Study of an Iterative Technique to Minimize Completion Times of Non-Makespan Machines

Luis Diego Briceño[1], Mohana Oltikar[1], Howard Jay Siegel[1,2], and Anthony A. Maciejewski[1]

Colorado State University
[1]Department of Electrical & Computer Engineering
[2]Department of Computer Science
Fort Collins, CO 80523-1373
{ldbricen, mohana, hj, aam}@colostate.edu

Abstract

Heterogeneous computing (HC) is the coordinated use of different types of machines, networks, and interfaces to maximize the combined performance and/or cost effectiveness of the system. Heuristics for allocating resources in an HC system have different optimization criteria. A common optimization criterion is to minimize the completion time of the last to finish machine (makespan). In some environments, it is useful to minimize the finishing times of the other machines in the system, i.e., those machines that are not the last to finish. Consider a production environment where a set of known tasks are to be mapped to resources off-line before execution begins. Minimizing the finishing times of all the machines will provide the earliest available ready time for these machines to execute tasks that were not initially considered. In this study, we examine an iterative approach that decreases machine finishing times by repeatedly running a resource allocation heuristic. The goal of this study is to investigate whether this iterative procedure can reduce the finishing time of some machines compared to the mapping initially generated by the heuristic. We show that the effectiveness of the iterative approach is heuristic dependent and study the behavior of the iterative approach for each of the chosen heuristics. This work which identifies heuristics can and cannot attain improvements in the completion time of non-makespan machines using this iterative approach.

This research was supported by the NSF under grant CNS-0615170 and by the Colorado State University George T. Abell Endowment.
1-4244-0910-1/07/$20.00 ©2007 IEEE.

1 Introduction

The use of heuristics for resource allocation in a heterogeneous computing (HC) environment is an important area of research and has been widely studied (e.g., [2, 3, 15, 18]). One metric for evaluating the performance of heuristics is the maximum completion time over all machines (makespan). The makespan machine is defined as the machine with the largest completion time.

In some environments, it is useful to minimize the finishing times of the non-makespan machines, i.e., those machines that are not the last to complete. As an example, consider a production environment where a set of known tasks are mapped to resources off-line before execution begins. Minimizing the finishing times of all the machines will provide the earliest available ready time for these machines to execute tasks that were not initially considered. In this study, we present one approach, for attempting the finishing time of each machine in a given resource allocation. There are different ways to capture the concept of minimizing the finishing times of a set of heterogeneous machines, e.g., minimize the average finishing time, or minimize the largest finishing time among the machines. This iterative approach decreases the finishing time by repeatedly running a mapping heuristic to minimize the makespan of the considered machines and tasks. For each successive iteration, the makespan machine of the previous iteration and the tasks assigned to it are removed from the set of considered machines and tasks. This work studies the performance of resource allocation heuristics that use the iterative approach.

This paper has two main contributions. The first contribution is the introduction of an iterative technique that may

be able to reduce the completion times of non-makespan machines when used with certain heuristics. The second contribution is a detailed mathematical study of the performance of a variety of resource allocation heuristics for which the iterative technique does not change the mapping.

The heuristics considered for this study were Minimum Execution Time, Minimum Completion Time, Min-Min [8], Genitor [17], Switching Algorithm [14], Sufferage [14, 4], and K-Percent Best [14]. The remainder of the paper is organized as follows. Section 2 describes the problem statement in detail. The heuristics are discussed in detail in Section 3. The related work is reviewed in Section 4, and Section 5 concludes the work.

2 Problem Statement

Let T be the set of tasks that must be executed on a set of machines, M. The estimated time to compute (ETC) each task on each machine is assumed to be known in advance and contained in an ETC matrix [3]. The ETC values can be based on user supplied information, experimental data, or task profiling and analytical benchmarking [1, 6, 7, 10, 13, 20]. Determination of ETC values is a separate research problem; the assumption of such ETC information is a common practice in resource allocation research (e.g., [7, 9, 10, 12, 16, 19]). The initial ready time for a machine is the time at which the machine will become available to begin processing its first task from the set of tasks T. Tasks are assumed to be independent, i.e., no inter-task communication is required. We make the common simplifying assumption that each machine can only execute one task at a time, i.e., multitasking is not allowed (e.g., [5, 11]).

