Research Issues for Executing Real-Time C3 Applications on Parallel Processing Systems*

Richard C. Metzger
Rome Laboratory/C3CB
Software Engineering Branch
525 Brooks Road
Griffiss AFB, NY 13441-5700

John K. Antonio
School of Electrical Engineering
1285 Electrical Engineering Building
Purdue University
West Lafayette, IN 47907-1285

Abstract

Current trends indicate that future C3 (command, control, and communications) systems will likely contain suites of heterogeneous computing facilities composed of both parallel and sequential computing components. Also, "commercial-off-the-shelf" components are being heavily considered for use in these future systems. In the context of these trends, brief overviews of related work within the areas of parallel processing and real-time computing are given and areas of future research are outlined. A central theme throughout the paper is that in order to make effective use of future C3 platforms, a significant amount of "cross fertilization" between researchers in the parallel processing and real-time computing communities will be required. As an example of the need for combined expertise in both areas, the problem of how to effectively allocate real-time tasks onto the processing elements of a hypercube architecture is discussed. Other research issues, including the question of how to cope with the degree of nondeterminism present in many commercial parallel processing systems, are also discussed.

1 Introduction

With rapid advances being made in technology for information acquisition for C3 systems, the Department of Defense has recognized the need for extensive research in high-performance real-time information processing. The Global Surveillance and Communications thrust is an example of an area that requires technology for acquiring information from remote sensors, transmitting information from remote senors to a command and control center, and processing information so that orders can be issued to investigate and/or counter threats. The command and control centers for Global Surveillance and Communications Systems may be ground based or airborne.

In general, command and planning centers must have the computing power to make sense of incoming information, respond to new threats, and plan missions to counter threats. Recently, much of the required processing power has come in the form of distributed networks of workstations. Advantages of using networks of workstations include the relative ease of programming, well supported operating systems, and a large user base in the industrial sector.

As high-performance parallel processing systems become more suited physically (in terms of size and ruggedness), they too will find their way into operational use in command and control centers. Some airborne systems are already being developed such as the nCUBE/Microlithics hypercube [9]; also, the Defense Advanced Research Projects Agency (DARPA) has recently sponsored an effort with Intel and Honeywell to produce an airborne Touchstone machine. Unfortunately, a systematic software development process for such systems, especially real-time information processing systems, is still largely undefined. One difficulty associated with developing real-time software for parallel systems is that the degree of nondeterminism in (some) parallel systems makes it difficult to determine schedules for resources such as the processors and memory modules. This nondeterministic timing problem is in direct conflict with the inherently time-critical nature of real-time C3 applications.

The types of real-time computations required by C3 systems cover a wide spectrum: the deadlines for completing a computational task may be soft or hard and the repetition of the computational tasks may be periodic or asynchronous. A deadline for a real-time task is said be a hard deadline if completing the computation after the deadline is of no value and/or results in catastrophic consequences. For instance, from the time an incoming enemy missile is first detected,

*This work was supported by Rome Laboratory under contract F30602-92-C-0108 and the Air Force Office of Scientific Research under Task 2304F2.

0-8186-4130-4/93 $03.00

the computation required to effectively counter the attack must be delivered before a critical time deadline, i.e., before the missile enters a critical zone. A task with a soft deadline implies that while completing the computation before the deadline is desirable, occasionally missing the deadline can be tolerated [19]. Examples of computational tasks that may have soft deadlines include certain image/speech processing tasks, statistical processing of archived data, and data-base queries. A periodic task is a task for which the repetition of its execution is known and fixed. Aircraft flight control systems and flight simulators require the computation of periodic real-time tasks, e.g., in a flight control system, the task of computing the control signal for driving an actuator of a control surface is a periodic task. An asynchronous task is one in which the demands for its execution are either unknown or unpredictable. Certain tasks within C3 that are activated by sensory information, e.g., the "determination of 'friend or foe' of an approaching aircraft," could be classified as asynchronous real-time tasks.

The remainder of the paper is organized in the following manner. In Section 2, brief overviews of related research in the areas of parallel processing and real-time computing are given. Section 3 describes an example research problem—how to effectively map periodic real-time tasks onto a hypercube architecture—as a means of demonstrating the importance of having expertise from both the parallel processing and real-time computing areas. In Section 4, some general research issues are briefly outlined in the context of how to effectively use parallel processing systems for real-time C3 applications. A summary of the paper and some concluding remarks are included in the last section.

