Heterogeneous Computing: Goals, Methods, and Open Problems*

Tracy D. Braun[1], Howard Jay Siegel[2], and Anthony A. Maciejewski[2]

[1] NOEMIX, Inc.
1425 Russ Blvd. Ste. T-110
San Diego, CA 92101-4717 USA
tdbraun@noemix.com
http://members.home.net/brauntd

[2] Electrical and Computer Engineering Department
Colorado State University
Fort Collins, CO 80523-1373 USA
{hj, aam}@colostate.edu
http://www.engr.colostate.edu/{~hj, ~aam}

Abstract. This paper discusses the material to be presented by H. J. Siegel in his keynote talk. Distributed high-performance heterogeneous computing (HC) environments are composed of machines with varied computational capabilities interconnected by high-speed links. These environments are well suited to meet the computational demands of large, diverse groups of applications. One key factor in achieving the best performance possible from HC environments is the ability to assign effectively the applications to machines and schedule their execution. Several factors must be considered during this assignment. A conceptual model for the automatic decomposition of an application into tasks and assignment of tasks to machines is presented. An example of a static matching and scheduling approach for an HC environment is summarized. Some examples of current HC technology and open research problems are discussed.

1 Introduction

Existing high-performance computers sometimes achieve only a fraction of their peak performance capabilities on some tasks [14]. This is because different tasks can have very different computational requirements that result in the need for different machine capabilities. A single machine architecture may not satisfy all the computational requirements of different tasks equally well. Thus, the use of a heterogeneous computing environment is more appropriate.

* This research was supported in part by the DARPA/ITO Quorum Program under the NPS subcontract numbers N62271-97-M-0900, N62271-98-M-0217, and 62271-98-M-0448, and under the GSA subcontract number GS09K99BH0250. Some of the equipment used was donated by Intel and Microsoft.

This paper summarizes and extends some of the material in [6,26], and corresponds to the keynote presentation H. J. Siegel will give at the conference. He will give an overview of some research in the area of heterogeneous computing (HC), where a suite of different kinds of machines are interconnected by high-speed links. Such a system provides a variety of architectural capabilities, orchestrated to perform tasks with diverse execution requirements by exploiting the heterogeneity of the system [10,26,29]. An HC system may consist of a set of high-performance machines. A cluster composed of different types (or models or ages) of machines also constitutes an HC system. Alternatively, a cluster could be treated as a single machine in an HC suite. An HC system could also be part of a larger grid [12].

An application is assumed to be composed of one or more independent (i.e., non-communicating) tasks. It is also assumed that some tasks may be further decomposed into two or more communicating subtasks. The subtasks have data dependencies among them, but are able to be assigned to different machines for execution.

Consider Fig. 1, which shows a hypothetical example of an application program with various components that are best suited for execution on different machine architectures [14]. The example application in Fig. 1 consists of one task, decomposed into four consecutive subtasks. The application executes for 100 time units on a baseline workstation, where each subtask is best suited to the machine architecture and takes the amount of time indicated underneath it in the figure.

By performing the entire application on a cluster of workstations, the execution time of the large cluster-oriented subtask may decrease from 35 to 0.3 time units. The overall execution time improvement for the entire application is only by about a factor of two because the other subtasks may not be well suited for a cluster architecture.

Alternatively, the use of four different machine architectures, each matched to the computational requirements of the subtask to which it was assigned, can result in an execution 50 times as fast as the baseline workstation. This is because each subtask is executing on the high performance architecture for which it is best suited. The execution time shown below the bars for the HC suite in Fig. 1 include inter-machine communication overhead for each subtask to pass data to the next subtask. This inter-machine communication overhead is not needed in the single machine implementations of the task.

The construction of an HC suite with all four of these types of machines is, of course, more costly than just a single workstation or a single cluster of workstations. Thus, the steady state workload of applications must be sufficient to justify the system cost.

A key factor in achieving the best performance possible from HC environments is the ability to match (assign) the tasks and subtasks to the machines and schedule (order) the tasks and subtasks on each machine in an effective and efficient manner. The matching and scheduling of tasks and subtasks is defined as a mapping.

