Parallelizing Existing Applications in a Distributed Heterogeneous Environment*

Michael A. Iverson, Füsun Özgüner

Department of Electrical Engineering
The Ohio State University
2015 Neil Avenue
Columbus, OH 43210

Gregory J. Follen

Interdisciplinary Technology Office
NASA Lewis Research Center
21000 Brookpark Road, M.S. 142-5
Cleveland, OH 44135

Abstract

Applications based upon the finite element method are well known for their demand for computational resources. An effective method for satisfying this demand is heterogeneous parallel computing. This paper presents the results obtained by applying heterogeneous computing to a large, existing finite element application code: CSTEM. A difficult problem associated with heterogeneous computing is the mapping and scheduling problem—the process of assigning the tasks of a parallel program to the individual processors. A simple assignment heuristic, Levelized Min-Time (LMT), is presented, along with simulated results from applying the LMT algorithm to heterogeneous CSTEM on a variety of different heterogeneous machine clusters.

1 Introduction

Performance gains in computer design are quickly consumed as users seek to analyze larger problems to a higher degree of accuracy. This trend is certainly true for applications used in the design of aircraft gas turbine engines (AGTE). In recent years, the reality of budget limitations has made the purchase of high-cost/high-speed computer architectures unattractive. Therefore, innovative computing techniques are required to satisfy the computational demands of large applications, while, at the same time, making more effective use of available computing platforms. One such computing technique is heterogeneous distributed computing, which combines a set of dissimilar computer architectures into a single computational entity.

Heterogeneous computing can be viewed in two ways: either as a means of increasing the performance of an application beyond the level it can achieve on any single machine [12], or as a means of reducing the cost of executing an application without affecting performance [1]. Either of these goals can be accomplished by breaking the application into fragments, and executing each fragment on the best suited architecture, in parallel where possible.

This paper will present results on the use of heterogeneous computing to increase the performance of CSTEM [17], a finite element based application for the design of composite AGTE components. A publicly available programming tool for heterogeneous computing, HeNCE [3, 4], was used to create a heterogeneous version of CSTEM.

The most difficult problem associated with heterogeneous distributed computing, as well as parallel computing in general, is the mapping and scheduling problem. This problem assigns the individual code fragments, or tasks, to the set of processors such that the overall completion time of the application is minimized. The mapping and scheduling problem is a very broad problem, taking different forms depending upon the processor architecture, the network architecture, and the task structure. It is also very costly to compute an exact solution to the mapping and scheduling problem, since the problem is NP-hard [19]. Therefore, heuristic methods are used to obtain approximate solutions. This paper introduces a new mapping and scheduling heuristic, the *Levelized Min-Time* (LMT) algorithm. To evaluate the performance of this algorithm, and the performance of heterogeneous CSTEM, a series of simulations were performed. These simulations estimated the performance of heterogeneous CSTEM on a variety of potential clusters of heterogeneous machines.

*This work was supported by NASA Lewis Research Center Grant NAG3-1440.

The remainder of this paper will be organized in the following form. Section 2 will present CSTEM and the process used to create a heterogeneous version. Section 3 will introduce the heuristic developed to solve the mapping and scheduling problem. Finally, Section 4 will examine the simulated performance of heterogeneous CSTEM, and Section 5 will offer some conclusions from these results.

2 Creating Heterogeneous CSTEM

CSTEM, an acronym for Coupled Structural-Thermal-Electromagnetic Analysis and Tailoring of Graded Composite Structures, is a finite element-based computer program developed for the NASA Lewis Research Center [17]. As its name implies, CSTEM analyzes and optimizes the performance of composite structures using a variety of dissimilar analysis modules, including a structural analysis module, a thermal analysis module, an electromagnetic absorption analysis module, and an acoustic analysis module. Large, coupled structures codes, like CSTEM, have huge demands for computational resources. CSTEM, for example, consists of approximately 81,000 lines of FORTRAN code, and requires several minutes of CPU time even for small, trivial problems.

Almost all of the analysis routines that make up CSTEM are called from a single FORTRAN subroutine. Data is passed between these routines using three different means: as parameters of functions, in *common* blocks, or in files. To complicate matters, these methods are often used interchangeably and inconsistently, largely because CSTEM derives a large portion of its code from existing applications.

