Computing with Heterogeneous Parallel Machines: Advantages and Challenges

Howard Jay Siegel, Lee Wang, Vwani P. Roychowdhury, and Min Tan

Parallel Processing Laboratory, School of Electrical and Computer Engineering
Purdue University, West Lafayette, IN 47907-1285 USA

Abstract

This invited keynote paper discusses the advantages of computing with heterogeneous parallel machines, and examines the research challenges for automating the use of such systems. One type of heterogeneous computing system is a mixed-mode machine, where a single machine can operate in different modes of parallelism. Another is a mixed-machine system, where a suite of different kinds of parallel machines are interconnected by high-speed links. To exploit such systems, a task must be decomposed into subtasks, where each subtask is computationally homogeneous. The subtasks are then assigned to and executed with the machines (or modes) that will result in a minimal overall execution time. Typically, users must specify this decomposition and assignment. One long-term pursuit in heterogeneous computing is to do this automatically. An overview of a conceptual model of what this involves is given. As an example of the research in this area, a genetic-algorithm-based approach to the subtask assignment and scheduling problem is explored. Open problems in heterogeneous computing are described.

1: Introduction

Existing supercomputers generally achieve only a fraction of their peak performance on certain portions of some application tasks. This is because different subtasks of an application can have very different computational requirements that result in different needs for machine capabilities. A single machine architecture cannot satisfy all the computational requirements in certain applications equally well. Thus, the construction of a heterogeneous computing environment is more appropriate.

A heterogeneous computing (HC) system provides a variety of architectural capabilities, orchestrated to perform an application whose subtasks have diverse execution requirements [14]. Two types of HC systems are mixed-mode machines and mixed-machine systems. A mixed–mode machine is a single parallel machine that is capable of operating in either the SIMD or MIMD mode of parallelism and can dynamically switch between modes at instruction-level granularity with generally negligible overhead [17]. Thus, a mixed-mode machine is temporally heterogeneous, in that it can operate in different modes at different times. There are various tradeoffs between the SIMD and MIMD modes of parallelism [14], and mixed-mode machines can exploit these by matching each subtask with the mode that results in the best overall performance. A number of prototype mixed-mode machines have been built, including EXECUBE [10], MeshSP [7], OPSILA [2], PASM [15], TRAC [11], and Triton [12]. PASM can be dynamically partitioned into independent communicating mixed-mode submachines, so it is spatially and temporally heterogeneous (see [15] for more information).

A mixed–machine system is a heterogeneous suite of different types of independent machines interconnected by a high-speed network. The goal is to match each subtask to the machine that results in the lowest overall task execution time. To exploit HC systems, a task must be decomposed into subtasks, where each subtask is computationally homogeneous, and different subtasks may have different machine architectural requirements. Typically, users must specify this decomposition and assignment. One long-term pursuit in HC is to do this automatically.

Unlike mixed-mode machines, switching execution among machines in a mixed-machine system can require measurable overhead because of data transfers among machines. Thus, the mixed-machine systems considered are assumed to have high-speed connections among machines that make decomposition at the subtask level feasible.

Fig. 1 shows a hypothetical example of an application program whose various subtasks are best suited for execution on different machine architectures. Executing the whole program on a SIMD machine only gives approximately five times the performance achieved by a baseline serial machine. Only the SIMD portion of the

Supported in part by Rome Laboratory under contract number F30602-94-C-0022 and by NRaD under subcontract number 20-950001-70.