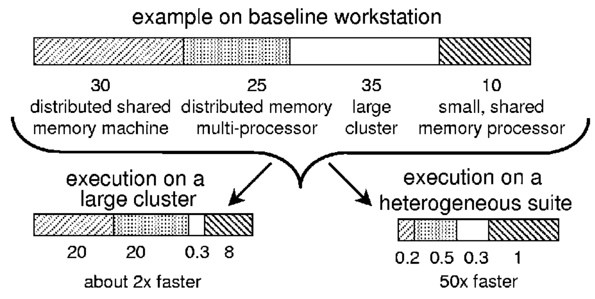
example on baseline workstation
30
distributed shared memory machine
25
distributed memory multi-processor
35
large cluster
10
small, shared memory processor
execution on a large cluster
20
20
0.3
8
about 2x faster
execution on a heterogeneous suite
0.2
0.5
0.3
1
50x faster

Fig. 1. Hypothetical example of the advantage of using a heterogeneous suite of machines, where the heterogeneous suite execution time includes inter-machine communication overhead (based on [14]). Not drawn to scale

Two different types of mapping are static and dynamic. Static mapping is performed when the tasks are mapped in an off-line planning phase, e.g., planning the schedule for tomorrow. Dynamic mapping is performed when the tasks are mapped in an on-line, real-time fashion, e.g., when tasks arrive at random intervals and are mapped as they arrive. In either case, the mapping problem has been shown, in general, to be NP-complete [7,11,18]. Thus, the development of heuristic techniques to find near-optimal mappings is an active area of research, e.g. [3,5,6,10,17,25,27,31].

A conceptual model for automatic HC is introduced in Sect. 2. As an example of current research in static matching and scheduling, Sect. 3 presents a greedy-based approach. Section 4 gives a brief sampling of some HC environments and applications. Open problems in the field of HC are discussed in Sect. 5.

This research is supported by the DARPA/ITO Quorum Program project called MSHN (Management System for Heterogeneous Networks) [16]. MSHN is a collaborative research effort among Colorado State University, Purdue University, the University of Southern California, NOEMIX, and the Naval Postgraduate School. One objective of MSHN is to design and evaluate mapping heuristics for different types of HC environments.

2 A Conceptual Model for HC

One of the long-term goals of HC research is to develop software environments that will automatically map and execute applications expressed in a machine-independent, high-level language. Developing such environments will facilitate the use of HC suites by (1) increasing software portability, because programmers need not be concerned with the constitution of the HC suite, and (2) increasing the possibility of deriving better mappings than users themselves derive with *ad hoc* methods. Thus, it will improve the performance of and encourage the use of HC in general.

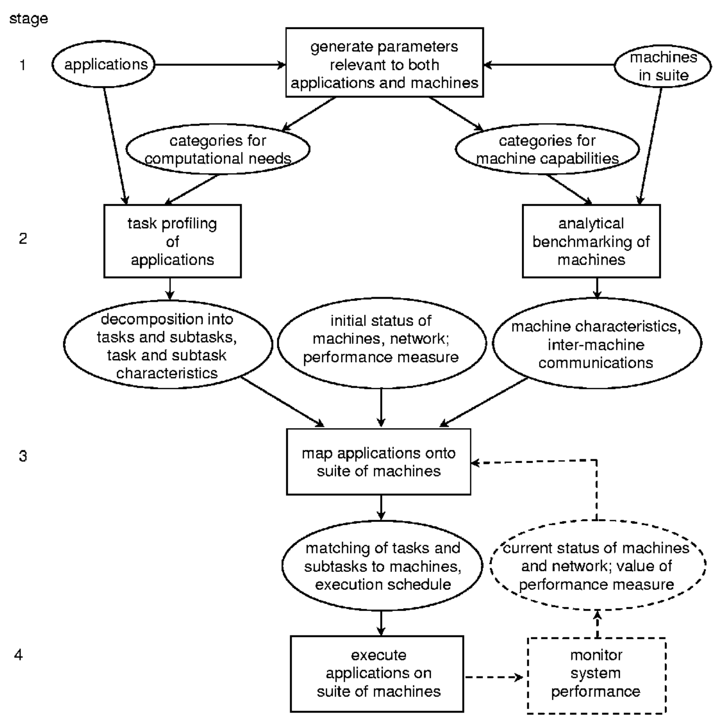
stage
1
2
3
4
applications
generate parameters relevant to both applications and machines
machines in suite
categories for computational needs
categories for machine capabilities
task profiling of applications
analytical benchmarking of machines
decomposition into tasks and subtasks, task and subtask characteristics
initial status of machines, network; performance measure
machine characteristics, inter-machine communications
map applications onto suite of machines
matching of tasks and subtasks to machines, execution schedule
current status of machines and network; value of performance measure
execute applications on suite of machines
monitor system performance

