

A High-Performance Mapping Algorithm for Heterogeneous Computing Systems

Min-You Wu and Wei Shu
Department of Electrical and Computer Engineering
The University of New Mexico

Abstract

A mapping algorithm for heterogeneous computing systems is proposed in this paper. This algorithm utilizes a new indicator — the relative cost — to obtain optimal mapping. The existing Min-min algorithm can be well explained under synergy of this new indicator. It is found that the Min-min algorithm leaves room for improvement because of its haste to reduce completion time by overlooking the impact of load balance. Our new algorithm retains the advantages of the Min-min algorithm and balances the load very well. It demonstrates the ability to generate good mapping in various heterogeneous environments.

1. Introduction

It is a common phenomenon in the area of mapping and scheduling that a simple, early-invented algorithm performs fairly well and which even outperforms many later algorithms. The Min-min algorithm [10] is such an algorithm in a heterogeneous computing environment. Heterogeneous computing utilizes a suite of different machines interconnected by high-speed networks to execute different computationally intensive applications that have diverse computational requirements [13]. Static mapping assigns tasks to machines before runtime. The general problem of static mapping tasks to machines has been shown to be NP-complete [10]. Many useful heuristics have been designed for this mapping function. Among many sophisticated algorithms, the Min-min algorithm stands out as a simple algorithm, running fast and delivering satisfactory performance.

In every iteration of the Min-min algorithm, from all unmapped tasks a new task is selected and assigned to a particular machine such that the completion time is minimized. This simple procedure exhibits a reasonably good performance, especially compared to many other rather complicated algorithms. However, for lack of a comprehensive explanation, the underlying characteristics of the Min-min algorithm are not thoroughly understood. Through a considerable amount of experimentation and investigation, we

found that good performance of the Min-min algorithm lies mainly in its tendency toward mapping tasks to their first choices in terms of execution time. However, the Min-min algorithm is unable to balance the load well since it usually maps small tasks first. With this observation, we propose a new algorithm. The proposed algorithm retains the advantage of the Min-min algorithm and achieves good load balance at the same time. The new algorithm, named the *Relative Cost (RC)* algorithm, is presented in this paper. RC separates the relative execution cost from the execution time of a task. Thus, large tasks with low costs can be assigned early for a good load balance.

2 A Revisit of the Min-min Algorithm

A heterogeneous system is characterized by a static matrix *ETC*. The *Expected Time to Compute*, $ETC(i, j)$, is the estimated execution time for task i on machine j . If there are t tasks and m machines, we can obtain a $t \times m$ *ETC matrix*.

For each task i , there is a boolean flag, $\rho(i)$, which is set if task i has been mapped on a machine j , $j = \mathcal{F}(i)$, where $\mathcal{F}(i)$ is a mapping function. At iteration k , k tasks had been mapped. Each machine j has a current time $C_k(j)$, which depends on which tasks have been mapped on machine j as well as their execution time:

$$C_k(j) = \sum_{1 \leq l \leq t} \{ETC(l, j) | \rho(l) = 1 \text{ and } \mathcal{F}(l) = j\}$$

Initially, $C_0(j) = 0$ for every machine j .

Mapping a new task i on top of mapped tasks, the completion time on machine j is defined as $ct_k(i, j)$:

$$ct_k(i, j) = ETC(i, j) + C_{k-1}(j)$$

Many heuristic algorithms have been proposed to map meta-tasks to heterogeneous computing systems. A meta-task is defined as a collection of independent tasks with no data dependences. In these heuristics, meta-tasks are mapped onto machines statically; each machine executes a single task at a time. For static mapping, it is assumed that

the number of tasks, t , and the number of machines, m , are known a priori. In [4], eleven commonly used algorithms have been evaluated, that are, OLB [3, 9], UDA [3, 9], Fast Greedy [3], Min-min [3, 9, 10], Max-min [3, 9, 10], Greedy [3, 9], Genetic algorithm (GA) [16, 14], Simulated Annealing (SA) [7, 11], GSA [5], Tabu [8], and A* [6]. The experimental results from [4] show that OLB, UDA, Max-min, SA, GSA, and Tabu do not produce good mapping in general. Min-min, GA, and A* are able to deliver good performance. The difference between the completion times of mapping (makespans) generated by these three algorithms is normally within 5%. GA is consistently better than Min-min by a few percents, since it is seeding the population with a Min-min chromosome. A*, on the other hand, produces better or worse mapping than Min-min and GA in different situations. Among the three algorithms, Min-min is the fastest algorithm, GA is much slower, and A* is very slow. For 512 tasks and 16 machines, the running time of Min-min is about 1 second, GA 100 seconds, and A* 1,200 seconds. The Sufferage heuristic [12] is an algorithm proposed for dynamic mapping which also generates better solutions than Min-min.