2.1 HeNCE

HeNCE, an acronym for Heterogeneous Network Computing Environment, is an automated tool for the development of heterogeneous applications, developed at Oak Ridge National Laboratories. To create a heterogeneous application, the HeNCE programmer only needs to provide a set of C or FORTRAN function calls and a data dependency graph. HeNCE provides a graphical interface, called *htool*, for creating HeNCE applications and for graphical performance monitoring. HeNCE is built upon the PVM (Parallel Virtual Machine) message passing libraries [13], also developed at Oak Ridge.

2.2 Heterogeneous CSTEM

When constructing heterogeneous CSTEM, a number of factors needed to be balanced. The amount of programming effort needed to be minimized, while the overall performance improvement was maximized. While attempting to balance these factors, CSTEM was divided into 15 separate tasks. Each of these tasks has its own separate source code, simplifying the process of porting individual tasks to different architectures. Although, for the initial version of this code, none of the tasks used any data parallelism, individual tasks that are in the critical execution path can be parallelized using data parallel techniques to increase the overall performance. Figure 1 shows a representation of the resulting HeNCE program graph. The numbers within the nodes of the graph represent the approximate execution time, in seconds, on a Sun Microsystems Sparc 10 workstation, excluding any communication costs. The edges between the nodes represent the precedence relationships between the nodes. These edges do not represent all of the nodes which communicate, since adding all of the communication edges would have made the graph unreadable. The names assigned to the nodes are arbitrary names uniquely identifying each task. From the task execution times in the program graph, the tasks causing performance bottlenecks are clearly visible, making task F and task M definite candidates for parallelization.

3 Mapping and Scheduling

3.1 Definitions

The following set of definitions will be used in presenting the mapping and scheduling algorithm. The set of parallel tasks can be represented by a directed, acyclic graph (DAG) $G = (V, E)$, where the set of vertices $V = \{v_1, v_2, \ldots, v_n\}$ represents the set of tasks to be executed, and the set of directed edges E represents communication between tasks, where $e_{ij} = (v_i, v_j) \in E$ indicates communication from task v_i to v_j. The collection of heterogeneous machines used in the computation can be represented by the set $P = \{p_1, p_2, \ldots, p_q\}$.

The computation cost matrix $X_{n \times q}$ represents the execution costs of n tasks on q heterogeneous machines. The value $x_{ij} \in X$ represents the computation cost of task v_i on machine p_j. The communication matrix $C_{n \times n}$ holds the number of bytes sent between the tasks. The value of $c_{ij} \in E$ is equal to the communication volume if $e_{ij} \in E$, otherwise, $c_{ij} = 0$.

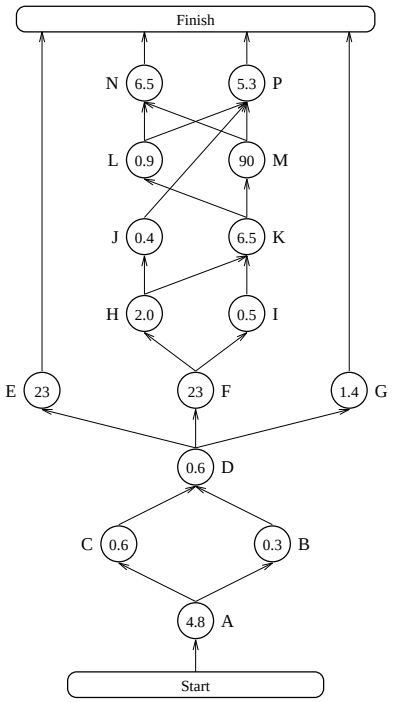

Figure 1: HeNCE Program Graph for Heterogeneous CSTEM

A solution to the mapping and scheduling problem is defined as $\mu : V \rightarrow P$, mapping the tasks onto the heterogeneous machines. Thus, task v_i is mapped onto machine $\mu(v_i)$. The communication cost function $\delta : \mathcal{N} \times P \times P \rightarrow \mathcal{N}$ defines the communication costs of a given mapping, where $\mathcal{N}$ is the set of natural numbers. The value $\delta(c_{ij}, \mu(v_i), \mu(v_j))$ represents the cost of sending c_{ij} bytes from task v_i on processor $\mu(v_i)$ to task v_j on processor $\mu(v_j)$.