Fig. 2. Model for the support needed for automating the use of HC systems (based on [29]). Ovals indicate information and rectangles represent actions. The dashed lines represent the components needed to perform dynamic mapping

A conceptual model for such an environment using an HC suite of dedicated machines is described in Fig. 2. The conceptual model consists of four stages. It builds on the model presented in [29] and is referred to as a "conceptual" model because no complete automatic implementation currently exists.

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 example, if none of the expected applications include floating point operations, there is no need to characterize the floating point performance of each machine in the suite. For each parameter relevant to both the expected applications and the expected suite of machines, categories for computational characteristics and categories for machine architecture features are derived.

Stage 2 consists of two components, task profiling and analytical benchmarking. Task profiling decomposes the application into tasks (and possibly subtasks), each of which is computationally homogeneous. Different tasks (and subtasks) may have different computational needs. The computational requirements for each task (and subtask) are then quantified by profiling the code and data. Analytical benchmarking is used 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 task and subtask on each machine in the HC suite (as well as other quality of service (QoS) attributes, such as security level [23]), and the associated inter-machine communication overhead. Then, these results, along with the machine and inter-machine network initial loading and "status" (e.g., machine/network casualties) are used to generate a mapping of tasks and subtasks to machines based on certain performance measures [23] (e.g., minimizing the overall task execution time).

Stage 4 is the execution of the given application. In systems where the mapping is done statically in stage 3, estimated information about all of the tasks and subtasks to execute is known in advance. In dynamic mapping systems, general information about the applications to execute might be known *a priori* (e.g., from benchmarking in stage 2), but specific information (e.g., the exact set of applications or when an application will be submitted for execution) may not be known in advance. Thus, in dynamic mapping systems the task completion times and loading/status of the machines/network are monitored (shown by the dashed lines in Fig. 2). This information may be used to reinvoke the mapping of stage 3 to improve the machine assignment and execution schedule, or to alter the mapping based on changing user needs.

Automatic HC is a relatively new field. Frameworks for task profiling, analytical benchmarking, and mapping have been proposed, however, further research is needed to make this conceptual model a reality [26,29].

3 Static Mapping Heuristics

3.1 Introduction

As mentioned in Sect. 1, the heuristics to map tasks to machines can execute statically (off-line) or dynamically (on-line). There are several trade-offs between these two approaches. Static heuristics typically can use more time to determine a mapping because it is being done off-line, e.g., for production environments; but static heuristics must then use estimated values for parameters such as when a machine will be available. In contrast, dynamic heuristics operate on-line, therefore they must make scheduling decisions in real-time, but have feedback for actual parameter values instead of estimates for many system parameters.

As an example of current HC research on mapping statically, a greedy approach from [6] is summarized. Examples of other static mapping heuristics are given in [5,10,18,26,31]. Examples of dynamic mapping heuristics in HC environments can be found in [3,25,26].