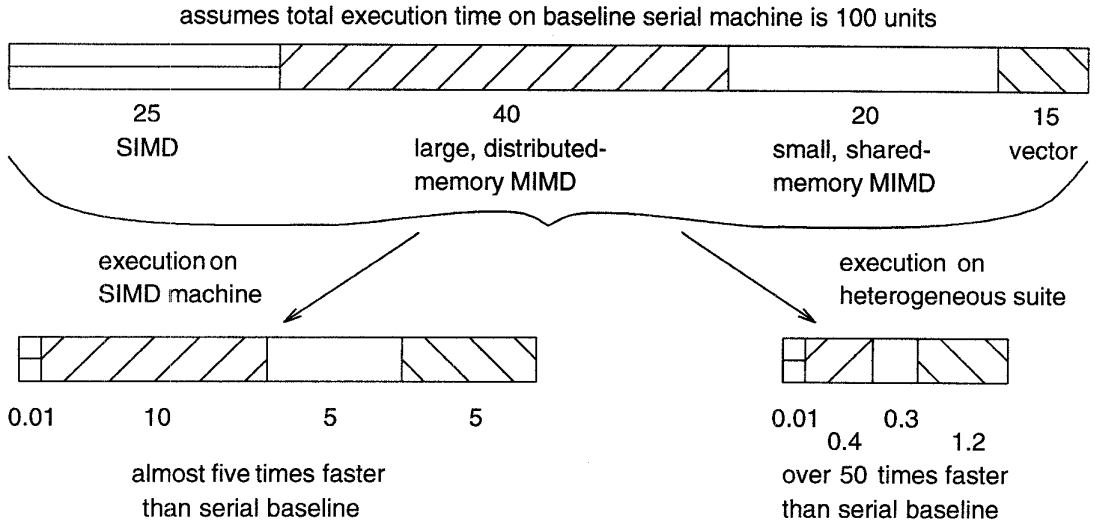


Fig. 1: A hypothetical example for showing the advantage of using HC, where the execution time for the heterogeneous suite includes inter-machine communications. Execution times assume the total time on the baseline serial system is 100 units.

program can be executed significantly faster because of the mismatch between each subtask's unique computational requirement and the SIMD architecture. Alternatively, the use of four different machines, each matched to the computational requirements of the subtask for which it is used, can result in an execution 50 times as fast as the baseline serial machine. Examples of existing HC systems include the simulator for mixing in turbulent convection [9], the interactive rendering of multiple earth science data sets on the CASA testbed [18], the computation of the radiation treatment planning for cancer patients on VISTAnet [13], and SmartNet [3].

A conceptual model for automatic HC is introduced in Section 2. As an example of current HC research, Section 3 presents a genetic-algorithm-based approach for matching subtasks to machines and scheduling subtasks in a mixed-machine HC environment. Open problems in the field of HC are discussed in Section 4.

2: A Conceptual Model for HC

A conceptual model for automatic task decomposition, matching subtasks to machines, and scheduling subtasks in a mixed-machine HC environment is shown in Fig. 2. It builds on the one presented in [4] and is referred to as a "conceptual" model because no complete automatic implementation currently exists. A more detailed description can be found in [14].

In stage 1, using information about the expected types of application tasks and about the machines in the HC suite, a set of parameters is generated that is relevant to both the computational requirements of the applications and the machine capabilities of the HC system. For each parameter, categories for computational characteristics and categories for machine architecture features are derived. The total number of parameters impacts the complexity of the automation process. The chosen parameters evolve dynamically when new types of applications or new types of machines are added.

In stage 2, task profiling decomposes the application task into subtasks, each of which is computationally homogeneous. The computational requirements for each subtask are then quantified. Both the code and data upon which the specified HC system will operate must be profiled. Analytical benchmarking is used in stage 2 to quantify how effectively each of the available machines in the suite performs on each of the types of computations being considered.

One of the functions of stage 3 is to be able to use the information from stage 2 to derive the estimated execution time of each subtask on each machine in the HC suite and the associated inter-machine communication overhead. Then, these static results and the dynamic information about the current loading and "status" of the machines and inter-machine network (i.e., machine/network faults and expected subtask/transfer completion times) are used to generate an assignment of the subtasks to machines in the HC system and an execution schedule based on certain cost metrics (e.g, minimizing the overall task execution time).

Stage 4 is the execution of the given applications on the HC suite. Because the loading/status of the