A path W through the DAG $G = (V, E)$ is defined as a sequence of nodes such that, for all adjacent pairs nodes v_i and v_j in the sequence (v_i is ordered before v_j), $e_{ij} \in E$. The cost ϕ of a solution to the mapping and scheduling problem, for a given mapping, is defined as the path W through the graph that maximizes the sum of the communication costs and computation costs along that path. This can be represented by the expression:

$$\phi = \max_{W}\left(\sum_{v_i \in W} x_{i\mu(v_i)} + \sum_{v_i, v_j \in W} \delta(c_{ij}, \mu(v_i), \mu(v_j))\right) \quad (1)$$

3.2 Previous Work

There are a wide variety of different approaches that have been taken solve the mapping and scheduling problem. The methods that have been used include iterative methods [23], global optimization methods [22, 16], greedy selection methods [10, 14], hierarchical methods [5, 8], and combination methods [6]. Many of these methods do not explicitly consider the precedence relations that exist between tasks, and instead concentrate on mapping tasks onto the processors. These methods, in most circumstances, are not applicable to the type of task system defined above.

Therefore, methods which directly consider precedence relations will be emphasized. These methods can be broken down into two categories: those for homogeneous processor systems [20, 14, 24, 9, 2] and those for heterogeneous processor systems [11, 15, 21]. The remainder of this section will examine some of the relevant homogeneous and heterogeneous methods.

For homogeneous task systems, Sarkar and Hennessy [20] present a two-stage technique known as internalization, which first clusters the tasks into an arbitrary number of groups, and assigns these groups to the physical processors. Hwang et al. [14] present a heuristic called earliest task first (ETF), which uses a greedy selection to schedule tasks in homogeneous processor systems. Yang and Gerasoulis [24] present the DSC algorithm, which, on an unbounded number of processors, produces better results than either of the methods presented in [20] or [14]. Colin and Chrétienne [9] present a polynomial algorithm for optimally scheduling tasks on a homogeneous array of processors, provided task duplication is allowed. This method uses a critical path based algorithm. Atallah et al. [2] examine a method for balancing a background computation across a cluster of distributed, homogeneous workstations.

For heterogeneous processor systems, Kim and Browne [15] present a technique called linear clustering, which clusters tasks into chains of tasks, and maps the clusters onto the physical machines. El-Rewini and Lewis [11] present an algorithm known as the MH algorithm. This algorithm prioritizes the tasks based upon an estimate of the starting time, and assigns the tasks based upon those priorities. Both of these heterogeneous methods have limited application to the problem formulated here, since they assume that the individual processors perform uniformly for all code types (i.e. the performance of a task on each heterogeneous processor varies only by a scale factor). This assumption leads to sub-optimal results when applied to

a heterogeneous system composed of a diverse range of machine architectures. Sih and Lee [21] present a technique for mapping and scheduling in heterogeneous processor systems called Dynamic Level Scheduling, which assign a series of dynamically changing priorities to the tasks being scheduled. This method is very similar to the technique used by El-Rewini, although it uses a more robust assumption about the nature of the heterogeneous processors, avoiding the problems associated with the MH algorithm.

3.3 Mapping and Scheduling with CSTEM

Since many mapping and scheduling algorithms are optimized for specific types of problems, when searching for an algorithm, there are several characteristics that need to be matched to the problem. Obviously, with heterogeneous CSTEM, any algorithm must either support heterogeneity or be capable of being extended to support heterogeneity. Other important details include finding an algorithm suitable for both the type of network and the type of machine used. The heterogeneous environment used in this project consists of a variety of general purpose workstations and supercomputers, connected by either a shared medium, like ethernet, or a completely connected packet switched medium, like an ATM switch. In either case, the communication cost between nodes is, under most circumstances, independent of the physical locations of the sending and receiving processors. This type of network is also known as a uniform network.

An algorithm must also be suitable for the type of tasks to be allocated. The amount and grain size of the parallelism, combined with the number of precedence relations between individual tasks, plays a key role in the performance of any algorithm. When parallelizing an existing application, the tasks tend to have a large number of precedence constraints and a relatively low degree of parallelism between them. Precedence is the single most limiting factor to the overall performance of the algorithm; therefore, an algorithm that handles precedence well is essential.