Static mapping is utilized for many different purposes. Static mapping is used in large production environments to plan work to perform over a future interval, e.g., mapping tasks to execute the next day. Static mapping is also used for "what-if" predictive studies. For example, a system administrator might want to estimate the benefits of adding a new machine to the HC suite to justify purchasing it. Static mapping can also be used to do a post-mortem analysis of the performance of on-line dynamic mappers (where the static mapper would assume advance knowledge of all the applications to be executed).

The static mapping heuristics in this section are evaluated using simulated execution times for an HC environment. Because these are static heuristics, it is assumed that an estimate of the expected execution time for each task on each machine is known prior to execution and contained within an *ETC* (expected time to compute) matrix [1,2]. The assumption that these estimated expected execution times are known is commonly made when studying mapping heuristics for HC systems (e.g., [9,15,20,30]). Approaches for doing this estimation based on task profiling and analytical benchmarking of real systems are discussed in [22,26,29].

3.2 Problem Description

Assume the tasks being mapped to the HC environment have the following additional characteristics, increasing the complexity of the basic mapping problem: priorities, deadlines, multiple versions, and user-assigned preferences. Some tasks are also decomposed into subtasks with inter-subtask data dependencies. Each of these characteristics is described below.

In this study, each task $\underline{t_i}$ will have one of four possible weighted priorities, $\underline{p_i} \in \{1,\ 4,\ 16,\ 64\}$ (each value is equally likely to be assigned to a task in the simulation studies conducted). Values of $p_i = 64$ represent the most important or highest priority tasks.

This research assumes an oversubscribed system, i.e., there are not enough resources available to satisfy all the requirements of all the tasks. To model this, the tasks in the simulation studies conducted are assigned an arrival time and a deadline. Let the arrival time for task t_i be denoted $\underline{a_i}$, and let the deadline for task t_i be denoted $\underline{D_i}$.

Deadlines are assigned to each task as follows. First, for task t_i, the median execution time, $\underline{med_i}$, of the task on all machines in the HC suite is found. Next, each task t_i is randomly assigned a deadline factor, $\underline{\delta_i}$, where $\delta_i \in \{1, 2, 3, 4\}$ (each value has an equal likelihood of being assigned). Finally, the deadline for task t_i, D_i, is assigned as $D_i = (\delta_i \times med_i) + a_i$. All the subtasks within a task have that task's arrival time and deadline.

Because the system is oversubscribed, there may be tasks that cannot complete before their deadline. A deadline characteristic function, $\underline{d_i}$, indicating whether or not each task t_i is able to finish executing before its deadline, D_i, based on the mapping used, is

$$d_i = \begin{cases} 1 & : \quad \text{if } t_i \text{ finishes at or before } D_i \\ 0 & : \quad \text{otherwise.} \end{cases} \tag{1}$$

Several types of applications can be executed in more than one format or version. For example, weather information may be computed using varying sensor grid densities. Each task version may have different resource requirements and execution characteristics. It is assumed that each task has three versions available to execute, but at most one version of any given task is executed. It is assumed here that version i is always preferred over version $i+1$ of a task, but also requires more execution time (e.g., uses more resources).

The estimated expected execution time for task t_i, on machine m_j, using version $\underline{v_k}$ would be $ETC(i,j,k)$. Thus, based on the previous assumption, $ETC(i,j,0) > ETC(i,j,1) > ETC(i,j,2)$. To generate the simulated execution times in the ETC matrix, the coefficient-of-variation-based (CVB) method from [1] was used. The HC environment in the simulation study had eight machines.

The intuition behind allowing execution of lower preference versions of a task is that, typically, the lower preference versions will have reduced requirements, e.g., reduced execution times. Let $\underline{r_{ik}}$ be the normalized user-defined preference for version v_k of task t_i [23]. For the simulation studies:

$$\begin{aligned} r_{i0} &= 1 && \text{(most preferred)} \\ r_{i1} &= r_{i0} \times U[0,\ 0.99] && \text{(medially preferred)} \\ r_{i2} &= r_{i1} \times U[0,\ 0.99] && \text{(least preferred).} \end{aligned} \tag{2}$$

where $\underline{U[w,x]}$ is a uniform random (floating point) number sampled from $[w,\ x]$. The lowest version assigned to any subtask in a task is the version (and preference) enforced on all of the other subtasks within the task.