2 Related Research

Brief overviews of related work, in the areas of both parallel processing and real-time computing, are included in this section. These overviews include only a representative subset of important topics within each area and should not be viewed as complete tutorials. The overviews are presented to demonstrate the range of issues that must be considered in order to make effective use of parallel processing platforms for use in real-time applications. Just as the discussions within the overviews contain only a representative subset of important topics from within each area, the cited references for the various topics are also only a representative few.

2.1 Parallel Processing

The basic concept of parallel processing is to perform computation by decomposing a given task into sub-tasks and executing these sub-tasks on independent processors. The goal is to decrease the total time required to complete the overall task by simultaneous execution of the sub-tasks. While the philosophy of parallel processing is clear, and while there are both practical and theoretical success stories, there are still many open questions. To illustrate the range of unresolved issues that exist in parallel processing, consider the following representative list of questions. To a certain degree, each of these questions is associated with an active area of research.

- How much parallelism is present within any given problem domain?
- Can general sequential programs be effectively parallelized using automated compiler techniques?
- How much parallelism should the user be expected to detect and exploit?
- How much parallelism should the compiler be expected to detect and exploit?
- What attributes should the language for parallel processing systems have?
- What system model should the programmer use in order to develop parallel software?
- What types of meaningful system models can be realistically implemented or approximated?
- What is the "best" general purpose parallel processing architecture?

With so many unresolved (and perhaps unresolvable) questions, one may ask "Why consider parallel processing?" and/or "Why not wait one year for the speed of sequential machines to double?"

The fact that the speed of light is a constant, and thus there is a definite upper bound on the attainable speed of sequential machines, provides some motivation for investigating the potential of parallel processing. Another more practical motivator involves economics. For certain application domains, some commercially available massively parallel processing systems have demonstrated cost/performance ratios superior to those of the more traditional state-of-the-art vector supercomputers. Cray Computers Inc., perhaps the most successful manufacturer of vector machines, has recognized that massively parallel processors will eventually be the only way to continue to increase performance [13].

Perhaps the most popular taxonomy for parallel processors is the concept of the multiplicity of instruction stream and data stream, introduced by Flynn [8]. A single instruction stream - multiple data stream (SIMD) system executes the same thread of control (i.e., instruction stream) on all processors and each processor executes the instructions on distinct (locally stored) data. The term "SIMD" is often used synonymously with the term "data parallel." A multiple instruction stream - multiple data stream (MIMD) system is able to execute multiple independent threads of control, one or more on each processor. The term "MIMD" is often used synonymously with the terms "control parallel" and "functional parallel."

Machines that support the SIMD mode of parallelism typically have a special processor called the control unit that executes the single thread of control and broadcasts instructions to an array of processors that operate on there own data memories. Thus, the processors of an SIMD machine execute the same instruction at the same time (in lockstep synchronization) on distinct data. An example of a commercially available SIMD computer capable of supporting up to 16,384 processors is the MasPar MP-1 system, see [4] for a detailed description of its architecture.

Machines that support the MIMD mode of parallelism consist of an interconnected collection of independent processors, each capable of executing its own independent thread of control. An example of an MIMD machine capable of supporting up to 8,192 processors is the nCUBE 2 system [12].

It is generally acknowledged that the SIMD and MIMD modes of parallelism each have their own sets of advantages and disadvantages. For instance, it is usually agreed that SIMD machines are easier to program than MIMD machines; however, MIMD machines allow more flexibility for expressing parallelism and thus offer the (potential) advantage of being suitable for a wider range of application domains.

Some experimental machines have been proposed and/or built that are capable of supporting both modes of parallelism. An example of such a "mixed-mode" machine is the the PASM (*p*artionable *S*IMD/*M*IMD) system, which offers the advantage of being able to switch between modes of parallelism at instruction level granularity [16].

An important issue in the parallel processing area is how to effectively map parallel algorithms onto parallel architectures. Because the interconnection networks employed by different machines have a wide range of characteristics and properties, the best mapping strategy for one architecture is generally not the best strategy for another. For more information on the various types of interconnection networks proposed and/or used in parallel processing systems, refer to [14]. To appreciate the range of strategies and techniques developed for mapping tasks onto parallel systems, compare the technique described in [15] with that described in [1].

2.2 Real-Time Computing

An important issue in real-time computing is the scheduling of tasks to insure that deadlines are met. Failure to adhere to deadlines can lead to degraded system performance and/or loss of life or property.