3.4 The Levelized Min-Time Algorithm

The combination of precedence constraints and variable execution times complicates the assignment process. To simplify the problem, a two phase approach will be used. The first phase reduces the precedence constrained mapping and scheduling problem into a series of non-precedence constrained subproblems. The technique that will be used to accomplish this is known as level sorting [18, 7]. Once the problem has been divided using this technique, a much simpler algorithm can be used to solve the individual sub-problems. This algorithm is called the *Min-Time* algorithm. Together, these two stages form the *Levelized Min-Time* (LMT) algorithm.

Considering each subproblem to be completely independent does cause some inaccuracies to be introduced into the solution. Therefore, in order to improve the quality of the solution, some techniques will be given that will include some information from the other subproblems.

3.4.1 Level Sorting

The method used for the first phase is a technique for ordering the nodes based upon their precedence constraints, called level sorting. Level sorting has applications in several different areas, including logic simulation, fault simulation, and scheduling [18, 7].

The exact definition of the level sorting process can be given recursively: Given a graph $G = (V, E)$, level 0 contains all vertices v_j such that there is no vertex v_i with $e_{ij} \in E$. (i.e. v_j does not have any incident edges). Level k consists of all vertices v_j such that, for all edges $e_{ij} \in E$, every vertex v_i is in a level less than k, and at least one vertex is in level $k-1$. Figure 2 shows a sample DAG that has been level sorted.

The level sorting technique clusters nodes that are able to execute in parallel. By clustering tasks in this fashion, the tasks within each level have no precedence constraints between them. The second stage of the LMT algorithm will assign tasks level by level, using an assignment heuristic which does not use precedence information.

3.4.2 Min-Time

The second stage of the assignment process uses a heuristic called the *Min-Time* algorithm. The *Min-Time* algorithm is a greedy method that attempts to assign each task to the "best" processor—the processor on which the task runs the fastest.

The algorithm operates according to the following steps. First, the average execution time of each task, across all available machines, is calculated. Second, if the number of tasks is greater than the number of available processors, the number of tasks is reduced by merging the smallest tasks (based on the average time) until the number of tasks is equal to the number of processors. Third, the tasks are sorted in reverse order (largest first) by the average execution time. Finally,

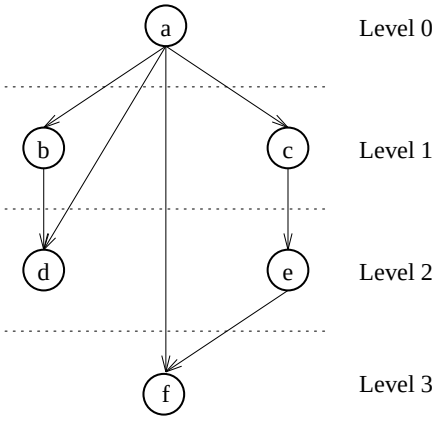

Figure 2: Level Sorting Example.

each task is assigned, in sorted order, to the processor on which it executes the fastest, with at most one task per processor. The sorting process increases the likelihood of large tasks being assigned to the fastest processors, while less demanding tasks are assigned to slower processors.

3.4.3 Final Considerations

Above, the assumption was made that the mapping and scheduling problem could be decomposed into a number of independent subproblems. In reality, these problems are not completely independent. The interactions between tasks in different levels can affect the overall cost of a mapping. This section will present some features that improve the quality of the solutions.

First, when making an assignment, there is a possibility that the *Min-Time* algorithm might have to choose between two or more identical machines. An effective way to resolve such a choice is to assign the task to the processor from which it receives most of its data, since the cost of sending a byte to another machine is significantly higher than the cost of communicating that byte locally. Therefore, by including the additional information regarding communication between the separate levels, the overall solution can be improved. The simplest way to add this information is to include the cost of communicating with tasks in previous levels into the overall execution cost of each task, increasing the likelihood that tasks which share a large amount of data are assigned to the same processor.

Another drawback with the original assumption results from the *Min-Time* algorithm failing to look at tasks in subsequent levels. The algorithm assumes that in order for a task on level $i+1$ to begin, every task on level i must be complete. In reality, the results produced by task i may not be used for several successive levels, giving that task more time to execute without creating a bottleneck. For example, consider the graph shown in Figure 3. If task C is large when compared to task B, under the LO algorithm task C would automatically get priority for assignment to the fastest processors. However, the results produced by task C are first used by task G in level 5. Therefore, Task C will not produce a performance bottleneck unless its execution time is greater than the sum of the execution times of tasks B, D, E, and F. There is a significant problem that occurs when trying to incorporate this information into an algorithm: the execution times of tasks D, E, and F are unknown when tasks B and C are being assigned. Furthermore, in a more realistic example, there would be more than one task in levels 2, 3, and 4, making the the time when task G will begin even more difficult to compute. To solve this problem, a method is needed to establish a reasonable estimate of the execution time of each subsequent level. The method used in this algorithm to estimate this time, is to use the average time of all of the tasks in the level as an estimate of the execution time of that level. Therefore, if the results from a task in level 1 are not used until level 5, the sum of the average times for the levels 2 through 4 can be used as an estimate of the extra time that the task has to complete execution.