In instances where either a non-decomposed task or a communicating subtask (within a task) is being mapped, the term m-task (mappable task) is used. The number of m-tasks in the simulation study was $\underline{T} = 2000$. Approximately 1000 of the m-tasks are non-decomposed, and approximately 1000 of the m-tasks are subtasks. The size and inter-dependencies of the subtasks within a task were generated randomly. The number of subtasks within a task was $U[2,5]$. Thus, there were approximately 1000 non-decomposed tasks, and 285 tasks with subtasks. For evaluation purposes, there were $T_{eval} \approx 1285$ tasks.

3.3 Simulation Study

Greedy techniques perform well in many situations, and have been well-studied (e.g., [8,18]). One greedy technique, Min-min, has been shown to perform well in many situations (e.g., [5,6,18,31]), and is applied to this mapping problem.

To rate the quality of the mappings produced by the heuristics, a post-mapping evaluation function, $\underline{E}$, is used. Assume that if any version of task t_i completes, it is version v_k (k may differ for different tasks). Then, let E be defined as

$$E = \sum_{i=0}^{T_{eval}-1} (d_i \times p_i \times r_{ik}). \tag{3}$$

Here, the mapping problem is considered a maximization problem (i.e., higher values of E represent better mappings). Analysis of Eqn. 3 reveals the upper bound (UB) is $UB = \sum_{i=0}^{T_{eval}-1} p_i$.

To describe Min-min, it is useful to define $\underline{f_i}$, the task fitness for m-task t_i,

$$f_i = -(d_i \times p_i \times r_{ik}), \tag{4}$$

where t_i is executed using version v_k. The task fitness f_i represents (the negative of) the contribution of each task to the post-mapping evaluation function, E. Because Min-min is a minimization heuristic, it can be used to maximize E by simply applying it with $-E$ as the minimization criterion.

To compute d_i in the above equations, the task's completion time is found. Let $\underline{mav(i,j)}$ be the earliest time (after m-task t_i's arrival time) at which machine m_j (1) is not reserved, (2) is available for a long enough interval to execute t_i, and (3) can receive all input data if t_i is a subtask. Then, the completion time of m-task t_i, version v_k, on machine m_j, denoted $\underline{ct(i,j,k)}$, is $ct(i,j,k) = mav(i,j) + ETC(i,j,k)$.

The Min-min heuristic from [18] that was adapted and applied to this mapping problem is outlined now. Let $\underline{U}$ be the set of all unmapped m-tasks. Let $\underline{UP}$ be the set of all unmapped m-tasks such that if the m-task is a subtask, all of that subtask's predecessor subtasks have been mapped. The first phase of Min-min (the first "min") finds the best machine (i.e., minimum f_i) for each m-task in UP, and then stores these m-task/machine pairs in the set $\underline{Made}$. The m-tasks within $Made$ are referred to as candidate tasks.

After phase one has completed, phase two of Min-min selects the candidate task from $Made$ with the minimum f_i over all candidate tasks, and maps this m-task to its corresponding machine. This task is then removed from U. Phase one and two are repeated until all m-tasks are mapped (or removed from consideration because they cannot meet their deadline).

A comparison of how Min-min performs for two different arrival rates against two other mapping heuristics is shown in Fig. 3. The results are averaged over 50 different ETC matrices. The range bars show the 95% confidence interval for each average [19].

The other techniques shown in Fig. 3 are a simple FIFO-based greedy technique (MCF), and a genetic algorithm (GA) [6]. Recall that a higher value of the post-mapping evaluation function E represents a better mapping. Also note that because an oversubscribed system is assumed, the UB is unachievable. As shown in Fig. 3, Min-min does very well, achieving about 95% of the performance of the GA but with a significantly shorter running time.