There are two major approaches to solving the scheduling problem. The first is dynamic scheduling, where the scheduling of tasks is determined on-line as processes arrive [19]. The appeal of dynamic scheduling is that tasks can be scheduled for execution as a need arises and/or as resources become available in the system. This may be especially useful in environments where there are many asynchronous real-time tasks to be executed. An important issue in dynamic scheduling research is to devise fast scheduling algorithms so that the amount of time spent executing the scheduling algorithm does not infringe on the performance of the system. Refer to [19, 20] for more detailed discussions of dynamic scheduling techniques.

A second scheduling approach is static scheduling, where the scheduling of tasks is determined off-line, before execution begins. This type of scheduling technique requires that certain attributes of the tasks be known in advance in order to compute a feasible schedule (if one exists). Static scheduling techniques are especially well-suited for environments where the majority of real-time tasks are periodic. Refer to [19] for more a detailed discussion of static scheduling techniques.

A well-known result for the scheduling of periodic tasks on a single processor system is the so-called rate monotone algorithm [11]. The rate monotone algorithm is a preemptive, priority-driven algorithm (priorities are assigned to tasks proportional to their rates of repetition). Given a collection of K periodic tasks with execution times and repetition periods denoted by τ_i and p_i, respectively, the rate monotone algorithm guarantees to provide a feasible schedule (i.e., no missed deadlines) provided that the following inequality is satisfied:

$$\sum_{i=1}^{K} \frac{\tau_i}{p_i} \leq K(2^{1/K} - 1).$$

The expression on the left-hand side of the equation represents the processor utilization. It is easy to show that the right-hand side of the equation is bounded

above by unity for all values of $K \geq 1$, and it approaches $\ln 2$ as $K \rightarrow \infty$.

The problem of how to schedule real-time tasks for multiprocessor systems has also been studied by the real-time computing community. A common approach to the problem of scheduling real-time tasks on multiple processors is to first assign (i.e., map) the tasks onto the processors once and for all, and then schedule the tasks assigned to each processor independently of the tasks assigned to other processors [6]. In order to decrease the likelihood of a missed deadline, it is desirable in practice to assign the tasks so that the processor utilizations are as uniform as possible across all processors. In [3], a simple greedy heuristic for assigning tasks to processors is analyzed in terms of how its assignments compare to an optimal assignment (where an optimal assignment is one that yields a minimal variation in processor utilizations). It is proven in [3] that the simple heuristic algorithm produces near optimal assignments (based on the variance of processor utilizations) under relatively mild conditions.

An important assumption made in [3], and one that is typically made for the task assignment problem (within the real-time computing area), is that the cost of interprocessor communication is independent of the assignment. While this assumption is reasonable for some multiprocessor systems designed specifically for real-time applications (which often employ a common bus for interprocessor communication, see for example [18]), it is not generally valid for commercially available massively parallel processing systems.

3 Mapping Periodic Real-Time Tasks onto the Hypercube Topology

The problem of how to map periodic real-time tasks onto an MIMD hypercube architecture is discussed in this section. The hypercube structure has been a popular choice for interconnecting large numbers of processing elements in parallel processing systems. Examples of commercially available MIMD machines that utilize a hypercube interconnection topology include nCUBE's nCUBE 2 and Intel's iPSC2.

3.1 The Hypercube Topology

An n-dimensional hypercube has two connected nodes along each of n dimensions for a total of $N = 2^n$ nodes. By labeling the nodes from 0 to $N-1$, the node-to-node interconnection pattern is conveniently defined using these node labels as follows: there is a direct communication link between two nodes if and only if the binary representation of their addresses differ in exactly one bit position. For example, in a 3-dimensional hypercube, node 0 is directly connected (only) to nodes 1, 2, and 4. Some of the attractive features of the hypercube are: a relatively low number of incident links at each node (node degree $= n = \log_2 N$), a small hop distance between nodes (network diameter $= n = \log_2 N$), and a large number of alternate paths between node pairs.

3.2 The Hypercube Embedding Problem

The standard hypercube embedding problem (as typically defined in the parallel processing literature) is to map a given collection of tasks onto the processors of the hypercube topology so that the available communication resources are effectively utilized. Assumed to be given is the inter-task communication demands. For general communication patterns, most formulations of the hypercube embedding problem are known to be NP-Hard [7].

A specific objective often used for the hypercube embedding problem is to minimize the average Hamming distance (i.e., path length) between those pairs of tasks that require communication. For the case of circuit-switched and virtual cut-through routing schemes [10], minimizing the average distance between communicating process pairs reduces the total number of communication links needed to establish all required connections and can therefore potentially reduce the latency caused by contention for common links.