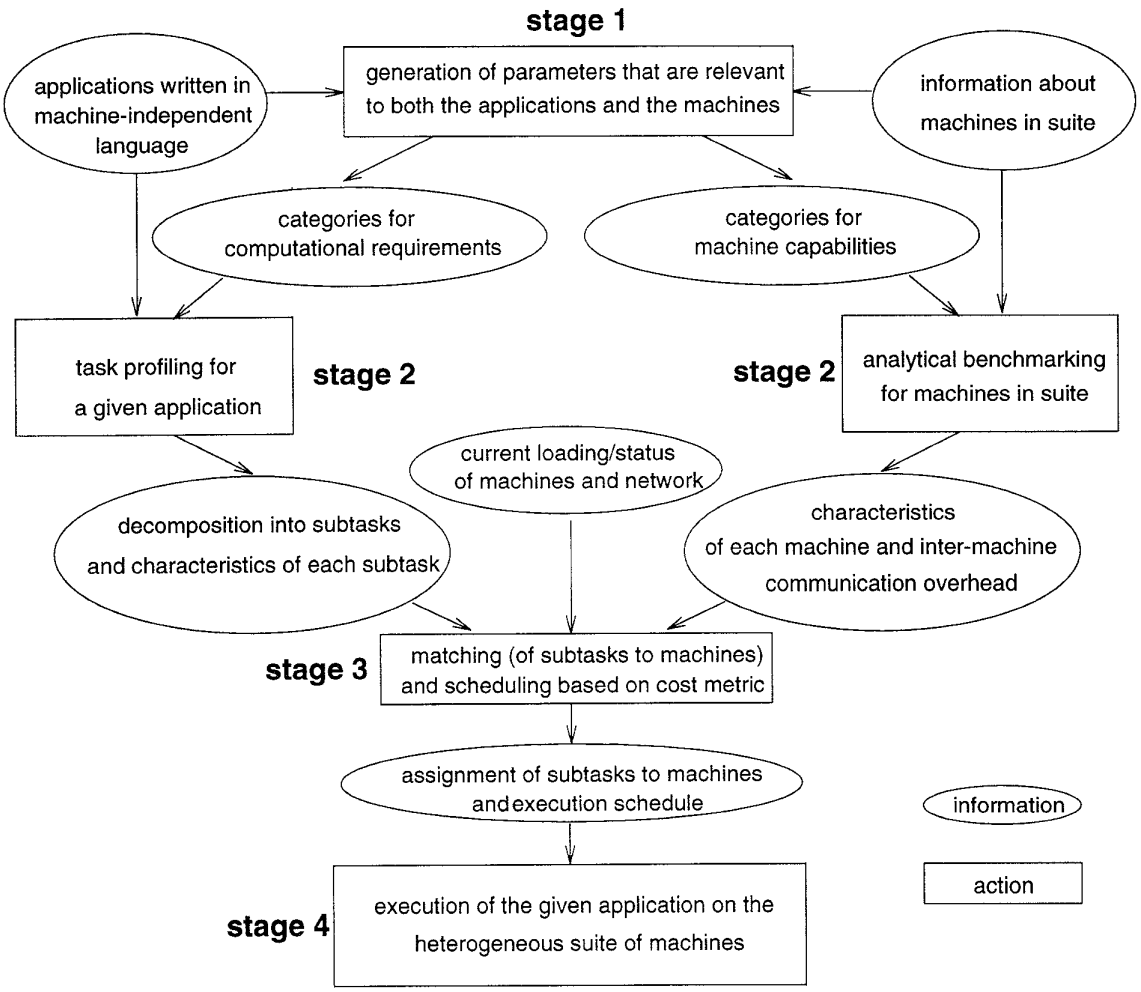


Fig. 2: Conceptual model of the automatic assignment of subtasks to machines (from [16]).

machines/network may change, it may be necessary to reselect machines for certain subtasks by reactivating stage 3.

Automatic HC is a relatively new field. Frameworks for task profiling, analytical benchmarking, and matching and scheduling have been proposed, however, further research is needed to make this conceptual model a reality [14].

3: A Genetic-Algorithm-Based Approach for Task Matching and Scheduling

As an example of current HC research, this section summarizes a genetic-algorithm-based approach for task matching and scheduling in HC environments [20]. First, representations of the HC system and the application are described. Then the structure of the genetic algorithm used is explained, followed by the results achieved.

An application task is decomposed into a set of subtasks Γ of size $|\Gamma|$. Let γ_i be the i-th subtask. An HC environment consists of a set of machines Π of size $|\Pi|$. Let π_j be the j-th machine. The estimated expected execution time of subtask γ_i on machine π_j is ET_{ij}. The global data items (*gdis*), i.e., data items that need to be transferred between subtasks, form a set G of size $|G|$. Let gdi_k be the k-th global data item.

The following assumptions about the applications and HC environment are made. The data dependencies among the subtasks are known and are represented by a directed acyclic graph. For each global data item, there is a single subtask that produces it (producer) and there are some subtasks that need this data item (consumers). Hence, the task is represented by a single producer

directed acyclic graph (SPDAG). Each edge goes from a producer to a consumer and is labeled by the global data item that is transferred over it. This application has exclusive use of the HC environment, and the genetic-algorithm-based matcher/scheduler controls the HC machine suite. Subtask execution is non-preemptive. All input data items of a subtask must be received before its execution can begin, and none of its output data items is available until the execution of this subtask is finished.