For each arrival rate, the same number of tasks are available to map, thus UB is the same. However, for the moderate arrival rate, tasks arrive over a larger interval of time (and have deadlines over a larger interval of time). Hence, there is less contention for machines, and the heuristics perform better. Other variations of Min-min, other scenarios (including different weighted priorities), other heuristics (including the genetic algorithm), and their experimental results are defined and compared in [6].

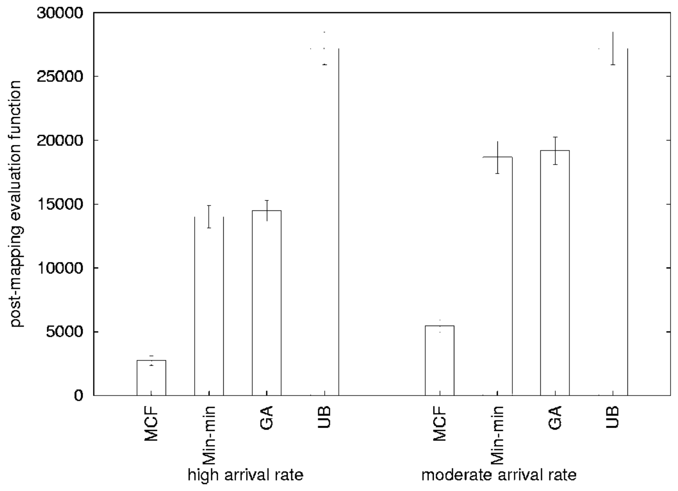

Fig. 3. Comparison of the Min-min technique against two other techniques for 2000 m-tasks with high and moderate arrival rates (based on [6]). Results are averaged over 50 ETC matrices

4 Environments and Applications

Examples of HC environments that have actually been deployed are: (1) the Purdue University Network Computing Hub, a wide area network computing system which can be used to run a selection of software tools via a World Wide Web browser [21]; (2) the Globus meta-computing infrastructure toolkit, a set of low-level mechanisms that can be built upon to develop higher level HC services [12]; and (3) SmartNet, a mapping framework that can be employed for managing jobs and resources in an HC environment [13].

Example applications that have demonstrated the usefulness of HC include: (1) a three-dimensional simulation of mixing and turbulent convection at the Minnesota Supercomputer Center [24]; (2) the shipboard anti-air warfare program (HiPer-D) used at the Naval Surface Warfare Center for threat detection, engagement, and missile guidance (e.g., [17]); and (3) a simulation of colliding galaxies performed by solving large n-body and large gas dynamics problems at the National Center for Supercomputing Applications [28].

5 Open Research Problems

HC is a relatively new research area for the computer field. Interest in HC systems continues to grow in the research, industrial, and military communities. However, there are many open problems that remain to be solved before HC can be made available to application programmers in a transparent way. Some of these problems are outlined below.

The realization of the automatic HC environment envisioned in Fig. 2 requires further research in many areas. Machine-independent languages with user-specified directives are needed to (1) allow compilation of a given application into efficient code for any machine in the HC suite, (2) aid in decomposing applications into tasks and subtasks, and (3) facilitate determination of task and subtask computational requirements. Methods must be refined for measuring the loading and status of the machines in the HC suite and the network, and for estimating task and subtask completion times. Also, the uncertainty present in the estimated parameter values, such as task completion times, should be taken into consideration in determining the mappings.

Another area of research is that of modeling the application tasks to execute on the HC suites [1,4]. Individual tasks that execute once (e.g., a simulation) have different requirements than continuously-running tasks (e.g., monitoring and analyzing sensor information). Methods for allowing feedback communications among subtasks tasks are necessary.

Several HC environments have inherent QoS requirements that must be met. For example, these requirements might involve priority semantics, bandwidth requirements, guaranteed processor time for certain users, or real-time response capabilities. Research is being conducted on how to incorporate all of these components into HC environments.

