Tractable Real-Time Air Traffic Control Automation

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

Abstract

A different paradigm is needed for real-time command and control (C&C) problems. Past approaches, using multiprocessors (MP), for real-time computing have had great difficulty in meeting real problem requirements. We review some reasons why C&C problems that require a solution on a MP architecture may be intractable, and then show an architecture where these reasons for intractability are non-existent. We describe a polynomial time solution to the air traffic control (ATC) problem, which is a typical C&C problem. This solution uses a static, non-preemptive table driven schedule using a SIMD architecture called an associative processor (AP). The AP is an ideal processor for set and database operations since its single thread instruction stream can operate on an entire set of data with each instruction. The AP eliminates multi-thread instructions, which account for much of the MP intractability mentioned above.

Keywords: Real-time processing, air traffic control, associative processors, multiprocessors

1. Introduction

It has been widely established that almost all real-time task scheduling problems for a multiprocessor that require the inclusion of any of several common MP activities in the solution are NP-hard. Scheduling for the Air Traffic Control (ATC) system is typical of real-time scheduling, and all solutions to the ATC problem that assume a MP is used in any of several conventional ways will have exponential running time. Further information can be found in [1, 2, 3].

The ATC system consists of a number of sensors that report an aircraft position. This data is joined with a track table which develops the best estimate of current position and velocity of each flight in the system. Flight plan information, the intent of each controlled flight system is maintained, as is weather information, aircraft data, terminal conditions, etc. The above information is considered to exist in a client-server database system. Data is supplied to a number of clients including the ATC controllers, who are responsible for safety of scheduled flights, airlines, general aviation pilots, and data is retained for information and future evaluation.

In a recent special broadcast on ABC news about current ATC system difficulties [4], they stated that the cost of the development of the current solution to the ATC problem through 2005 has been projected to be 44 billion dollars. All attempts at providing a multiprocessor solution to the ATC problem dating back to 1963 have failed to meet their performance requirements (CCC which started in 1963, DABS/IPC in 1974, AAS in 1983, STARS in 1994.). The STARS project is now over budget and nearly four years behind schedule.

We offer a distinctly different computational approach, *a new dimension in memory access*, for solution of these ATC problems that are generally considered NP-hard. We suggest using a simple single instruction stream multiple data stream (SIMD) associative processor. The AP is very similar to the conventional von Neumann processor. It has a similar instruction-processing unit, but has thousands of arithmetic logical units (ALU) instead of the single von Neumann ALU. Figure 1 provides an overview of the AP architecture that is assumed in this paper. The AP is seen to provide a new dimension of memory access. Instead of accessing 32 bits in one memory cycle, an AP with 16,000 ALUs, can access 16,000 bits at one time.

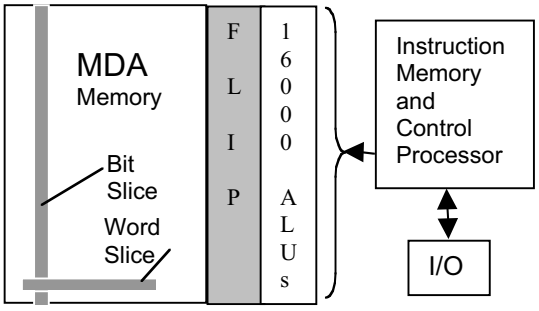

Figure 1 ATC Associative Processor

We point out the important fact that nearly all SIMDs that have been proposed and built could not handle the ATC requirement because of I/O limitations, e.g. MPP by Goodyear Aerospace Corp and the Connection Machine by Thinking Machines. This I/O limitation is overcome by the MDA (multidimensional access) memory [5, 6, 7] of Figure 1. The flip network is a corner turning network that provides access to a bit slice of data, in the normal SIMD mode, or a word slice for I/O. The word direction of data is compatible with the concept of entries of a relational database record. The features of Figure 1 exist in STARAN and ASPRO machines [5, 6, 7, 8].

With the AP, set data processing, as exemplified in the SQL database language, can be realized at the hardware level. The AP can process a set of thousands of operands simultaneously. We, henceforth, call a sequence of operations on such a set of operands a **jobset.**

The AP is a much simpler hardware/software approach to a feasible solution for real-time command and control systems. We use our Nation's Air Traffic Control (ATC) system as an example problem that can be solved using an AP.