Genetic algorithms (GAs) are a promising heuristic approach to optimization problems that are intractable [5, 6]. The first step is to encode some of the possible solutions (chromosomes), the set of which is referred to as a population. Let *mat* be the matching string, which is a vector of length $|\Gamma|$, where $mat(i) = \pi_j$ (i.e., subtask γ_i is assigned to machine π_j). The scheduling string is a topological sort of the SPDAG. For subtasks assigned to different machines, execution scheduling is determined by the data dependencies among them. Thus, different scheduling strings may represent the same overall schedule. Define *ss* to be the scheduling string, which is a vector of length $|\Gamma|$, where $ss(k) = \gamma_i$, $0 \le i, k < |\Gamma|$ (i.e., subtask γ_i is the k-th subtask in the scheduling string). Because it is a topological sort, if $ss(k)$ is a consumer of a *gdi* produced by $ss(j)$, then $j < k$. The scheduling string is used only to determine execution ordering of the subtasks assigned to the same machine. In this approach, each chromosome is a two-tuple <*mat*, *ss*>.

In the initial population generation step, a predefined number of chromosomes are created. A new matching string is obtained by assigning each subtask to a machine randomly. The SPDAG is first topologically sorted to form a basis scheduling string. Then, for each chromosome to be generated, this basis string is mutated multiple times using the scheduling string mutation operator (defined below) to generate the *ss* vector. The solution from a non-evolutionary baseline (BL) heuristic is also included in the initial population. It is guaranteed that the chromosomes in the initial population are distinct from each other.

After the initial population is determined, the genetic algorithm iterates until a predefined termination criteria is met. Each iteration consists of the selection, the crossover, the mutation, and the evaluation steps.

Each chromosome is associated with a fitness value, which is the completion time of the solution (i.e., matching and scheduling) represented by this chromosome. Overlapping among all of the computations and communications performed is limited only by inter-subtask data dependencies and machine/network availability. The fitness values are determined in the evaluation step (discussed later). In the selection step after the initial population generation, the chromosomes are ordered (ranked) by their fitness values. Then a rank-based roulette wheel selection scheme is used to implement proportionate selection [6]. The population size is kept constant and a chromosome representing a better solution has a higher probability of having one or more copies in the next generation. This GA-based approach also incorporates elitism, i.e., the best solution found so far is always maintained in the population.

The selection step is followed by the crossover step, where some chromosomes are paired and corresponding components of the paired chromosomes are exchanged. The crossover operator for the scheduling strings randomly chooses some pairs of the scheduling strings. For each pair, it randomly generates a cutoff point, which cuts the scheduling strings of the pair into top and bottom parts. Then, the subtasks in each bottom part are reordered. The new ordering of the subtasks in the bottom part of one string is the relative positions of these subtasks in the other original scheduling string, thus guaranteeing that the newly generated scheduling strings are valid schedules.

The crossover operator for the matching strings randomly chooses some pairs of the matching strings. For each pair, it randomly generates a cutoff point to cut both matching strings of the pair into two parts. Then the machine assignments of the bottom parts are exchanged.

The next step is the mutation step. The scheduling string mutation operator randomly chooses some scheduling strings. Then for each chosen scheduling string, it randomly selects a victim subtask. The free range of the victim subtask is the set of the positions in the scheduling string at which this victim subtask can be placed and still have a valid topological sort of the SPDAG. After a victim subtask is chosen, it is moved randomly to another position in the scheduling string within its free range. The matching string mutation operator randomly chooses some matching strings. On each chosen matching string, it randomly selects a subtask/machine pair. Then the machine assignment for the selected pair is changed randomly to another machine.

The last step of an evolution iteration is the evaluation of the fitness value of each chromosome. The communication characteristics of the given HC system are needed only in the evaluation step. To demonstrate the evaluation process, a communication subsystem, which is modeled after a HiPPI LAN with a central crossbar switch [19], is assumed. If a subtask needs a *gdi* that is produced or consumed earlier by a different subtask on the same machine, the communication cost is zero. (Currently, communications are source and

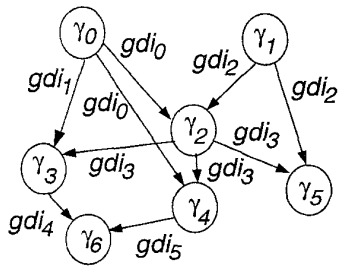
destination machine independent; machine dependence is being added.) For each chromosome, the evaluation procedure evaluates the subtasks in the order they appear in the scheduling string.