Hierarchical scheduling techniques are under development to allow for HC suites to be scaled up to very large sizes. Also, in some environments, distributed mapping heuristics (as opposed to centralized ones) are necessary. Incorporating multi-tasking is an area of ongoing HC research. Most operating systems support multi-tasking at the individual processor level, but how to incorporate this into the mapping process, allocate processor time among different tasks, and still leverage this capability with other QoS requirements is being investigated. Furthermore, the security issues inherent with multi-user, distributed, peer-to-peer environments where users are executing tasks on other machines must be addressed.

In summary, with the use of existing HC systems, significant benefits have been demonstrated. However, 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, make HC accessible to more users, and allow HC to realize its inherent potential.

Acknowledgments. The authors thank S. Ali, R. F. Freund, and L. L. Vandenberg for their comments. A version of this manuscript appeared as an invited keynote paper in the *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2001).*

References

1. Ali, S., Siegel, H. J., Maheswaran, M., Hensgen, D., Ali, S.: Representing task and machine heterogeneities for heterogeneous computing systems. Tamkang Journal of Science and Engineering. **3** (2000) 195–207
2. Armstrong, R.: Investigation of Effect of Different Run-Time Distributions on SmartNet Performance. Master's Thesis, Department of Computer Science, Naval Postgraduate School, Monterey, CA (1997)
3. Banicescu, I., Velusamy, V.: Performance of scheduling scientific applications with adaptive weighted factoring. In: 10th IEEE Heterogeneous Computing Workshop (HCW 2001), in the CD-ROM Proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS 2001). (2001) HCW_06
4. Braun, T. D., Siegel, H. J., Beck, N., Bölöni, L., Maheswaran, M., Reuther, A. I., Robertson, J. P., Theys, M. D., Yao, B.: A taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems. In: IEEE Workshop on Advances in Parallel and Distributed Systems, in the Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems. (1998) 330–335
5. Braun, T. D., Siegel, H. J., Beck, N., Bölöni, L. L., Maheswaran, M., Reuther, A. I., Robertson, J. P., Theys, M. D., Yao, B., Hensgen, D., Freund, R. F.: A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. Journal of Parallel and Distributed Computing. **61** (2001) 180–837
6. Braun, T. D.: Heterogeneous Distributed Computing: Off-line Mapping Heuristics for Independent Tasks and for Tasks with Dependencies, Priorities, Deadlines, and Multiple Versions. Ph.D. Thesis, School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN (2001)
7. Coffman, Jr., E. G. (ed.): Computer and Job-Shop Scheduling Theory. John Wiley & Sons, New York, NY (1976)
8. Cormen, T. H., Leiserson, C. E., Rivest, R. L.: Introduction to Algorithms. MIT Press, Cambridge, MA (1992)
9. Dietz, H. G., Cohen, W. E., Grant, B. K.: Would you run it here... or there? (AHS: Automatic heterogeneous supercomputing). In: International Conference on Parallel Processing. **II** (1993) 217-221
10. Eshaghian, M. M. (ed.): Heterogeneous Computing. Artech House, Norwood, MA (1996)
11. Fernandez-Baca, D.: Allocating modules to processors in a distributed system. In: IEEE Transactions Software Engineering. **SE-15** (1989) 1427–1436
12. Foster, I., Kesselman, C. (eds.): The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, San Francisco, CA (1999)
13. Freund, R. F., Gherrity, M., Ambrosius, S., Campbell, M., Halderman, M., Hensgen, D., Keith, E., Kidd, T., Kussow, M., Lima, J. D., Mirabile, F., Moore, L., Rust, B., Siegel, H. J.: Scheduling resources in multi-user, heterogeneous computing environments with SmartNet. In: 7th Heterogeneous Computing Workshop (HCW '98). (1998) 184–199
14. Freund, R. F., Siegel, H. J.: Heterogeneous processing. In: IEEE Computer. **26** (1993) 13–17
15. Ghafoor, A., Yang, J.: Distributed heterogeneous supercomputing management system. In: IEEE Computer. **26** (1993) 78–86