By including the above factors, the formal definition of this algorithm is:

Procedure: LMT
begin
  Level sort tasks.
  For each level, in order, do:
  begin
    Assign tasks in level i using *Min-Time*.
  end
end

Procedure: Min-Time
begin
  For each task v_i do:
  begin
    Let avg_i = average value of x_{ij} for all possible j.
    Adjust average values based upon when results will be needed.

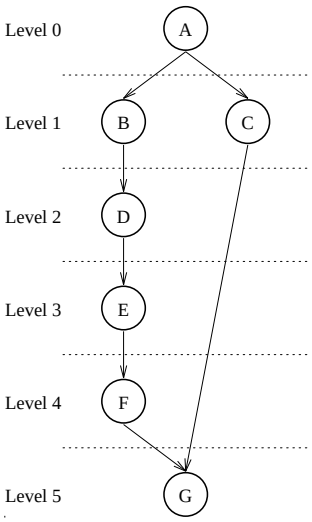

Figure 3: A Task Graph Showing the Drawback of Not Examining Subsequent Levels.

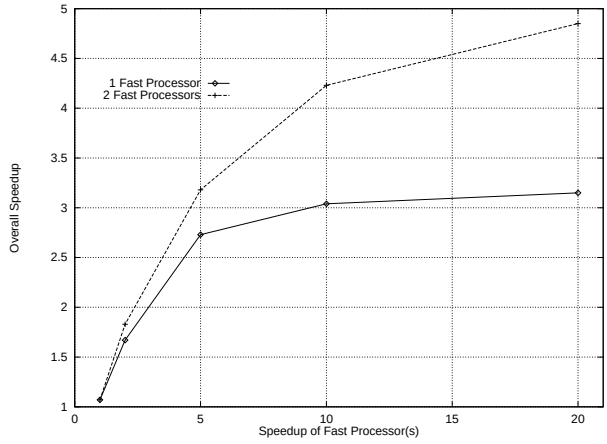

Figure 4: Speedup of Heterogeneous CSTEM with Different Processor Speeds.

end
Sort groups in reverse order by avg_i.
For each task v_i in sorted order do:
begin
Find j such that processor p_j does not have a task assigned to it and $x_{ij} + \sum_k \delta(c_{ki}, \mu(v_k), p_j)$ is minimal.
Assign task v_i to processor p_j.
end
end

4 Experimental Results

The intended execution environment for heterogeneous CSTEM is the Advanced Computational Concepts Laboratory (ACCL) at the NASA Lewis Research Center. This laboratory consists of a variety of high performance workstations and parallel machines connected by several different high performance networks. To evaluate potential heterogeneous clusters within this environment while limiting the overall programming effort, a series of simulations were performed. A custom event-based simulator, using timing information derived from actual measurements (shown in Figure 1) on a Sun Microsystems Sparc 10 workstation, was used to generate these results. The network timings were measured from standard PVM over a conventional ethernet based network. In addition to execution time and network speeds, the simulator also considers the effects of multiprogramming in its computations. The high communication overhead of PVM over a conventional network greatly affects the performance of CSTEM. The setup time for communication using PVM is exceptionally high, clearly creating a performance bottleneck. This problem can be reduced by using a more advanced network with an optimized version of PVM. Several of these networking technologies are available at ACCL, including ATM and other high speed switching technologies.

The results presented in this section are not intended to perform an accurate evaluation of the LMT algorithm, but to explore the potential performance of heterogeneous CSTEM. Given the amount of task parallelism present in the task graph, heterogeneous CSTEM can only effectively utilize about three machines in parallel. Therefore, Figure 4 shows the overall speedup obtained by applying the LMT algorithm to heterogeneous CSTEM, using various three machine heterogeneous clusters. The machines in these clusters could potentially be any type of machine, including workstations, parallel machines, or vector supercomputers. One trace shows the estimated speedup when one of the three machines is either 2, 5, 10, or 20 times faster than the baseline machine. The second trace shows the speedup when two of the machines are either 2, 5, 10, or 20 times faster than the reference machine. The network timings are that of a conventional ethernet based network.