GA simulation studies were conducted using the following parameters. The probabilities for scheduling and matching string crossovers and scheduling and matching string mutations were 0.4, 0.4, 0.1, 0.1, respectively. This set of numbers was selected by experimentation. A GA execution stops if either the number of iterations reached 1000, the population converged (all the chromosomes had the same fitness value), or the best solution found was not improved after 150 iterations.

The results from a small-scale test illustrate the search process. This test had $|\Gamma| = 7$, $|\Pi| = 3$, and $|G| = 6$. Fig. 3 shows this test, where the SPDAG and the table entry values are generated randomly. The total number of possible different matching strings was 3^7 = 2187 and scheduling string was 16 (the number of possible valid topological sorts). The population size for this test was chosen to be 50. Fig. 4 depicts the evolution process for this test. The *ss* axis is the scheduling string axis and the *mat* axis is the matching string axis. If there is a chromosome at a point (*mat*, *ss*), then there is a vertical pole at (*mat*, *ss*). The greater the height of the pole, the better a chromosome (solution) is. Multiple identical chromosomes at the same point are not differentiated.

The GA run converged at iteration 43. Convergence does not necessarily mean that all chromosomes become identical; this GA-based approach could find multiple best solutions that have the same completion time. For each of the small-scale tests that were conducted (up to ten subtasks, three machines, and seven global data items), the GA-based approach found solution(s) that had the same completion time as that of the optimal solution found by exhaustive searches.

Larger tests with up to 100 subtasks and 20 machines were conducted. Each of them had its number of global data items in the range $(2/3)|\Gamma| < |G| \leq |\Gamma|$. The population size for these larger tests was chosen to be 200. Fig. 5 shows the performance comparisons among the non-evolutionary baseline (BL) heuristic, the levelized min-time (LMT) heuristic proposed in [8], and the GA-based approach. In the figure, the horizontal axes are the number of subtasks in log scale. The vertical axes are the relative solution quality, which is defined as the task completion time of the solution (matching and scheduling) found by the LMT heuristic divided by that found by the approach being plotted. Each point in the figure is the average of 50 independent tests.



(a)

	γ_0	γ_1	γ_2	γ_3	γ_4	γ_5	γ_6
π_0	872	251	542	40	742	970	457
π_1	898	624	786	737	247	749	451
π_2	708	778	23	258	535	776	15

(b)

gdi_0	347
gdi_1	883
gdi_2	44
gdi_3	589
gdi_4	275
gdi_5	709

(c)

Fig. 3: A small-scale test: (a) the SPDAG, (b) the estimated execution times, and (c) the transfer times of the global data items.

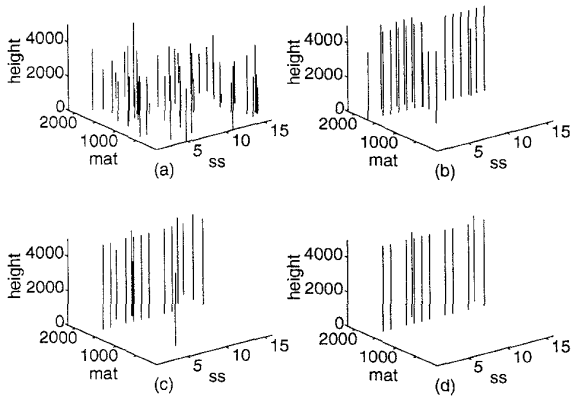


Fig. 4: Evolution of a GA run for the test in Fig. 3 at the (a) 0th, (b) 20th, (c) 40th, and (d) 43rd iterations.

Research on this approach is continuing. More experiments varying the GA parameters are being conducted.

4: Open Problems

There are a great many open problems that need to be solved before HC can be made available to application programmers in a transparent way (summarized here from [14, 16, 17]). Many need to be addressed just to facilitate near-optimal practical programmer-specified use of HC systems.