Min-min is a simple algorithm, fast, and able to deliver relatively good performance. Even GA has to “seed” the population with a Min-min chromosome to obtain its good performance. Min-min maps the “best case” tasks first. To determine the priorities of tasks for mapping, the Min-min algorithm selects, from all remaining tasks, one that minimizes the completion time. That is, at iteration k , Min-min picks task i as the next task to be mapped to machine j such that,

$$ct_k(i, j) = \min_{1 \leq p \leq t, 1 \leq q \leq m} \{ct_k(p, q) | \rho(p) = 0\}$$

$$= \min_{1 \leq p \leq t, 1 \leq q \leq m} \{ETC(p, q) + C_{k-1}(q) | \rho(p) = 0\}$$

In a heterogeneous system, a common criterion is to match a task to the machine that executes the task fastest. Here, although the machine selected is not necessarily the task’s first choice of machines, it is quite possible to obtain a good match between the task and the machine.

As one of its limitations, Min-min also gives small tasks higher priorities and therefore assigns them early, going against a general principle that the large tasks should be mapped first for a balanced load. When small tasks execute first, it tends to execute a few larger tasks near the end, leaving some machines sitting idle, which results in poor system utilization. If large tasks were mapped first, small tasks can run in parallel with other tasks to fill up holes at the end, achieving a balanced load.

From this observation, we summarize two essential criteria for a high-quality mapping algorithm for heterogeneous computing systems, matching and load balancing:

1) Matching. To better match the tasks and machines, a task should be mapped to the machine with smaller ETC.

2) Load balancing. To better utilize the machines, the load should be balanced to minimize the machine idle time.

To balance the load, the sizes of tasks must be taken into consideration. In a heterogeneous system, execution time of a task varies on different machines. The task size can simply be measured as the average of the execution times over all machines. Tasks can be assigned with priorities based on their sizes, either favorable to the smaller tasks or to the larger tasks. Or the priority can be assigned independently of task sizes. It has been found that the load is severely imbalanced when smaller tasks are mapped first. If large tasks are given higher priorities, it generally leads to a better balanced load. Also, a priority independent of tasks sizes has a fairly good chance for load balancing.

Matching of tasks and machines can be measured by a parameter, *matching proximity*. When every task is executed on the machine with the shortest execution time, such as in the UDA algorithm, it is defined as the ideal matching $\mathcal{G}(i), 1 \leq i \leq t$. We define $j = \mathcal{F}(i)$ as a general mapping from the task domain $i = 1, \dots, t$ to the machine domain $j = 1, \dots, m$. Then, the ideal matching $\mathcal{G}(i)$ can be represented as,

$$\mathcal{G}(i) = j \quad \text{such that} \quad ETC(i, j) = \min_{1 \leq g \leq m} ETC(i, g)$$

for task i . Any matching other than the ideal matching \mathcal{G} may demand longer total execution time of all tasks, since

$$\sum_{1 \leq i \leq t} ETC(i, \mathcal{F}(i)) \geq \sum_{1 \leq i \leq t} ETC(i, \mathcal{G}(i)).$$

Thus, *matching proximity* is defined as follows:

$$\eta = \frac{\sum_{1 \leq i \leq t} ETC(i, \mathcal{G}(i))}{\sum_{1 \leq i \leq t} ETC(i, \mathcal{F}(i))},$$

where $\eta \leq 1$. When $\eta = 1$, we have the ideal matching. Load balance can be measured by *system utilization*:

$$\mu = \frac{\sum_{1 \leq i \leq t} ETC(i, \mathcal{F}(i))}{m \times T_m},$$

where T_m is the makespan (completion time) on m machines, which equals

$$T_m = \max_{1 \leq j \leq m} C_t(j).$$

When the system is completely balanced, $\mu = 1$; otherwise, $\mu < 1$. Thus, the makespan of a mapping can be expressed as:

$$T_m = \frac{\sum_{1 \leq i \leq t} ETC(i, \mathcal{G}(i))/m}{\eta \times \mu},$$

where $\sum_{1 \leq i \leq t} ETC(i, \mathcal{G}(i))/m$ is a lower bound of the makespan.