2. Definitions, assumptions, constraints

Some of our definitions, assumptions and constraints differ from those commonly accepted for multiprocessor scheduling. The justification for the somewhat simpler definitions is the simplicity of the AP. We use chapter 2 of [9] as a guide in what follows.

- **Definitions**

"A *real-time task* is an executable entity of work, which, at a minimum, is characterized by a worst case execution time and a time constraint."[9] In the AP each task consists of a worst-case set of jobsets. (Unlike average case, worst case, is based on the maximum functional requirements of the system under consideration.)

Compared with conventional understanding of a job as an instance of a task in the MP, in the AP we consider a *task* to be a sequence of jobsets. An example of a task is detection of conflicts between current flights. In this operation a future flight envelope is developed for each flight in the environment. Then one flight is simultaneously evaluated against all the other flights to see if some defined minimum separation may be violated. This is a jobset. Next, each flight is evaluated in similar fashion to all other flights completing a set of jobsets or a task. This AP solution is O(n) where n is the number of flights in the airspace. The usual MP algorithm for this task is $O(n^2)$. In particular a jobset involves simultaneous execution of multiple jobs. There are three types of jobsets: periodic, aperiodic, and sporadic. Each is assigned to a periodic task.

A *system period* P is the time during which the set of all tasks must be completed. The system deadline D is the constraint time for a system period. Task deadlines d are applied to each task. Deadlines could be hard, soft or firm. We consider only hard deadlines in the work that follows. If the summation of task times for the system period is less than D the static schedule is ready to use on line, otherwise it is infeasible and must be redesigned.

Periodic tasks T are real-time tasks that are activated (i. e. released) regularly at predefined times called release times. Each task has a period p. Different tasks may have different periods in a system period. Each task has a release time r deadline time d. Task constraints are release and deadline times.

Each task is statically scheduled. That static schedule, developed off-line, is used in the operational on-line system. The release time r_{i+1} of any task T_{i+1} is always greater than the deadline time d_i of the preceding task T_i. Since the AP has a single thread instruction stream, only one task can be executing at any time. Thus there is never a task executing that could interfere with execution of any other task.

Following Chapter 2 in [12], we next list the assumptions and constraints implicit in the AP solution to ATC.

- **Assumptions for AP in ATC**

1. All tasks are periodic. All are statically scheduled.
2. All tasks are ready to run at their scheduled release times.
3. All deadlines are known in the task release.
4. Tasks do not suspend themselves. However they can finish before their deadline.
5. Tasks are independent in that there is neither synchronization between them, nor shared resources, nor precedence dependencies. The static schedule fixes release times. Deadline times for each task are also fixed.
6. Overhead costs for interrupt handling are included in each task cost.
7. Task preemption cannot occur since single instruction prevents concurrent task execution. All tasks are non-preemptable.
8. Each task can have only one predecessor task and one successor task.

- **Constraints for AP in ATC**

1. Resource constraints - a task may require access to certain resources other than the processing system, such as, I/O buffers which are defined to retain information for non-conflicting use by scheduled tasks at each task release time.
2. Precedence relationships - tasks are statically scheduled in precedence order.
3. Concurrency constraints - tasks are not allowed concurrent access to resources.
4. Communication requirements - a system that has distributed elements requiring bi-directional communication paths is not used. Communication is limited to conventional system input/output.
5. Fault tolerance - when multiple instances of a task are executed for fault-tolerance, the different instances are executed simultaneously on different processor systems.
6. Criticalness - all tasks are considered critical to the mission, and none can be preempted or deleted without possible adverse effects.

3. Our AP solution

A command and control system may be considered to consist of a database and methods for processing transactions on that database. The ATC database can be represented as a relational database. It should be observed that the SIMD is the only known architecture that can directly implement a relational database as

originally described by A. E. Codd in 1970. This implies no ordering of rows or columns. There are no pointers except existence in a row of a table and relationships between entities. This configuration supports AP queries that can be designed in declarative terms similar to the widely accepted SQL database language.

