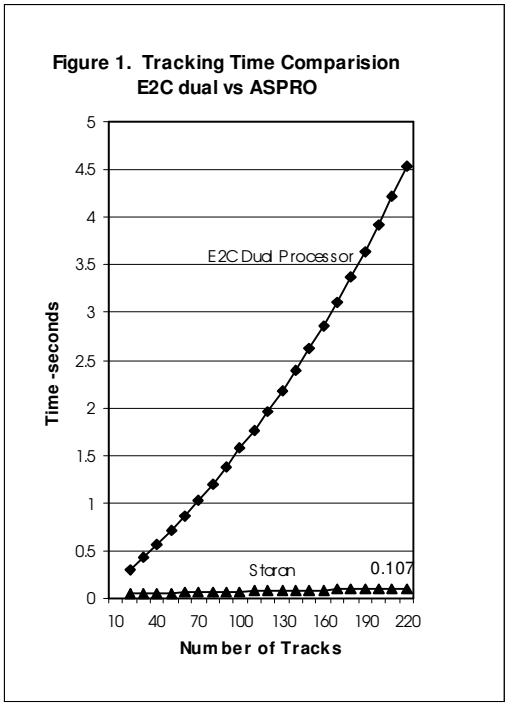
We cite a real example with a comparison of radar tracking times using the L304 dual processors and an ASPRO associative processor. Figure 1 shows the tracking time comparison for a small track environment. Information for Fig. 1 from program reviews at Grumman Aerospace in November 1979.

2000 Tracks - 4000 Reports

Routine	Instruction count	Time in milliseconds Predicted	Measured
Association Pair	415	*	640
Aq Comp and sort	1012	*	14
Correlation	788	22.16	4.5
Tent.. Track	555	16.68	12.5
Track update	661	14.84	8.9
Hrtup	407	2.68	2.9
Range pred.	640	37.04	24.77
Association Gates	443	9.12	8
Kalman Tracking	1026	46.64	39.2
Track Quality	209	7.28	5.06
Air/Surface	326	*	0.66
Estab Track	407	0.88	0.71
Final bookkeeping	243	15.98	6.6
Total	7132		767.8

Table 3 – Tracking predictability of an AP
* Not predicted

Table 3 gives the instruction count and both the predicted and measured time in milliseconds for the ASPRO with a large (4k/2k) report/track environment. Information for Fig. 1 from program reviews at Grumman Aerospace in January 1980. The deadline time for this process was 5 seconds. Ignoring the established deadline time, the L304 dual processor required 212 seconds to complete this task. As Table 3 shows, the ASPRO



time was 0.768 seconds, or a 276 times throughput improvement. It should be noted that the L304 memory speed was 900 nanoseconds whereas the memory speed in ASPRO was 450. nanoseconds, Thus, the comparable improvement factor was 138. The L304 software was the culmination of many years of effort and, to optimize performance, was written in machine assembly code. On the other hand the ASPRO software, also assembly code, was developed in about six months by the same people who developed the dual processor code. They commented favorably on the simplicity of AP software.

To demonstrate the increase of task time with increasing aircraft load, we consider a universal task in command and control: the operation of correlation of incoming sensor reports with established tracks. When a set of reports has been received from the system sensors they are converted from sensor coordinates to system coordinates. Assume a set of track records has been previously developed and the reports must be correlated with the predicted position of each established track. This correlation process proceeds by comparing each report position with each track position to find the best fit between each report and every track. When the best fit has been found the report is assigned to the "found" track. This proceeds until each report has "found" its track. The process is an order n^2 task where n is the number of reports and tracks. For this example we ignore multiple "finds" and false reports.

Process time can be reduced, using a 2-d sort for sorting the tracks and reports into "sort boxes". In the National Airspace Enroute System [17], the boxes are 16 miles on a side. Then, with the addition of two sort operations, it is only necessary to compare each report with the contents of nine adjacent track boxes. FAA found an excellent improvement of this process by offsetting the report sort boxes by eight miles. Then correlation could be made by comparing the contents of a report box with the contents of only four adjacent track boxes, instead of nine.

If we assume a track density *d* of one flight per sort box then four comparisons are required. Observe that if the track density *d* is increased to two flights per box then the number of comparisons has increased to sixteen. Next observe that, in addition to the two $n*\log(n)$ sort operations, the number of comparisons is equal to $(2*d)^2$. Thus the task is increasing at a greater rate that the square of the target density. By contrast the identical process in an AP is linear with *n* since each report is compared with *n* tracks simultaneously.