The Min-min algorithm satisfies the first matching criterion but not the second one, load balancing. It uses the

minimum completion time as the criterion to select a task for mapping. Thus, the small tasks get a higher priority and are mapped first so that the load cannot be balanced well.

The objective of mapping is to minimize the makespan. In a homogeneous system, it is equivalent to maximizing load balancing. In a heterogeneous system, in addition to load balancing, a task should be mapped to a machine that can execute the task fastest. An ideal algorithm should satisfy both criteria simultaneously. However, these design goals are in conflict with each other. Mapping tasks to their first choice of machines may cause load imbalance, especially in the *consistent case*, where all tasks have the same first choice of machines. To balance the load, some tasks have to be mapped to other machines instead of their first choices. Thus, the mapping problem is essentially a trade-off between the two criteria. A good algorithm must balance between matching proximity and system utilization.

3 Simulation Model

Here we summarize the simulation model presented in [4], which is used for our comparison study. In this model, characteristics of the ETC matrices were varied in an attempt to represent a variety of possible heterogeneous computing environments. This kind of variation is represented by setting the value scope of random numbers used to produce ETC matrices. Initially, a $t \times 1$ baseline column vector, B , of floating point values is created. Let ϕ_b be the upper-bound of the range of possible values within the baseline vector. B is generated by repeatedly selecting a uniform random number, $x_b^i \in [1, \phi_b)$, and letting $B(i) = x_b^i$ for $0 \leq i < t$. Next, the rows of the ETC matrix are constructed. Each element $ETC(i, j)$ in row i of the ETC matrix is created by taking the baseline value, $B(i)$, and multiplying it by a uniform random number, $x_r^{i,j}$, which has an upper-bound of ϕ_r . This new random number, $x_r^{i,j} \in [1, \phi_r)$, is called a row multiplier. One row requires m different row multipliers, $0 \leq j < m$. Each row i of the ETC matrix can be then described as $ETC(i, j) = B(i) * x_r^{i,j}$, for $0 \leq j < m$. This process is repeated for each row until the $t \times m$ ETC matrix is full. Therefore, any given value in the ETC matrix is within the range $[1, \phi_b \times \phi_r)$.

To let the ETC matrix denote various kinds of possible mapping situations, the task heterogeneity and machine heterogeneity were defined. Task heterogeneity was varied by changing the upper-bound of the random numbers within the baseline column vector. High task heterogeneity was represented by $\phi_b = 3000$ and low task heterogeneity used $\phi_b = 100$. Machine heterogeneity was varied by changing the upper-bound of the random numbers used to multiply the baseline values. High machine heterogeneity values were generated using $\phi_r = 1,000$, while low machine het-

erogeneity values used $\phi_r = 10$.

Different ETC matrix consistencies were used to capture more aspects of realistic mapping situations. An ETC matrix is said to be inconsistent if the ETC matrices are kept in the unordered, random state in which they were created. The ETC matrix indicates consistent characteristics if a machine j executes any task i faster than machine k , then machine j executes all tasks faster than machine k [2]. The consistent matrix can be obtained by sorting every row of the matrix independently. Between two special situations, a Semi-consistent matrix represents a partial ordering among the machine/task execution times. For the Semi-consistent matrix used here, the row elements in even columns of row i are extracted, sorted and replaced in order, while the row elements in odd columns remain unordered.

4 The Relative Cost Algorithm

We propose a new algorithm, named *Relative Cost (RC)* algorithm. By assigning priorities independently of task sizes, the load can be balanced well since small tasks are no longer given higher priorities. Instead, a new criterion, *relative cost*, will determine the mapping order of tasks. Once a task is selected, it is mapped to the machine that allows the earliest completion time.

In general, when designing a mapping algorithm, there are two most important mapping decisions:

- which task is assigned first; and
- which machine the task is assigned to.

Let's discuss the second decision first. Although mapping a task to its first choice of machines minimizes its overall execution cost, it is not favorable to load balancing in general. It has been shown that in a homogeneous system, a heuristics that maps a task to its earliest start time generates good schedules [1]. In a heterogeneous system, this rule is modified to the earliest finish (completion) time [15]. Following this rule, a task is assigned to the machine that allows the earliest completion time. The difficult decision is which task should be assigned first. This issue is addressed for both load balancing and matching criteria as follows.