3.1 Flight Plan/Track Conformance, Example Jobset

This section includes an example of a jobset that is regularly executed in the ATC environment. In this jobset the previous flight plan (FP) position is updated to current time, and evaluated to determine its conformance to the filed plan by comparing the plan and the current sensor based track information. A one to one relationship exists between the track and flight plan entities. The pilot is allowed a distance deviation K1 laterally, a distance deviation K2 along his flight plan, and an altitude deviation K3. If the flight is within that space, the flight is "on track". If the flight deviates more than K1 or if the flight deviates from the correct altitude more than K3, the deviation is flagged to the attention of the controller. If the flight is off along track (i.e., deviation more than K2), the flight plan current position is modified to the current track position. Table 1 shows estimated instruction and data accessing and processing times for an AP built using available current technology.

In the Flight plan conformance problem Xf, Yf, Hf, xd, yd, hd are the flight plan last update position and velocity increment. Xt, Yt, Ht are the current track position; sin(hdg), cos(hdg) hdg is the flight plan heading parameter. Xf_1, Yf_1, Hf_1 are the updated flight plan position. Each step in Table 1 is executed simultaneously by all active ALUs.

It is seen from Table 1 that 7.34 microseconds is the total time required to evaluate track conformation for the worst-case set of 4,000, or maximum, IFR flights in the environment. There is no movement of data or instructions except between memory and processor. (It should be noted the execution time would be the same if there were only one flight in the system.) Thus, this is an O(1) time. The usual MP algorithm for this problem is O(n) and requires 4,000 iterations.

The analysis in Table 1 is based on work that was done on the STARAN [5, 6, 7] and ASPRO [8] architectures. In these machines the time for each instruction, at the object code level, was available to the developer. For example, in STARAN, an equal comparand (exact match) operation over the entire set of operands was executed in 0.82 + .19n microseconds, where n is the number of bits in the operand. A maximum field search took 0.84 + .63n microseconds. An add fields instruction was executed in 5.7 + .68n microseconds, where n is the number of bits in the operands. These operations were done with a single bit ALU since this was the state of art in 1972 [6, 7]. Similar performance was realized with ASPRO for the USN where tracking was both predicted and measured to be 276 time faster than a dual processor for an environment of 4,000 sensor reports, as reported in [10, 11]. Today's AP is assumed to have a number of general purpose registers and a 32 bit wide ALU which would allow accessing data only once, storing the data in registers, and then executing the required operation and storing the results back to memory.

Operations	Memory Accesses	Time (μs)
1 Select all flight plans	2	
2 Get Xf, Yf, Hf, xd, yd, hd	116	
3 $Xf_1 = Xf + xd$, $Yf_1 = Yf + yd$, $Hf_1 = Hf + hd$	6	0.06
4 Get Xt, Yt, Ht, sin(hdg), cos(hdg)	116	
Calculate displacement of track from FP		
5 X'=(Xt-Xf)*cos(hdg)+(Yt-Yf)*sin(hdg)	8	0.12
6 Y'=(Yt-Yf)*cos(hdg)-(Xt-Xf)*sin(hdg)	8	0.12
7 Check X'>K1, if true set alert flag	3	0.06
8 Check Ht – Hf > K3, if true set alert flag	4	0.06
9 Check Y'>K2, if true, update FP	4	0.08
10 Store Xf_1, Yf_1, Hf_1	75	
Total	342	0.50
Total proc. time with 20 ns memory access time		7.34 μs

Table 1 Flight Plan/Track Conformance Processing

3.2 An AP static schedule for ATC

A table driven scheduler [1, 9] is static, and is designed, off-line. The resulting fixed schedule of tasks provides sufficient time for the worst-case deadlines of the complete set of system tasks for the ATC system to be met. Since a static schedule is used, no execution time is used for scheduling. The periodic tasks run at their release times and each is completed by its deadline. If the summation of run times evaluated over the table of tasks is less than the major period the system is feasible.

Creating a static schedule begins with identification of the required jobsets. An example was given in 3.1 above. Observe that an aperiodic jobset might be a single job such as updating runway direction or weather at a specific terminal. When the jobsets are defined each of them is evaluated to determine its computation time j_j. Then each jobset is assigned to a task T on the basis of its function and repetition period. The summation c of the worst-case set of jobset times j_j assigned to each task is determined.

Some jobsets may be aperiodic or sporadic. These jobsets will not have a regular period and will be assigned to a task that handles all the aperiodic jobsets that have arrived within the last period. We use a one second period as shown in task 4 in Table 3. During this task all aperiodic jobsets that have arrived are processed. We assume there will be no more than 200 such jobsets per second that require less than 250 microseconds.