As predicted, these results show that heterogeneous CSTEM is very communication bound, clearly indi-

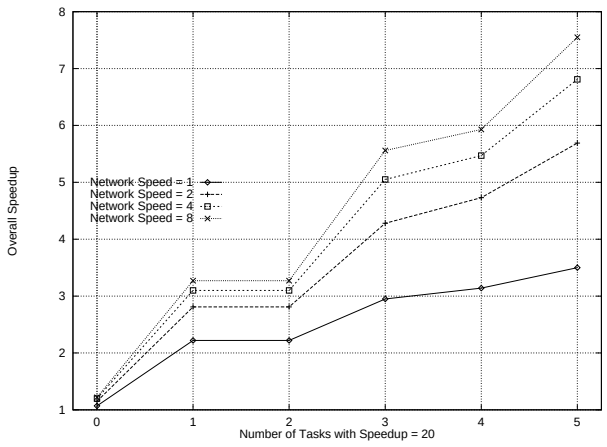

Figure 5: Speedups for Different Processor and Network Speeds

cating the need for a more advanced network architecture. It is also clear from the structure of the task graph shown in Figure 1 that there is a limited amount of parallelism present in the task graph. An effective solution to this problem is to exploit data parallelism within individual tasks.

In the above simulation, it is assumed that any task can execute on any processor. This assumption may not be valid in a real heterogeneous environment, since it may not be worth the programming effort to port non-critical tasks to every available architecture. Therefore, to demonstrate a more realistic example, Figure 5 shows the overall speedup obtained from increasing the speed of the five most computationally intensive tasks (M, F, E, K, and N in Figure 1) by a factor of 20, as well as the effects of increasing the speed of the network by a factor of 2, 4, and 8. These results show that, given adequate network resources, heterogeneous computing has the potential to provide a significant speedup, while limiting the programming effort to the most computation intensive tasks.

5 Conclusions

From the results presented above, heterogeneous computing has the potential to significantly increase the performance of existing applications, provided adequate network resources are available. Fortunately, high performance networking systems, like ATM, are becoming the standard method for connecting high performance machines, making heterogeneous computing a practical method for increasing the performance of an existing application. Furthermore, heterogeneous computing offers a number of advantages over other techniques in its ability to take advantage of different architectural features, and, with suitable programming tools, to limit the overall programming effort.

An effective mapping and scheduling algorithm is an essential element of heterogeneous computing, especially one which can take advantage of the available architectural features of a given heterogeneous processing system. Although a more complete analysis needs to be performed, the LMT algorithm should be a useful method for obtaining good solutions to the mapping and scheduling problem for heterogeneous processor systems.

As a continuation of this project, we are developing a set of heterogeneous programming tools, with the goal of simplifying the process of applying heterogeneous computing to existing applications. There will be four components to this tool set: a tool to assist in dividing code into a set of tasks, a tool to apply a variety of mapping and scheduling algorithms, a tool to evaluate the quality of a given mapping, and a tool to provide run-time performance monitoring and fault tolerant execution. Once this tool is in place, it will be possible to perform a more complete analysis of the LMT algorithm, comparing it to other heterogeneous mapping and scheduling algorithms.

References

[1] J. B. Andrews and C. D. Polychronopoulos, "An analytical approach to performance/cost modeling of parallel computers," *J. Parallel Distributed Computing*, vol. 12, pp. 345–356, 1991.

[2] M. J. Atallah, C. Lock, D. C. Marinescu, H. J. Seigel, and T. L. Casavant, "Co-scheduling compute-intensive tasks on a network of workstations: Models and algorithms," in *The 1991 IEEE Inter. Conf. Distributed Computing Systems.*, pp. 344–352, July 1991.

[3] A. Beguelin, J. Dongarra, A. Geist, and V. Sunderam, "Visualization and debugging in a heterogeneous environment," *IEEE Computer*, vol. 26, pp. 88–95, June 1993.