As mentioned earlier, in the Min-min algorithm the task with the smallest completion time is assigned first. It can achieve good matching, but ends up assigning high priorities to small tasks, which in turn will have an impact on load balance. To remedy this limitation, we divide the completion time $ct_k(i, j)$ by the average completion time of tasks i to obtain a *relative cost*, γ . Consequently, the tasks are assigned with different priorities based on their *relative cost*, instead of their sizes.

By using the relative cost, we present the RC algorithm here. First, we define the following terms:

RC Algorithm

- 1) For each task i and machine j , let $ct_1(i, j) = ETC(i, j)$ and $\rho(i) = 0$
 $\gamma_s(i, j) = ETC(i, j)/ETC(i)_{avg}$
 - 2) For $k = 1$ to t do
 - 2.1) for task i , $1 \leq i \leq t$ and $\rho(i) = 0$
 - 2.1.1) compute $ct_k(i)_{avg} = \sum_{1 \leq j \leq m} ct_k(i, j)/m$
 - 2.1.2) select machine B_i such that $ct_k(i, B_i) = \min_{1 \leq j \leq m} ct_k(i, j)$
 - 2.1.3) compute $\gamma_d(i, B_i) = ct_k(i, B_i)/ct_k(i)_{avg}$
 - 2.2) select task A_k such that

$$\gamma_s(A_k, B_{A_k})^\alpha \times \gamma_d(A_k, B_{A_k}) = \min_{1 \leq i \leq t, \rho(i)=0} \gamma_s(i, B_i)^\alpha \times \gamma_d(i, B_i)$$
 - 2.3) let $\rho(A_k) = 1$ and $\mathcal{F}(A_k) = B_{A_k}$, that is, assign task A_k to machine B_{A_k}
 - 2.4) for task i with $1 \leq i \leq t$ and $\rho(i) = 0$, modify

$$ct_{k+1}(i, j) = \begin{cases} ct_k(i, j) & \text{if } j \neq B_{A_k} \\ ct_k(i, j) + ETC(A_k, B_{A_k}) & \text{otherwise} \end{cases}$$
-

Figure 1. The RC Algorithm.

- the average ETC time for task i over all machines:

$$ETC(i)_{avg} = \sum_{1 \leq j \leq m} ETC(i, j)/m.$$

- the average completion time for task i over all machines after k tasks have been mapped:

$$ct_k(i)_{avg} = \sum_{1 \leq j \leq m} ct_k(i, j)/m.$$

- *Static Relative Cost* is defined as:

$$\gamma_s(i, j) = ETC(i, j)/ETC(i)_{avg}.$$

- *Dynamic Relative Cost* is defined as:

$$\gamma_d(i, j) = ct_k(i, j)/ct_k(i)_{avg}.$$

The complete RC algorithm is shown in Figure 1. An integrated factor, $\gamma_d^\alpha \times \gamma_s$, is used to take both matching and load balance criteria into consideration. The smaller the γ_d is, the better matching of machines. Now, the task that matches machines better as well as benefits load balance will be assigned a higher priority under this new indicator. Parameter α is used to control the effect of static relative cost, where $0 \leq \alpha \leq 1$.

In the RC algorithm, the higher priority is given to tasks that

- have a good match between tasks and machines; and
- minimize the completion time.

To better balance between the matching and load balancing criteria, the best value of α is determined. From experiment, a higher value α is better for the low machine heterogeneity and a lower value of α is better for the high machine heterogeneity. Although the value of α can be dynamically changed to adapt to different cases, we use the value $\alpha = 0.5$ in the following experiments for simplicity.

5 Experiments

5.1 Performance Comparison

In this performance study, we use the same simulation model described above. A program was used to produce various kinds of ETC matrix by user-specified parameters. These parameters included the consistent parameter (inconsistent, consistent, semi-consistent), task heterogeneity (high, low) and machine heterogeneity (high, low). The results of 100 different matrices were averaged for each case to compare different algorithms.

Here, we compare the RC algorithm to the Max-min, Min-min, and GA algorithms. GA is the best algorithm in the group of 11 selected algorithms. As mentioned above, Min-min is a fairly good algorithm and GA is seeding on it. Max-min is selected partly because it uses an opposite indicator to Min-min, partly because it archives the best load balance. All experimental results are for 512 tasks. The results on 16 machines are shown in Table 1. In this table, the third column shows system utilization and the fourth column the matching proximities. The fifth column is the makespan (completion time). The running time of Max-min, Min-min, and RC is about 1 second, whereas that of