Each task T has an i^{th} release time r_i, a worst-case computation time c, and a period p. Its i^{th} deadline $d_i = r_i + c + .01$. The next release time in the table for each task is the current release time plus its period i.e., $r_{i+1} = r_i + p$. All tasks are non-preemptable. Concurrent tasks are

impossible since there is only one instruction stream and only one task can be executing at any time. The processor will idle when a task finishes early. (This condition is expected to ensue most of the time.) On the other hand it is necessary that all tasks be capable of executing under the worst-case set of conditions.

Each task is scheduled in precedence order. Each task has a single predecessor task and a single successor task. It should be remembered when discussing processing time that all tasks consist of jobsets that are executed at run time. Further, tasks may be composed of different jobset types.

Each jobset within a task executes a set operation. That is, each jobset can execute the same operation simultaneously on multiple data items. Additionally the data for a task may involve a worst-case situation We emphasize the set operations because the AP is a set processor. In this example we suggest the maximum set of operands may be of the order of 16,000. An operation over the entire set of records is executed simultaneously (i.e., in lock step) over all the data using one instruction stream that is executed only once. Table 2 shows the worst-case environment for the ATC tasks we consider.

When the above tasks are evaluated, using today's technology in AP design, the above tasks can be completed in less than 50% of the time available in the 8-second system period. Similar performance has been shown in [7, 11]. Table 3 and Figure 2 show our static schedule for ATC tasks in an AP.

Reports per second	12,000
IFR flights	4,000
VFR/backup flights	10,000
Controllers	600

Table 2. ATC – Worst-Case Environment

	Task	p	j	c	d	Proc time
1.	Report Correlation & Tracking	.5	15	.09	.10	1.44
2.	Cockpit Display 750 /sec)	1.0	120	.09	.20	.72
3.	Controller Display Update (7500/sec)	1.0	12	.09	.30	.72
4.	Aperiodic Requests (200 /sec)	1.0	250	.05	.36	.4
5.	Automatic Voice Advisory (600 /sec)	4.0	75	.18	.78	.36
6.	Terrain Avoidance	8.0	40	.32	2.93	.32
7.	Conflict Detection & Resolution	8.0	60	.36	3.97	.36
8.	Final Approach (100 runways)	8.0	33	.2	6.81	.2
	Summation of Tasks in a period P					4.52

The system period P (in which all tasks must be completed) is 8 seconds
p the task period time, is used to determine the next task release time $r_{i+1} = r_i + p$,
j is the execution time, in microseconds, for each jobset in a task,
c is the cost for each task for the worst-case set of jobsets,
d the deadline time for each task $r_i + c + .01$ (includes 10 ms interrupt processing per task)

Table 3. Statically Scheduled Solution Time

3.3 Analysis of our solution

The AP provides a simpler solution to the air traffic control automation problem. In addition to the 250 to 500 times or more increase in data bandwidth that can be realized the AP virtually eliminates the need for the following real-time processing software in the ATC multiprocessor environment**:**

- processor assignment
- queue management software
- task predictability evaluation
- data assignment
- concurrency control
- memory coherency management
- table and data locking
- cache management
- multitasking
- indexing
- sorting
- mosaicing
- data pointer assignment and update
- preemption control
- iterative processing of items of data.

The AP not only eliminates the extensive and complex software needed in a multiprocessor, but also eliminates the processing time needed to execute that complex software.

Static scheduling assures the predictability of system performance. A static schedule is both very reliable and highly adaptable and can easily be modified to incorporate system changes as needed.

3.4 A different system approach

When compared with any of today's MP techniques, "Associative processing is a different way of storing, manipulating and retrieving data compared to traditional computational techniques." [12]. One main feature of the AP is a set of processing elements (PEs or ALUs) that can be used to realize a content addressable memory [6]. As previously shown an AP system provides very efficient and natural support for the common database processes such as insert, retrieve, update and delete over a set of records. Further the AP eliminates many operations that are ubiquitous in MP systems such as indexing, sorting, moving or distributing data etc. None of these operations can change data.