For each heuristic, the mapping it produces when all tasks and machines are available is called the original mapping. After each iteration (of the iterative approach), the makespan machine and the tasks assigned to it are removed from consideration, and the ready times for all other machines are reset to their initial ready times. The tasks that are available for mapping (mappable tasks) are mapped again, using the same heuristic to minimize makespan among the remaining machines; this mapping is called the iterative mapping. This iterative process is repeated until only one machine remains. The goal of this study is to investigate whether this iterative procedure can reduce the finishing time of some machines compared to using only the original mapping. We show that the effectiveness of the iterative approach is heuristic dependent and study the behavior of the iterative approach for each of the chosen heuristics.

Whether the iterative approach will change a mapping often depends on how ties are broken within a heuristic. A tie in a resource allocation heuristic is when a heuristic must choose from two equally good solutions, i.e., the heuristic determines both mappings are the best possible mappings. Two types of methods to break ties will be considered for this study. The first method is to break ties deterministically, e.g., the oldest task is chosen. The second method is to break ties randomly, e.g., if two machines are tied each will have a 0.5 probability of being chosen.

We define the function RT that receives as an argument a machine (m) from the HC suite and returns the ready time for this machine given the tasks that are currently assigned to the machine. The function ETC receives as arguments a task (t) and a machine (m) from the HC suite and returns the estimated time to compute. We can then define the completion time, CT, of a new task t on machine m with Equation 1.

$$CT(t, m) = ETC(t, m) + RT(m) \tag{1}$$

3 Heuristics

3.1 Genitor

Genitor [17] is a steady-state genetic algorithm that has been shown to work well for several problem domains, including resource allocation and job shop scheduling. Genitor uses chromosomes to represent possible solutions, e.g., all tasks and the machines to which they are assigned. Genitor has a population that consists of multiple chromosomes and has two operators to search for better solutions. The first operator is crossover, an operator that combines two chromosomes to produce two new chromosomes. The second operator is mutation, an operator that has a probability of changing tasks assignments within a chromosome. Genitor can be summarized by the procedure shown in Figure 1.

For each iteration (of the iterative approach), the mapping found by Genitor in the previous iteration, excluding the makespan machine and the tasks assigned to it, is seeded into the population of the current iteration. The ranking in Genitor guarantees that the final mapping is either the seeded mapping or a mapping with a smaller makespan, among the machines considered in the current iteration. Thus, for Genitor the iterative technique will result in either an improvement or no change.

3.2 Min-Min

The Min-Min heuristic [8] is a two phase greedy heuristic. The procedure for this heuristic is given in Figure 2. The performance of the Min-Min heuristic will depend on the method used to break ties. If the ties are broken deterministically then the individual completion times for each machine do not improve. If ties are broken randomly then the makespan can increase.

1 An initial population of mappings is generated.

2 The mappings in the population are ordered based on makespan.

3 while (the stopping criteria not met)

  a Two chromosomes are randomly selected to act as parents for crossover.

    i A random cut-off point is generated.

    ii the machine assignments of the tasks below the cut-off point are exchanged.

    iii The offspring is inserted into the sorted population based on its makespan. The worst chromosomes are removed (population size stays fixed).

  b A chromosome is randomly selected for mutation.

    i For the chosen chromosome, a random task is chosen and its machine assignment is arbitrarily modified.

    ii The offspring is inserted into the sorted population based on its makespan. The worst chromosome is removed (population size stays fixed).

4 The best solution is output.

Figure 1. Summary of one possible procedure that can be used to implement Genitor

1 A task list is generated that includes all the tasks as unmapped tasks.

2 For each task in the task list, the machine that gives the task its minimum completion time (first Min) is determined (ignoring other unmapped tasks).