To further evaluate the comparative performance of APs and MPs we consider a typical ATC task [4] of updating a flight plan (FP) file and testing the FP for conformance with its sensor-developed track, 4000 flights are chosen as a problem. This was the AAS requirement. Memory access count is chosen as the metric of performance. In Table 4, Xf, Yf, and Hf give the flight plan (FP) position; xd, yd, and hd give the FP distance increments per unit time; Xt, Yt, and Ht give the track parameters corresponding to each flight; and hdg is the FP heading. These results may seem surprising, but they are real. Note that instructions for sorting, data distribution, task assignment, etc. for the MP are not shown but increase the count significantly. Values shown are a lower bound for the MP [4].

Op#	Function	Memory Accesses ⇒ MP	AP
1	Get next/all FP pointer	4,000	1
2	Get Xf, Yf, Hf, xd, yd, hd	24,000	144
3	Add operands and store Xf, Yf, Hf	12,000	144
4	Get associated track pointer	4,000	0
5	Get Xt, Yt, Ht, sin(hdg), cos(hdg)	20,000	120
6	Calculate displacement of track from FP X'=(Xt-Xf)*cos(hdg)+(Yt-Yf)*sin(hdg) Y'=(Yt-Yf)*cos(hdg)-(Xt-Xf)*sin(hdg)	0	0
7	Check X'>Klat, if true set alert flag	?	1
8	Check Y'>Klong if true, update FP	?	1
	Position to Xt, Yt.	6	96
	Total	>64,006	506

Table 4-Memory access count for flight plan conformance

5.AP ADVANTAGES - DISADVANTAGES

Associative processing offers many advantages in the world of real-time command and control. Among these are the substantial reductions in hardware and software. The amount of hardware to carry out the requirement is likely to be only one fifth that needed for an equivalent MP solution, if one can be found. The reduction in hardware is due to: 1) the elimination of most of the control units required in the MP, 2) the reduction of memory needed for the multitudinous pointers, index and sort tables that are eliminated in the AP, and 3) the near elimination of the costly data management software so essential to the MP hardware software configuration. The AP offers a substantial software reduction, to about 20% of the software needed in the MP. This is achieved by eliminating internal loops so ubiquitous in the MP and through elimination of the many time consuming data management overhead operations which must be included to fairly measure the MP performance. A major disadvantage of APs is the general misunderstanding by most computer experts of the types of problems that can be easily solved by such architectures. Surprisingly, AP solutions for many problems are much simpler than sequential solutions to the same problems. In discussing parallel processing Patton writes, "While the world around us works in parallel, our perception of it has been filtered through 300 years of sequential mathematics, 50 years of the theory of algorithms and 28 years of Fortran programming."[20] A real disadvantage of APs is that the APs advantages of hardware simplicity and ease of programming are widely misunderstood.

Another supposed disadvantage of the AP that is often mentioned is that when multiple branches occur in a program, the AP must execute each of these branches sequentially. However, one should remember that an AP can also execute each branch over a set in parallel. Moreover, it should be recognized that many of the branches required to reduce processing time in an MP are nonexistent in the parallel AP program. In contrast, when an MP executes a job control parallel program, each of the branches are executed sequentially by one CPU in an MP. When a data parallel program is executed by an MP with each of the CPUs containing multiple records, then typically most if not all of the CPUs execute some and perhaps all of the branches sequentially.

We have shown that many of the supposed disadvantages are minor or nonexistent. The simplicity of the hardware and software will more than compensate for any AP disadvantages. Some people cite life cycle cost as a measure of goodness for a system. Because of the simplicity of the AP software and hardware, the life cycle cost of the AP is very low. We believe that an AP could have handled the data processing for the canceled AAS system. How does one compare life cycle cost of a system that cannot meet the system requirements with a system that can?

6. CONCLUSION

The associative processor provides a much simpler hardware solution with the very important benefit of much simpler software. This simplicity is achieved because only one instruction stream exists so that an entire set of n data items (given $p \geq n$, p is the number of processors) may be processed with a single instruction. When $p < n$ then each PE handles n/p

data items and the running time is increased by a factor of n/p also. Many people believe an AP must have one PE per data item This is untrue and would seriously limit the capability of the AP. Thus, AP software is even simpler than sequential processors since 1) the inner most loops are not required, 2) many tasks such as sorting and indexing are not required, 3) dynamic memory allocation and release (i.e. garbage collection) are constant time (they only take two parallel instructions), and 4) the I/O data paths in an AP are just as wide as a memory to CPU data path. The FAA application shown in Table 4 bears this out. This is contrary to popular opinion, which holds that APs are more difficult to program and are not as efficient as MPs.