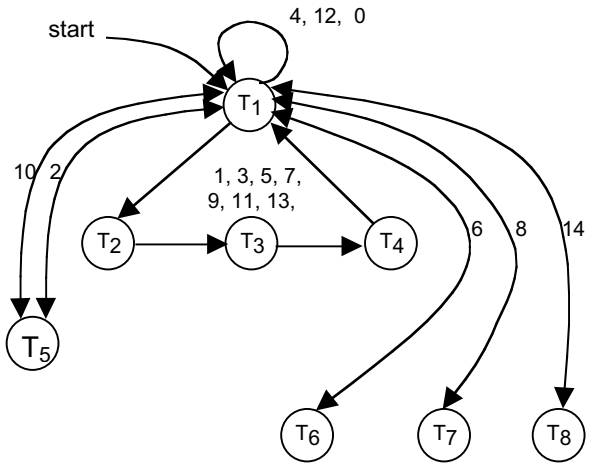

Figure 2. AP ATC schedule

An example is data access. All data in any relational table must be accessed on the basis of content. In the MP any significant amount of textual or positional data must be indexed or sorted. Maintaining a sort or index path to the data and following this path until the desired object is found achieves content addressed access. In many applications many secondary and cluster indexes are developed to increase access speed. In highly dynamic real-time systems such as air traffic control the cost of updating a large set of indexes in a real-time system can be very high.

In contrast such index/sort structures are never necessary in an AP since the stored data is physically content addressable [6, 12, 17]. That is, the characteristic of the desired data item is entered and in one instruction cycle is found or is determined to be missing. Similarly a point in 3-D space is found in a single instruction sequence by establishing a three-dimensional search box about the point sought. If they exist, the desired point or points in this space are easily and quickly located using a constant time search.

Both text and position search operations can be made over a set of items stored in memory. While such a search is order O(1) in an AP, the algorithms used in a MP (ignoring the additional costs mentioned in Section 3.3 and 3.5) is order O(n) in the MP. When using indexing or sorting in the MP, the O(n), may be reduced at the cost of establishing and maintaining an index plus the cost of using the index. In any case, the cost of computation and the number of instructions executed for these operations in an MP is typically much larger than the same operation in an AP. Also, the detection of conflicts between flights are order O(n) in an AP but the normal algorithm for this is $O(n^2)$ in a multiprocessor This occurs because, as discussed earlier, one flight projection is evaluated for minimum clearance with the set of other flight envelopes in the AP. In an MP each flight must be evaluated against every other flight individually.

4. MP Limitations

We next list and discuss several difficulties that occur whenever a MP system is used for scheduling real-time problems. This will lead to intractability of most real-time scheduling problems requiring a solution using a MP. In what follows we do not distinguish between different classes of multiprocessors. The only criteria we consider significant are the possibility of concurrent processes executing in the machines. It should be noted that a single processor may exhibit the same problems when operating under a multitasking operating system.

- Multitasking

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 [13] [SS8], 17). The MP has multiple instruction streams; therefore, tasks must be partitioned to allow concurrent executions while carefully avoiding the undesirable effects of concurrent operations. Consequently, most scheduling problems for multiprocessors are NP-hard [3].

- Shared resources

Shared resources usually cause NP-hardness in real-time scheduling problems [2, 9]. In 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 as shown earlier using the partition problem [9].

- Preemption

When preemption on an MP 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 MP systems. Inclusion of these difficulties will substantially increase the complexity of a solution to the real-time scheduling problem.

- Precedence constraints

The existence of precedence constraints can be another reason for NP-hardness in real-time scheduling problems. 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, an MP has 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 [9]. In contrast the AP schedules real-time tasks for air traffic control statically.

It takes advantage of its synchronous data parallel capabilities and uses a predefined precedence to assure enough time to complete worst-case time tasks. The order of task execution is stored in the table driven scheduler.

- Concurrent instructions

When the ATC problem is considered from a functional point of view, there seems to be no reason why the problem should be intractable. All the parameters in the ATC problem have a limited upper bound. None of the flights, sensor reports, terminals, controllers, control sectors, etc. are increasing exponentially. Then why do multiprocessor problems exist? They are the result of concurrent instruction streams [14]. Other reasons for these problems given in [3] and [9] involve the concurrent execution of multiple tasks The AP eliminates these problems by eliminating the condition that causes them. That is, the problem disappears through elimination of concurrent operations on the database. A single instruction stream cannot have concurrent actions.

We next quote from other researchers concerning real-time processing.