Most of the proposed algorithms for solving the hypercube embedding problem assume that N or fewer tasks are to be mapped onto the N processors of a hypercube. For a thorough discussion and evaluation of such techniques, refer to [5] and the references therein.

In [1], a nonlinear programming approach is introduced for solving the hypercube embedding problem. The basic idea of the proposed approach is to approximate the discrete space of an n-dimensional hypercube, i.e., $\{z : z \in \{0,1\}^n\}$, with the continuous space of an n-dimensional hypersphere, i.e., $\{x : x \in \Re^n \ \& \ ||x||^2 = 1\}$. The mapping problem is initially solved in the continuous domain by employing the gradient projection technique to a continuously differentiable objective function. The optimal tasks "locations" from the solution of the continuous hypersphere mapping problem are then discretized onto the n-dimensional hypercube. Unlike most past approaches, the technique proposed in [1] can solve, directly, the problem of mapping K tasks onto N nodes for the general case where $K > N$.

3.3 Mapping Real-Time Tasks onto the Hypercube

As discussed in Subsection 2.2, when mapping real-time tasks onto multiple processors, it is important to achieve balanced utilization of the processors. Like-

wise, therefore, when mapping real-time tasks onto a hypercube architecture, the objective of producing uniform processor utilizations should somehow be incorporated. In [2], the objective function of the hypersphere mapping algorithm of [1] is modified to reflect a weighted combination of the average distance between communicating tasks and the variance of the processor utilizations. Thus, by minimizing this objective function, the algorithm searches for mappings that simultaneously balance processor utilizations and minimize the effect of network contention. For certain types of communication patterns, this technique is shown to have advantages over a more straightforward technique of first allocating the tasks into N relatively uniform groups (using, for example, the greedy allocation algorithm described in [3]) and then applying a standard hypercube embedding algorithm based on the inter-group communication pattern associated with the N groups.

4 Other Research Issues

4.1 Task Partitioning

As discussed in [3], there are two basic approaches to using multiple processor systems for executing real-time tasks. The first approach is called the partitioning method, which seeks to partition a collection of tasks into groups and assign the groups to distinct processors. The discussion of the mapping problem described in the previous section for hypercube architectures is an application of the partitioning method. The second approach is called the nonpartitioning method, which treats the entire collection of processors as a powerful "virtual processor," and uses this increased processing power to quickly execute the entire set of tasks sequentially.

The question of how to apply the nonpartitioning approach is closely aligned with some of the research issues and questions found within the parallel processing community (refer to the list of questions listed in Subsection 2.1). The following important distinction can be made, however. Within the parallel processing community, the primary reason for exploiting parallelism is to decrease the execution time. Actually, because of the nondeterministic timing behaviors of many commercially available machines, perhaps a more accurate statement of this goal is to decrease the *expected* execution time.

In real-time environments, not only is it desirable to decrease the expected execution time, it is generally also important for the execution time to be highly predictable (i.e., of low variance). As the variability of task execution times increase, it becomes increasingly difficult to effectively schedule tasks to guarantee that all deadlines are always met. Of course, very conservative schedules could be used to account for large execution time variances, but such an approach may defeat the purpose of using a high performance parallel system, because conservative schedules may result in unacceptably low processor efficiencies.

4.2 Operating Systems

In order to effectively apply the partitioning method on an MIMD system for executing real-time tasks, it is necessary for the operating system (usually resident on each processor) to have some real-time task scheduling capability. While recent strides have been made in implementing real-time task scheduling functions within the domain of operating systems for workstation environments, it is not clear if porting this technology into the lean operating system kernels found on most massively parallel MIMD processors is a viable and cost-effective option.

4.3 I/O

Many commercially available parallel systems are not designed to provide high-throughput bursty I/O. There may be, however, clever ways to overlap computation and communication on such systems so that the I/O sub-systems are not a bottleneck for real-time applications.

4.4 Software Development Process

The software development process for real-time parallel systems is clearly distinct from any current sequential software development process model. For instance, the problem of how to map real-time tasks onto various classes of architectures is a new problem that must be addressed at some phase of the software development cycle. In order to effectively maintain and upgrade complex parallel software for C3 applications, a meaningful software development process model must be established and adhered to.

5 Conclusions

The next generation of C3 systems will likely contain suites of massively parallel processing platforms for executing real-time tasks. The primary goal of this paper has been to demonstrate the necessity for a combined effort between researchers in the parallel processing and real-time computing communities.