Conventional comparative analysis indicates that APs are less cost efficient than MPs. A problem in comparing the cost of AP algorithms is that the cost of each is defined theoretically to be the product of the number of algorithm steps and the number of processors. The AP is penalized by this analysis since it normally has a much larger number of PEs than the number of CPUs in an MP. However, this cost ignores the fact that each processor in an MP includes an instruction stream while the AP we consider here has only one instruction stream. The interaction between the instruction streams in an MP is the cause of the costly data management overhead software and this cost is ignored in this comparison between AP and MP algorithms. As indicated earlier, this data management overhead software running time may be far greater than the running time of either the AP or MP algorithm. A fair comparison between the cost of an AP and a MP algorithm should require the cost assigned to the MP algorithm to also include the cost of executing the data management overhead software on the MP during the execution of the MP algorithm. As indicated earlier, the corresponding overhead cost for executing the AP algorithm is insignificant.

In order to more correctly assess the potential of the AP, a more accurate method of comparing the cost of executing software on the AP and MP is required. Such a comparison would also consider the huge cost of the data management overhead software on the MP. Also, the CPU of an MP is roughly equivalent to one IS and one PE in an AP, but current complexity evaluation techniques consider a CPU of an MP to be equivalent to a PE of an AP. This costing model is unfair to the AP since there are normally many more PEs than CPUs and each PE is only a simple ALU while the CPU has the ability to operate independently and is far more complex and powerful than the PE. Additionally, the interaction between the ISs in these CPUs is the cause of the costly data management overhead software. Increasing the number of CPU's in an MP causes this overhead cost to increase dramatically. The corresponding hidden overhead cost due to increasing the number of PEs in an AP of comparable power is minor. One approach to comparing current AP and MP architectures would be to compare their running time on a suite of basic software packages of the type often required in real-time command and control problems, such as a real-time tracking algorithm. Since the two architectures are quite different and memory speed is the limiting factor in computer performance, perhaps counting the number of memory accesses that occur (including those in the data management overhead software) would be a useful way to compare their costs. However, as demonstrated by the difficulty encountered in establishing air traffic control systems, developing the software for the MP for this test is likely to be a very demanding task. An alternate approach of defining a "toy problem" that contains some of the critical ingredients of a real-time command and control problem was initiated in [12].

We believe that, when correctly compared to other alternatives, the AP architecture holds much promise for a wide range of important real-time applications. These include the ATC problem specifically and, more generally, real-time command and control problems. These problems are representative of dynamic database applications, which are rapidly expanding. An AP provides an efficient approach to processing these types of problems. However, in order to more correctly address the potential of the AP, a more accurate method for comparing the cost of executing software on the AP and the MP is required.

7. REFERENCES

[1] K. E. Batcher, STARAN parallel processor system hardware, “Procs. National Computer Conf.,” pp. 405-410, AFIPS, 1974.

[2] C. Foster, “Content Addressable Parallel Processors,” Van Nostrand-Reinhold, New York, 1976.

[3] E. Jacks, “Associative Information Techniques,” Elsevier, New York, 1971.

[4] W. C. Meilander, J. W. Baker, ATC Computers – Yesterday, Today, Tomorrow, The 43rd Annual Air Traffic Control Association Fall conference Procs, 1998, pp. 91-95,.

[5] J. L. Potter, “Associative Computing – A Programming Paradigm for Massively Parallel Computers,” Plenum Publishing, New York, 1992.

[6] J. A. Stankovic, M. Spuri, M. Di Natale, G. C. Buttazzo, Implications of Classical Scheduling Results for Read-Time Systems, Computer June, 1995, pp. 16-25.

[7] J. A. Stankovic, “Read-Time and Embedded Systems,” The Computer Science and Engineering Handbook, Ed. Allen B. Tucker, Jr., CRC Press, 1997, pp. 1709-1724.

[8] Richard Collins, “On Top”, Flying Magazine, December 1995.

[9] Gregory F. Pfister in “In Search of Clusters - The Coming Battle in Lowly Parallel Computing”, Prentice Hall, 1998, pg 516.

[10] Editorial April 19, 1999, USA Today.

[11] Krishna Kavi et al "The Pressure Is On", Computer, Jan 1999, pp 30-33.

[12] Lu, Qian. "Complexity Analysis of ATC." Thesis for Master's degree, Kent State University, Dec. 1997 Sponsored by W. C. Meilander and Johnnie Baker

[15] Assoc. Press March 11, 1998

[16] Schneider and Gersting, "An Invitation to Computer Science", p 228, PWS Publishing, Pacific Grove, 1998.

[17] FAA, NAS-MD-321, 7 July 1987, pg b-5

[18] //www.clbooks.com/ May 1995

[19] J. L. Potter and W. C. Meilander, " Array Processor Supercomputers," IEEE Spectrum, vol.. 77, no. 12, pp. 1897-1914, Dec. 1989.

[20] Peter C. Patton, "Microprocessors, Architectures and Applications," IEEE Computer Mag., Vol 18, no. 6, pp 19-29, June,1985