Klein et al [15] write about real-time scheduling in an MP: "It is tempting to think that speed (for example, processor or communication bandwidth) is the sole ingredient in meeting system timing requirements, but speed alone is not enough." The problem, according to Klein, et al is "…predictability, the ability to determine for a given set of tasks whether the system will be able to meet all the requirements of those tasks." We can see that All current ATC approaches use "probabilistic" or "dynamic" scheduling which is inherently unpredictable. Klein, et al continue; "Optimal methods are seldom useful in real-time systems because most realistic problems incorporating practical issues such as task blocking, and transient overloads are NP-hard." Garey, Graham and Johnson, write: "In fact, all but a few schedule optimization problems are considered insoluble. For these scheduling problems no efficient optimization algorithm has yet been found, and indeed, none is expected."[16]. "The problem of determining an optimal schedule even in a multi-processor system is known to be NP-hard. … These factors often necessitate a *heuristic* approach …" p. 247 [9].

Stankovic et al [3]: state that "One of the goals of hard real-time scheduling is to find feasible schedules for a multiprocessor system. It is a difficult problem and has been shown to be NP-complete …even for simple models" they further write: "…complexity results show most real-time multiprocessing scheduling is NP-hard."

5. Conclusion

We show a simple predictable polynomial time solution for our nations air traffic management problems. While all current MP approaches use a heuristic scheduling algorithm that is dynamically dependent on the current system state, the AP schedule is developed as part of the initial system programming effort and remains static and unchanged at run-time. We believe a working model can be fabricated and programmed to manage a system by using the schedule shown in Table 3. and performing the functions on the ATC environment shown in Table 2. Development of the necessary hardware and software, based on prior completed work, will be straightforward and simple. Because of the systems simplicity and performance predictability the risk of failure is near zero. In the view of the repeated failures of MP ATC systems from the late 1960s to the current time it is time to try a better approach. It only has to be done.

References:

[1] C. Murthy, G. Manimaran, "Resource Management in Real-Time Systems, MIT Press, 2001

[2] 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.

[3] 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.

[4] J Yang, "$2 Billion Control System Not Ready for Use.", ABC News, June 5, 2002 http://abcnews.go.com/sections/wnt/DailyNews/yang_faa020605.html

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

[6] W. C. Meilander, "STARAN an associative approach to multiprocessing", Multiprocessor Systems, Infotech State of the Art Reports, Infotech International 1976 pp 347-372.

[7] J. A. Rudolph, "A Production Implementation of an Associative Array Processor – STARAN", The Fall Joint Computer Conference (FJCC) December 5-8, 1972, Los Angeles, California, USA

[8] ASPRO-VME Hardware/Architecture ER3418-5 LORAL Defense Systems Akron, OH June 1992

[9] "Deadline Scheduling for Real-time Systems" John Stankovic et al, Kluwer, 1998

[10] Will C. Meilander, Jerry L. Potter, Kathy Liszka, Johnnie W. Baker; "Real-Time Scheduling in Command and Control." Midwest Parallel Processing Conference, August 20, 21 1999. (Unofficial version at www.cs.kent.edu/~parallel)

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

[12] A. Krikelis, C. C. Weems "Associative Processing and Associative Processors" IEEE Computer Society, 1994

[13] Garey and Johnson "Computers and Intractability A Guide to the Theory of NP-Completeness", W. H. Freeman, 1979.

[14] J. Potter, W. Meilander " Interarchitecture Comparative Analysis", Proc. International Conference On Communications In Computing, June 26, 2000, See www.cs.kent.edu/~parallel

[15] Mark H. Klein, et al "Rate-Monotonic Analysis for Real-Time Industrial Computing," Computer, January 1994.

[16] M.R.Garey, R.L.Graham and D. S. Johnson, "Performance Gaurantees for Scheduling Algorithms,"*Operations Research.* Vol. 26, No. 1, Jan,-Feb. 1978 pp 3-21

[17] "The Power of SIMDs vs. MIMDs in Real-Time Scheduling", Mingxian Jin, Johnnie W. Baker, and Will C. Meilander, Proc. of the 17th International Parallel and Distributed Processing Symposium (Workshop in Massively Parallel Processing), IEEE Digital Library, April 2002 (Unofficial version: www.cs.kent.edu/~parallel)