3 Among all task-machine pairs found in 2, the pair that has the minimum completion time (second Min) is determined.

4 The task selected in 3 is removed from the task list and is mapped to the paired machine.

5 The ready time of the machine on which the task is mapped is updated.

6 Steps 2-5 are repeated until all tasks have been mapped.

Figure 2. Procedure for Min-Min

Theorem

If Min-Min is used as a mapping heuristic with the iterative approach and ties are broken deterministically, then the mappings produced by all the iterations are identical.

Proof

The initial ready times, for this proof, can be considered 0 without loss of generality. Assume the makespan machine of the original mapping is machine μ. The iterative approach does not change the assignment of tasks that were assigned to machine μ in the original mapping; i.e., tasks assigned to the makespan machine in the original mapping are ignored in the first iterative mapping.

Inductive hypothesis: Consider the n^{th} task mapped by Min-Min in the original mapping. Let $P(n)$ be the statement that the n^{th} task will have the same completion time and be mapped to the same machine in both the original mapping and the first iterative mapping, if ties are broken deterministically. For the basis and inductive steps, there are two cases to consider: the case when the task is mapped to machine μ and the case when the task is not mapped to machine μ.

To prove $P(n)$ is true for $\forall n \geq 1$ we need to prove:
1) $P(1)$ is true
2) $(\forall k)$ [$P(r)$ is true for all r, $1 \leq r \leq k \Rightarrow P(k+1)$ is true]

Basis Step: Prove that $P(1)$ is true.
Case 1: The first task to be mapped in the original mapping is assigned to machine μ. For the first case, $P(1)$ is clearly true because the tasks assigned to machine μ in the original mapping remain assigned to machine μ in the iterative mapping;
Case 2: The first task, denoted t_1, to be mapped in the original mapping is not assigned to machine μ. In the original mapping, let t_1 be assigned to machine β_1 ($\beta_1 \in M$ and $\beta_1 \neq \mu$). Given the definition of the Min-Min mapping procedure and that ready times are 0 for all machines, t_1 is assigned to its minimum execution time machine. For the iterative mapping, t_1 is still the first task to be assigned, therefore, all of the machines are idle and t_1 is assigned to its minimum completion time machine. Because the ETC values in the iterative mapping are identical to those in the original mapping, the minimum execution time machine (i.e., the minimum completion time machine) for task t_1 remains

	machines		
task	m_1	m_2	m_3
t_1	4	1	3
t_2	5	6	7
t_3	3	1	2
t_4	5	5	4

Table 1. ETC matrix for Min-Min example

unchanged from the original mapping. Therefore, $P(1)$ is clearly true.

Inductive Step: For the inductive step assume that $\forall r$ $P(1 \leq r \leq k)$ is true and prove $P(k+1)$ is true. The assumption that $P(1 \leq r \leq k)$ is true implies that ready times, when the $(k+1)^{th}$ task is mapped, are equal in the original mapping and the first iterative mapping.
Case 1: The $(k+1)^{th}$ task to be mapped in the original mapping is assigned to machine μ; because the tasks assigned to machine μ in the original mapping remain assigned to machine μ in the iterative mapping, then $P(k+1)$ for the first case is clearly true.
Case 2: The $(k+1)^{th}$ task, denoted t_{k+1}, to be mapped in the original mapping is not assigned to machine μ. Assume t_{k+1} is assigned to machine β_{k+1} ($\beta_{k+1} \in M$ and $\beta_{k+1} \neq \mu$) in the original mapping. Task t_{k+1} was assigned to its minimum completion time machine, this implies t_{k+1} on machine β_{k+1} is the smallest task-machine pair. The inductive step assumes the completion times of all tasks are the same, therefore the ready times, when t_{k+1} is mapped, of the original mapping and the iterative mapping are identical. Thus, the completion time of t_{k+1} on machine β_{k+1} is still the smallest completion time; therefore, t_{k+1} is assigned to machine β_{k+1} in the iterative mapping, and $P(k+1)$ is clearly true.