16. Hensgen, D. A., Kidd, T., Schnaidt, M. C., St. John, D., Siegel, H. J., Braun, T. D., Maheswaran, M., Ali, S., Kim, J.-K., Irvine, C., Levin, T., Wright, R., Freund, R. F., Godfrey, M., Duman, A., Carff, P., Kidd, S., Prasanna, V., Bhat, P., Alhusaini, A.: An overview of MSHN: A management system for heterogeneous networks. In: 8th Heterogeneous Computing Workshop (HCW '99). (1999) 184–198
17. Huh, E.-N., Welch, L. R., Shirazi, B. A., Cavanaugh, C. D.: Heterogeneous resource management for dynamic real-time systems. In: 9th IEEE Heterogeneous Computing Workshop (HCW 2000). (2000) 287–294
18. Ibarra, O. H., Kim, C. E.: Heuristic algorithms for scheduling independent tasks on nonidentical processors. Journal of the ACM. **24** (1977) 280–289
19. Jain, R.: The Art of Computer Systems Performance Analysis Techniques for Experimental Design, Measurement, Simulation, and Modeling. John Wiley & Sons, New York, NY (1991)
20. Kafil, M., Ahmad, I.: Optimal task assignment in heterogeneous distributed computing systems. In: IEEE Concurrency. **6** (1998) 42–51
21. Kapadia, N. H., Fortes, J. A. B.: PUNCH: An architecture for web-enabled wide-area network-computing. Cluster Computing: The Journal of Networks, Software Tools and Applications. **2** (1999) 153–164
22. Khokhar, A., Prasanna, V. K., Shaaban, M., Wang, C. L.: Heterogeneous computing: Challenges and opportunities. In: IEEE Computer. **26** (1993) 18–27
23. Kim, J.-K., Kidd, T., Siegel, H. J., Irvine, C., Levin, T., Hensgen, D. A., St. John, D., Prasanna, V. K., Freund, R. F., Porter, N. W.: Collective value of QoS: A performance measure framework for distributed heterogeneous networks. In: 10th IEEE Heterogeneous Computing Workshop (HCW 2001), in the CD-ROM Proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS 2001). (2001) HCW_08
24. Klietz, A. E., Malevsky, A. V., Chin-Purcell, K.: A case study in metacomputing: Distributed simulations of mixing in turbulent convection. In: 2nd Workshop on Heterogeneous Processing (WHP '93). (1993) 101–106
25. Maheswaran, M. Ali, S., Siegel, H. J., Hensgen, D., Freund, R. F.: Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. Journal of Parallel and Distributed Computing. **59** (1999) 107–121
26. Maheswaran, M., Braun, T. D., Siegel, H. J.: Heterogeneous distributed computing. In: Webster, J. G. (ed.): Encyclopedia of Electrical and Electronics Engineering. John Wiley & Sons, New York, NY. **8** (1999) 679–690
27. Michalewicz, Z., Fogel, D. B.: How to Solve It: Modern Heuristics. Springer-Verlag, New York, NY (2000)
28. Norman, M. L., Beckman, P., Bryan, G., Dubinski, J., Gannon, D., Hernquist, L., Keahey, K., Ostriker, J. P., Shalf, J., Welling, J., Yang, S.: Galaxies Collide on the I-way: An example of heterogeneous wide-area collaborative supercomputing. The International Journal of Supercomputer Applications and High Performance Computing. **10** (1996) 132–144
29. Siegel, H. J., Dietz, H. G., Antonio, J. K.: Software support for heterogeneous computing. In: Tucker, Jr., A. B. (ed.): The Computer Science and Engineering Handbook. CRC Press, Boca Raton, FL (1997) 1886–1909
30. Singh, H., Youssef, A.: Mapping and scheduling heterogeneous task graphs using genetic algorithms. In: 5th Heterogeneous Computing Workshop (HCW '96). (1996) 86–97
31. Wu, M.-Y., Shu, W., Zhang, H.: Segmented min-min: A static mapping algorithm for meta-tasks on heterogeneous computing systems. In: 9th IEEE Heterogeneous Computing Workshop (HCW 2000). (2000) 375–385