Table 1. A Comparison: 16 Machines, Inconsistent

	Alg.	System Utiliz.	Match. Proxi.	Makespan (seconds)
Low Task, Low Mach. Heterog.	Max-min	99.9%	38.3%	5.425×10^3
	Min-min	91.0%	98.6%	2.915×10^3
	GA	94.1%	98.1%	2.833×10^3
	RC	99.7%	97.6%	2.691×10^3
Low Task, High Mach. Heterog.	Max-min	99.8%	39.2%	2.513×10^5
	Min-min	83.3%	97.3%	1.214×10^5
	GA	88.5%	94.6%	1.175×10^5
	RC	97.4%	94.7%	1.067×10^5
High Task, Low Mach. Heterog.	Max-min	99.9%	48.4%	15.943×10^4
	Min-min	91.0%	98.5%	8.589×10^4
	GA	94.5%	97.4%	8.353×10^4
	RC	99.7%	97.5%	7.926×10^4
High Task, High Mach. Heterog.	Max-min	99.8%	39.3%	7.375×10^6
	Min-min	83.4%	97.2%	3.573×10^6
	GA	88.8%	94.3%	3.457×10^6
	RC	97.4%	94.6%	3.143×10^6

Table 2. Improvement of RC over Max-min, Min-min, and GA on 16 Machines

		Improv. over Max-Min	Improv. over Min-min	Improv. over GA
Inconsistent	Low-Low	101.6%	8.3%	5.3%
	Low-High	135.5%	13.8%	10.1%
	High-Low	101.1%	8.4%	5.7%
	High-High	135.3%	13.7%	10.0%
Consistent	Low-Low	30.8%	3.3%	0.9%
	Low-High	50.2%	4.3%	1.1%
	High-Low	30.7%	3.4%	1.3%
	High-High	50.2%	4.3%	1.3%
Semi-consistent	Low-Low	78.7%	5.6%	2.4%
	Low-High	107.3%	7.8%	4.1%
	High-Low	78.4%	5.7%	3.2%
	High-High	106.2%	7.4%	3.7%

Table 3. Improvement of RC over Max-min, Min-min, and GA on 32 Machines

		Improv. over Max-Min	Improv. over Min-min	Improv. over GA
Inconsistent	Low-Low	73.4%	14.8%	11.5%
	Low-High	80.8%	11.9%	9.1%
	High-Low	73.3%	14.6%	12.1%
	High-High	79.9%	11.6%	8.9%
Consistent	Low-Low	20.0%	7.2%	4.5%
	Low-High	37.1%	4.1%	0.2%
	High-Low	19.9%	7.0%	7.6%
	High-High	36.9%	4.1%	0.2%
Semi-consistent	Low-Low	60.2%	9.9%	6.9%
	Low-High	91.7%	14.7%	10.5%
	High-Low	60.0%	10.0%	7.6%
	High-High	92.2%	14.7%	10.4%

GA is about 100 seconds. System utilization of Max-min and RC is high, whereas that of Min-min is consistently lower. Min-min can achieve a high matching proximity, whereas that of Max-min is very low. GA improve Min-min by increasing system utilization but reducing the matching proximity at the same time. Note that to achieve a more balanced load, the matching proximity cannot be too high. Also note that RC and GA have the similar matching proximity, but the system utilization of RC is higher than GA, indicating that RC achieves global optimization of load balancing. The RC algorithm is able to reach a compromise between the system utilization and the matching proximity. It achieves a balanced load, without sacrificing its matching proximity at the same time, and is therefore able to outperform Min-min and GA in all cases.

Tables 2 and 3 show the improvement of the RC algorithm over Max-min, Min-min, and GA on 16 machines and 32 machines, respectively. We ran these algorithms on 32 machines since a lower task/machine ratio will highlight the load balancing requirement. Improvement of RC over Max-min is between 30.7% and 135.5% on 16 machines with an average of 75.8%, and on 32 machines is between 19.9% and 92.2% with an average of 60.5%. Improvement of RC over Min-min is between 3.3% and 14.0% on 16 machines with an average of 7.2%, and on 32 machines is between 4.1% and 14.8% with an average of 10.4%. Improvement of RC over GA is between 0.9% and 10.3% on 16 machines with an average of 4.1%, and on 32 machines is between 0.2% and 12.1% with an average of 7.5%.

5.2 Discussion

In this group of algorithms, Max-min has the lowest matching proximity. This is because it uses a wrong indicator for task selection — the largest completion time, even if it assigns tasks to machines with minimum completion time. The purpose of using the largest completion time as the criterion was to select large tasks to map first. However, the same indicator used to select tasks often leads to a poor match between tasks and machines. Though Max-min has a very good load balancing, it suffers from mismatching tasks and machines and delivers poor performance.