In the hardware category of mixed-mode computing,

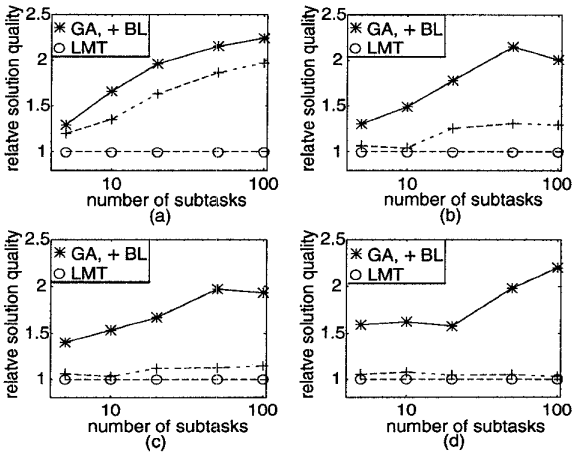


Fig. 5: Performance comparisons for larger tests in (a) two-, (b) five-, (c) ten-, and (d) 20-machine suites.

a variety of new architectural designs should be explored, based on current technology and expectations for future technology. These should include both designs based on incorporating commodity processors and designs using custom fabricated chips targeted for mixed-mode computing. Variations on the model of mixed-mode computing should also be examined. One such variation is a mixed–component HC system, where each separate component of a single machine represents one mode of parallelism, and two or more distinct modes can be employed concurrently in the machine (e.g., the SIMD/MIMD Image Understanding Architecture [21]).

In the programming language and compiling arena of mixed-mode computing, work is needed for supporting both explicitly and implicitly specified mixed-mode parallelism. For explicit languages, notations for supporting the specification of the interleaving of SIMD and MIMD operations need to be developed. Also, the mode switches should occur with minimal software overhead. For implicit languages, the compiler must automatically decompose the program into appropriate segments, assign each segment to the mode of parallelism that will result in the best overall program execution time, and then generate effective code.

In the area of application studies, it is critical to demonstrate important problem domains for which mixed-mode computing can result in a significant improvement in performance relative to single-mode parallelism. This can be done through a combination of algorithm complexity analyses and experimentation on current and future mixed-mode prototype machines.

Implementation of the automatic HC programming environment envisioned in Section 2 will require a great deal of research for devising practical and theoretically sound methodologies for each component of every stage. A general question that is particularly applicable to stages 1 and 2 of the conceptual model is: "What information should (must) the user provide and what information should (can) be determined automatically?"

To program an HC system, it would be best to have one or more machine-independent programming languages [22] that allow the user to augment the code with compiler directives. The language and directives should be designed to facilitate (a) the compilation of the program into efficient code for the machines in the suite, (b) the task decomposition, (c) the determination of computational requirements of each subtask, and (d) the use of machine-dependent subroutine libraries.

There is also a need for debugging and performance tuning tools that can be used across an HC suite of machines. This involves research in the areas of distributed programming environments and visualization tools.

Ideally, information about the current loading and status of the machines in the HC suite and the network used should be incorporated into the matching and scheduling decisions. Many questions arise here: what information to include in the status (e.g., faulty or not, pending subtasks), how to measure current loading, how to effectively incorporate current loading and status information into matching and scheduling decisions, how to communicate and structure this information in other machines, how often to update this information, and how to estimate subtask/transfer completion time.

There is much ongoing research in the area of inter-machine data transport. This research includes the hardware support required, the software protocols required, designing the network topology, computing the minimum-time path between two machines, and devising rerouting schemes in case of faults or heavy loads. Related to this is the data reformatting problem, involving issues such as data type storage formats and sizes, byte ordering within data types, and machines' network-interface buffer sizes.

Another area of research is dynamic task migration between different parallel machines at execution time. Current research in this area involves how to move an executing task between different machines and determining how and when to use dynamic task migration for load rebalancing or fault tolerance [1].

Thus, although the uses of existing HC systems demonstrate the significant benefit of HC, the amount of effort currently required to implement an application on an HC system can be substantial. Future research on the above open problems will improve this situation and allow HC to realize its inherent potential.

Acknowledgments: The authors thank John K. Antonio, Richard C. Metzger, and Janet M. Siegel for their valuable comments.

References

[1] J. B. Armstrong and H. J. Siegel, "Dynamic task migration from SIMD to SPMD virtual machines," *1st IEEE Int'l Conf. Engineering of Complex Computer Systems*, Nov. 1995, pp. 326-333.

[2] P. Duclos, F. Boeri, M. Auguin, and G. Giraudon, "Image processing on a SIMD/SPMD architecture: OPSILA," *9th Int'l Conf. Pattern Recognition,* Nov. 1988, pp. 430-433.