Generalization: The above can be generalized for iterations i and $i+1$ by considering i to be the original mapping and $i+1$ to be the first iterative mapping.

Example of Min-Min increasing makespan by breaking ties randomly

For this example, we can consider the initial ready times of machines to be 0. The ETC matrix used is shown in Table 1.

Original Mapping

The original mapping is shown in Table 2 and a visual representation is shown in Figure 3. The term "m_i CT" denotes the completion time of the task in the corresponding row for machine m_i. The following are the completion times after the original mapping: m_1: 5, m_2: 2, and m_3: 4. The makespan machine is m_1, therefore machine m_1 and task t_2 will not be considered for future iterations.

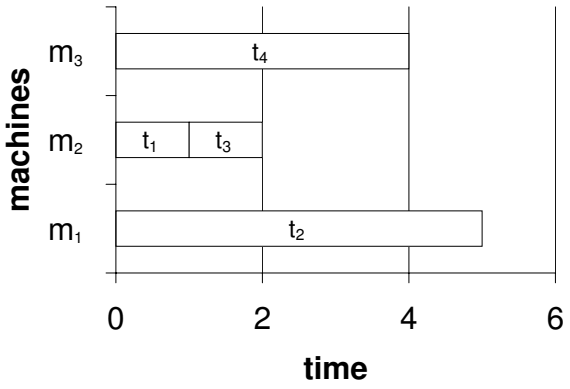

Figure 3. Original mapping for Min-Min example

	machines		
first resource allocation	m_1 CT	m_2 CT	m_3 CT
$t_1 \rightarrow m_2$	4	$\underline{1}$	3
t_2	5	6	7
t_3	3	$\underline{1}$	2
t_4	5	5	4
second resource allocation	m_1 CT	m_2 CT	m_3 CT
t_2	5	7	7
$t_3 \rightarrow m_2$	3	$\underline{2}$	$\underline{2}$
t_4	5	6	4
third resource allocation	m_1 CT	m_2 CT	m_3 CT
t_2	5	8	7
$t_4 \rightarrow m_3$	5	7	$\underline{4}$
fourth resouce allocation	m_1 CT	m_2 CT	m_3 CT
$t_2 \rightarrow m_1$	$\underline{5}$	8	11

Table 2. Original mapping for Min-Min example

First iterative mapping

The first iterative mapping shown in Table 3, considers the remaining tasks t_1,t_3, and t_4 and machines m_2 and m_3. A visual representation of the first iterative mapping is shown in Figure 4. In this example, t_3 may be assigned to m_3 because ties are broken randomly. In the original mapping we considered that this tie was broken by assigning t_3 to m_2. This change from the original mapping causes the makespan to increase. The final completion times are as follows: m_1: 5 (same as original mapping), m_2: 1, and m_3: 6. The new makespan machine is m_3.

This example proves that the makespan can increase if the Min-Min heuristic is used.

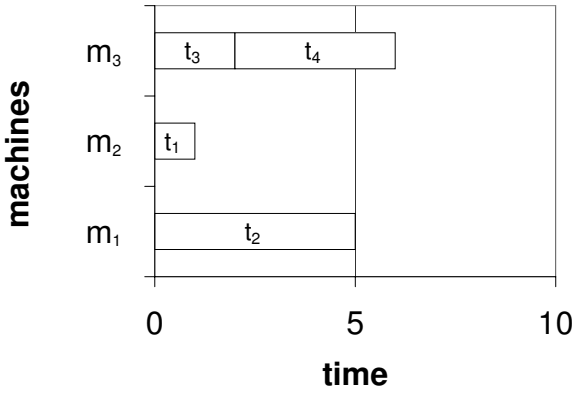

Figure 4. First iterative mapping for Min-Min example

	machines	
first resource allocation	m_2 CT	m_3 CT
$t_1 \rightarrow m_2$	$\underline{1}$	3
t_3	$\underline{1}$	2
t_4	5	4
second resource allocation	m_2 CT	m_3 CT
$t_3 \rightarrow m_3$	$\underline{2}$	$\underline{2}$
t_4	6	4
third resource allocation	m_2 CT	m_3 CT
$t_4 \rightarrow m_3$	$\underline{6}$	$\underline{6}$