Min-min is not good on load balancing because it maps smaller tasks first. GA can improve load balance of Min-min in some extent but its ability to load balance is still poor compared to Max-min and RC. Max-min selects the larger tasks first and balances the load very well. On the other hand, RC does not bias toward either small tasks or large tasks and can achieve good load balancing.

A good mapping algorithm compromises between two conflicting goals — the matching proximity and the load balance. The experimental results have shown that the RC algorithm is able to produce both good matching proximity and load balancing. It delivers the best performance in the group of selected algorithms in all cases.

Max-min, Min-min, and RC have similar running time since they have the same complexity. GA, although it has the ability to improve existing mapping, spends much longer time than the other algorithms.

6 Conclusion

The Relative Cost algorithm combines the advantages of Max-min and Min-min algorithms, balances the load very well and matches tasks to machines at the same time. It delivers good performance on different heterogeneous environments such as the Consistent case as well as Inconsistent cases. It outperforms the Genetic algorithm that delivered the best performance among 11 existing algorithms, and runs much faster than GA.

Acknowledgments

We are very grateful to Howard Jay Siegel and Muthucumar Maheswaran for providing the Genetic algorithm code written by Lee Wang, and Hong Zhang for her help on the simulation study. This research was partially supported by NSF grants CCR-9505300 and CCR-9625784.

References

[1] I. Ahmad, Y. Kwok, and M. Y. Wu. Performance comparison of algorithms for static scheduling of DAGs to multiprocessors. In *Second Australasian Conference on Parallel and Real-time Systems*, pages 185–192, Sept. 1995.

[2] R. Armstrong. *Investigation of Effect of Different Run-Time Distribution on SmartNet Performance*. PhD thesis, Department of Computer Science, Naval Postgraduate School, 1997.

[3] R. Armstrong, D. Hensgen, and T. Kidd. The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions. In *7th IEEE Heterogeneous Computing Workshop (HCW '98)*, pages 79–87, Mar. 1998.

[4] T. Braun, H. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, B. Yao, D. Hensgen, and R. Freund. A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems. In *8th IEEE Heterogeneous Computing Workshop (HCW '99)*, pages 15–29, Apr. 1999.

[5] H. Chen, N. S. Flann, and D. W. Watson. Parallel genetic simulated annealing: a massively parallel SIMD approach. *IEEE Transactions on Parallel and Distributed Computing*, 9(2):126–136, Feb. 1998.

[6] K. Chow and B. Liu. On mapping signal processing algorithms to a heterogeneous multiprocessor system. In *ICASSP 91*, pages 1585–1588, May 1991.

[7] M. Coli and P. Palazzari. Real time pipelined system design through simulated annealing. *Journal of Systems Architecture*, 42(6-7):465–475, Dec. 1996.

[8] I. D. Falco, R. D. Balio, E. Tarantino, and R. Vaccaro. Improving search by incorporating evolution principles in parallel tabu search. In *IEEE Conference on Evolutionary Computation*, pages 823–828, 1994.

[9] R. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J. Lima, F. Mirabile, L. Moore, B. Rust, and H. Siegel. Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet. In *7th IEEE Heterogeneous Computing Workshop (HCW '98)*, pages 184–199, Mar. 1998.

[10] O. Ibarra and C. Kim. Heuristic algorithms for scheduling independent tasks on nonidentical processors. *Journal of the ACM*, 77(2):280–289, Apr. 1977.

[11] S. Kirkpatrick, J. C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[12] M. Maheswaran, S. Ali, H. Siegel, D. Hensgen, and R. Freund. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. *Journal of Parallel and Distributed Computing*, 59:107–131, 1999.

[13] M. Maheswaran, T. D. Braun, and H. J. Siegel. *Encyclopedia of Electrical and Electronics Engineering*, chapter Heterogeneous Distributed Computing. John Wiley & Sons, 1999.

[14] M. Srinivas and L. Patnaik. Genetic algorithms: A survey. *IEEE Computer*, 27(6):17–26, June 1994.

[15] H. Topcuoglu, S. Hariri, and M. Wu. Task scheduling algorithms for heterogeneous resources. In *8th IEEE Heterogeneous Computing Workshop (HCW '99)*, pages 1–14, Apr. 1999.

[16] L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski. Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach. *Journal of Parallel and Distributed Computing*, 47(1):1–15, Nov. 1997.