As noted in [17], there is evidence that some degree of confusion exists concerning how the issues important to real-time computing are influenced by assuming a massively parallel computing platform. An attempt was made in the present paper to clear up some of this confusion by giving brief overviews of research issues important to both the parallel processing and

real-time computing communities. A specific research problem involving the mapping of real-time tasks onto a popular type of parallel architecture was introduced to demonstrate the importance of having expertise in both areas.

Representative examples of other more general issues associated with the use of massively parallel processing systems for real-time computing were briefly discussed. These issues represent fertile ground on which significant research can grow.

References

[1] J. K. Antonio and R. C. Metzger, "Hypersphere mapper: a nonlinear programming approach to the hypercube embedding problem," *Proc. 7th Int'l Parallel Processing Symposium*, Apr. 1993, pp. 538–547.

[2] J. K. Antonio and R. C. Metzger, "Mapping periodic tasks onto hypercube architectures," Technical Report, School of Electrical Engineering, Purdue University, (under preparation) 1993.

[3] J. A. Bannister and K. S. Trivedi, "Task allocation in fault-tolerant distributed systems," *Acta Informatica*, **20**, Springer-Verlag, Heidelberg, 1983, pp. 261–281.

[4] T. Blank, "The MasPar MP-1 architecture," *Proc. IEEE Compcon*, Feb. 1990, pp. 20–24.

[5] W. K. Chen, M. F. M. Stallman, and E. F. Gehringer, "Hypercube embedding heuristics: an evaluation," *Int'l J. Parallel Programming*, vol. 18, no. 6, 1989, pp. 505–549.

[6] J.-Y. Chung, J. W. S. Liu, and K.-J. Lin, "Scheduling periodic jobs that allow imprecise results," *IEEE Trans. Computers*, vol. C-39, no. 9, Sept. 1990, pp. 1156–1174.

[7] G. Cybenko, D. Krumme, and K. Venkataraman, "Fixed hypercube embedding," *Information Processing Letters*, **25**, 1987, pp. 35–39.

[8] M. J. Flynn, "Very high-speed computer systems," *Proceedings of the IEEE*, vol. 54, no. 12, Dec. 1966, pp. 1901–1909.

[9] M. Hewish, "Integrated avionics: the heart of future combat aircraft," *Defense Electronics & Computing*, no. 9, 1992, pp. 1007–1011.

[10] P. Kermani and L. Kleinrock, "A tradeoff study of switching systems in computer communications networks," *IEEE Trans. Computers*, vol. C-29, no. 12, Dec. 1980, pp. 1052–1060.

[11] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *J. ACM*, vol. 20, no. 1, Jan. 1973, pp. 41–61.

[12] nCUBE Corporation, *nCUBE 2 Processor Manual*, Order # 101636, nCUBE Corporation, Dec. 1990.

[13] S. Nelson, "Establishing an MPP guidepost," *Proc. Fourth Symposium on the Frontiers of Massively Parallel Computation*, Oct. 1992, pp. 471–473.

[14] H. J. Siegel, *Interconnection Networks for Large-Scale Parallel Processing: Theory and Case Studies, Second Edition*, McGraw-Hill, New York, NY, 1990.

[15] H. J. Siegel, J. B. Armstrong, and D. W. Watson, "Mapping computer-vision-related tasks onto reconfigurable parallel processing systems," *Computer*, vol. 25, Feb. 1992, pp. 54–63.

[16] H. J. Siegel, W. G. Nation, and M. D. Allemang, "The organization of the PASM reconfigurable parallel processing system," *Proc. 1990 Parallel Computing Workshop*, Ohio State University, Mar. 1990, pp. 1–12.

[17] J. A. Stankovic, "Misconceptions about real-time computing," *Computer*, Oct. 1988, pp. 10–19.

[18] C. B. Weinstock, "SIFT: system design and implementation," *Proc. Tenth Int'l Symposium on Fault Tolerant Computing*, 1980, pp. 75–77,

[19] J. Xu, and D. L. Parnas, "On satisfying timing constraints in hard real-time systems," *IEEE Trans. Software Eng.*, vol. SE-19, no. 1, Jan. 1993, pp. 70–84.

[20] W. Zhao, K. Ramamrithan, and J. Stankovic, "Scheduling tasks with resource requirements in hard real-time systems," *IEEE Trans. Software Eng.*, vol. SE-13, May 1987, pp. 564–577.