[3] R. Freund, T. Kidd, and D. Hensgen, "SmartNet: A scheduling framework for meta-computing," *2nd Int'l Symp. Parallel Architectures, Algorithms, and Networks (ISPAN '96),* June 1996, in this proceedings.

[4] R. F. Freund and H. J. Siegel, "Heterogeneous processing," *IEEE Computer*, Vol. 26, June 1993, pp. 13-17.

[5] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning,* Addison-Wesley, Reading, MA, 1989.

[6] J. H. Holland, *Adaptation in Natural and Artificial Systems,* Univ. of Michigan Press, Ann Arbor, MI, 1975.

[7] Integrated Computing Engines, Inc., *The MeshSP,* technical report, ICE, Inc., Waltham, MA, July 1995.

[8] M. A. Iverson, F. Ozguner, and G. J. Follen, "Parallelizing existing applications in a distributed heterogeneous environment," *4th Heterogeneous Computing Workshop (HCW '95),* Apr. 1995, pp. 93-100.

[9] A. E. Klietz, A. V. Malevsky, and K. Chin-Purcell, "A case study in metacomputing: distributed simulations of mixing in turbulent convection," *2nd Workshop on Heterogeneous Processing*, Apr. 1993, pp. 101-106.

[10] P. M. Kogge, "EXECUBE - a new architecture for scalable MPPs," *1994 Int'l Conf. Parallel Processing*, Vol. I, Aug. 1994, pp. 77-84.

[11] G. Lipovski and M. Malek, *Parallel Computing: Theory and Comparisons,* John Wiley & Sons, New York, NY, 1987.

[12] M. Philippsen, T. Warschko, W. Tichy, and C. Herter, "Project Triton: towards improved programmability of parallel machines," *26th Hawaii Int'l Conf. System Sciences,* Jan. 1993, pp. 192-201.

[13] J. Rosenman and T. Cullip, "High-performance computing in radiation cancer treatment," *CRC Critical Reviews in Biomedical Engineering*, Vol. 20, 1992, pp. 391-402.

[14] H. J. Siegel, J. K. Antonio, R. C. Metzger, M. Tan, and Y. A. Li, "Heterogeneous computing," in *Parallel and Distributed Computing Handbook*, A. Y. Zomaya, ed., McGraw-Hill, New York, NY, 1996, pp. 725-761.

[15] H. J. Siegel, T. D. Braun, H. G. Dietz, M. B. Kulaczewski, M. Maheswaran, P. H. Pero, J. M. Siegel, J. J. E. So, M. Tan, M. D. Theys, and L. Wang, "The PASM project: A study of reconfigurable parallel computing," *2nd Int'l Symp. Parallel Architectures, Algorithms, and Networks (ISPAN '96),* June 1996, in this proceedings.

[16] H. J. Siegel, H. G. Dietz, and J. K. Antonio, "Software support for heterogeneous computing," in *The Computer Science and Engineering Handbook*, A. B. Tucker, Jr., ed., CRC Press, Boca Raton, FL, 1996, to appear.

[17] H. J. Siegel, M. Maheswaran, D. W. Watson, J. K. Antonio, and M. J. Atallah, "Mixed-mode system heterogeneous computing," in *Heterogeneous Computing*, M. M. Eshaghian, ed., Artech House, Norwood, MA, 1996, to appear.

[18] "Special report: Gigabit network testbeds," *IEEE Computer*, Vol. 23, Sep. 1990, pp. 77-80.

[19] D. Tolmie and J. Renwick, "HiPPI: Simplicity yields success," *IEEE Network,* Vol. 7, Jan. 1993, pp. 28-32.

[20] L. Wang, H. J. Siegel, and V. P. Roychowdhury, "A genetic-algorithm-based approach for task matching and scheduling in heterogeneous computing environments," *5th Heterogeneous Computing Workshop (HCW '96)*, Apr. 1996, to appear.

[21] C. C. Weems, E. M. Riseman, and A. R. Hanson, "Image understanding architecture: Exploiting potential parallelism in machine vision," *IEEE Computer*, Vol. 25, Feb. 1992, pp. 65-68.

[22] C. C. Weems, G. E. Weaver, and S. G. Dropsho, "Linguistic support for heterogeneous parallel processing: a survey and an approach," *3rd Heterogeneous Computing Workshop (HCW '94)*, Apr. 1994, pp. 81-88.