[4] A. Beguelin, J. Dongarra, G. A. Geist, R. Mancheck, K. Moore, R. Wade, J. Plank, and V. Sunderam, *HeNCE Users' Manual*, Dec. 1992.

[5] N. S. Bowen, C. N. Nikolaou, and A. Ghafoor, "On the assignment problem of arbitrary process systems to heterogeneous distributed computer systems," *IEEE Trans. Computers*, vol. 41, pp. 257–273, Mar. 1992.

[6] V. Chaudhary and J. K. Aggarwal, "A generalized scheme for mapping parallel algorithms," *IEEE Trans. Parallel and Distributed Systems*, vol. 4, pp. 328–346, Mar. 1993.

[7] C. L. Chen, C. S. G. Lee, and E. S. H. Hou, "Efficient scheduling algorithms for robot inverse dynamic computation on a multiprocessor system," *IEEE Trans. Systems, Man, Cybernetics*, vol. 18, pp. 729–743, Sept. 1988.

[8] S. Chen, M. M. Eshaghian, A. Khokhar, and M. E. Shaaban, "A selection theory and methodology for heterogeneous supercomputing," in *Proc. of the 1993 Workshop on Heterogeneous Processing*, pp. 15–22, The IEEE Computer Society Press, 1993.

[9] J. Y. Colin and P. Chrétienne, "C.P.M. scheduling with small communication delays and task duplication," *Operational Research*, vol. 39, no. 4, pp. 680–684, 1991.

[10] K. Efe, "Heuristic models of task assignment scheduling in distributed systems," *IEEE Computer*, vol. 15, pp. 50–56, June 1982.

[11] H. El-Rewini and T. G. Lewis, "Scheduling parallel program tasks onto arbitrary target machines," *J. Parallel Distributed Computing*, vol. 9, pp. 138–153, 1990.

[12] R. F. Freund and H. J. Siegel, "Heterogeneous processing," *IEEE Computer*, vol. 26, pp. 13–17, June 1993.

[13] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Mancheck, and V. Sunderam, *PVM 3 User's Guide and Reference Manual.* Oak Ridge National Lab., May 1993.

[14] J.-J. Hwang, Y.-C. Chow, F. D. Anger, and C.-Y. Lee, "Scheduling precedence graphs in systems with interprocessor communication times.," *SIAM J. Computing*, vol. 18, pp. 244–257, Apr. 1989.

[15] S. J. Kim and J. C. Browne, "A general approach to mapping parallel computations upon multiprocessor architectures.," in *The 1988 Inter. Conf. on Parallel Processing*, vol. 3, pp. 1–8, 1988.

[16] V. M. Lo, "Heuristic algorithms for task assignment in distributed systems," *IEEE Trans. Computers*, vol. 37, pp. 1384–1397, Nov. 1988.

[17] R. McKnight, "Coupled disciplinary analysis for aircraft gas turbine engines," in *The 1992 Interdisciplinary System Simulation and Design Workshop*, pp. 7–21, The Ohio Aerospace Institute, 1992.

[18] R. R. Muntz and E. G. Coffman, "Optimal preemptive scheduling on two-processor systems," *IEEE Trans. Computers*, vol. C-18, pp. 1014–1020, Nov. 1969.

[19] M. J. Quinn, *Parallel Computing: Theory and Practice.* McGraw-Hill Book Company, 1993.

[20] V. Sarkar and J. Hennessy, "Compile-time partitioning and scheduling of parallel programs," *ACM SIGPLAN*, vol. 21, pp. 17–26, July 1986.

[21] G. C. Sih and E. A. Lee, "A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures," *IEEE Trans. Parallel and Distributed Systems*, vol. 4, pp. 175–187, Feb. 1993.

[22] H. S. Stone, "Multiprocessor scheduling with the aid of network flow algorithms," *IEEE Trans. Software Engineering*, vol. SE-3, pp. 85–93, Jan. 1977.

[23] L. Tao, B. Narahari, and Y. C. Zhao, "Heuristics for mapping parallel computations to heterogeneous parallel architectures," in *Proc. of the 1993 Workshop on Heterogeneous Processing*, pp. 36–41, The IEEE Computer Society Press, 1993.

[24] T. Yang and A. Gerasoulis, "DSC: Scheduling tasks on an unbounded number of processors," *IEEE Trans. Parallel and Distributed Systems*, vol. 5, pp. 951–967, Sept. 1994.