Table 3. First iterative mapping for Min-Min example

3.3 Minimum Completion Time (MCT)

The procedure to implement the Minimum Completion Time (MCT) heuristic [3] is shown in Figure 5. With the iterative approach, the individual completion times for each machine do not improve over iterative mappings.

Theorem

If MCT is used as a mapping heuristic with the iterative approach and ties are broken deterministically, then the mappings produced by all the iterations are identical.

Proof

The initial ready times, for this proof, can be considered 0 without loss of generality. Let the makespan machine of the original mapping be machine μ. The iterative approach does not change the assignment of tasks that were assigned to machine μ in the original mapping; i.e., tasks assigned to the makespan machine in the original mapping are ignored in the first iterative mapping. The order of the task list is arbitrary but fixed between iterations.

1 A task list is generated that includes all unmapped tasks in a given arbitrary order.

2 The first task in the list is mapped to its minimum completion time machine (machine ready time plus estimated computation time of the task on that machine).

3 The task selected in step 2 is removed from the task list.

4 The ready time of the machine on which the task is mapped is updated.

5 Steps 2-4 are repeated until all the tasks have been mapped.

Figure 5. Procedure for MCT

Inductive hypothesis: Consider the n^{th} task mapped by MCT in the original mapping. Let $\underline{P(n)}$ be the statement that the n^{th} task will have the same completion time and be mapped to the same machine in both the original mapping and the first iterative mapping, if ties are broken deterministically. For the basis and inductive steps, there are two cases to consider: the case when the task is mapped to machine μ and the case when the task is not mapped to machine μ.

To prove $P(n)$ is true for $n = 1$ we need to prove:
1) $P(1)$ is true
2) $(\forall k)$ [$P(r)$ is true for all r, $1 \leq r \leq k \Rightarrow P(k+1)$ is true]

Basis Step: Prove that $P(1)$ is true.
Case 1: The first task is assigned to machine μ in the original mapping; For the first case, $P(1)$ is clearly true because the tasks assigned to machine μ in the original mapping remain assigned to machine μ in the iterative mapping.
Case 2: The first task, denoted t_1, to be mapped in the original mapping is not assigned to machine μ. In the original mapping, let t_1 be assigned to machine β_1 ($\beta_1 \in M$ and $\beta_1 \neq \mu$). The MCT heuristic uses a list to determine the order tasks are mapped. The first task in the list will not change from the original mapping to the iterative mapping. This implies t_1 is also the first task to be assigned in the iterative mapping. The minimum execution time machine (i.e., the minimum completion time machine) for task t_1 remains unchanged from the original mapping because the ETC values in the iterative mapping are identical to those in the original mapping. Therefore, $P(1)$ is clearly true.

Inductive Step: For the inductive step we assume $\forall r$

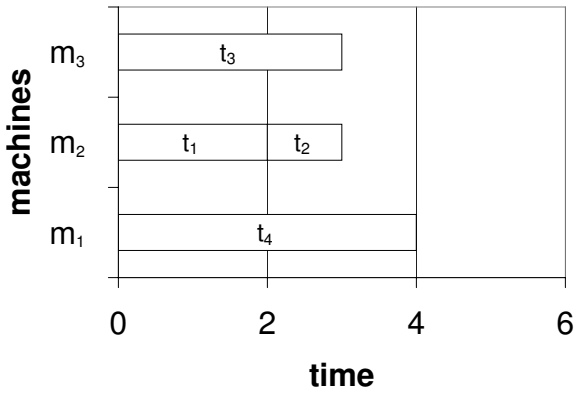

Figure 6. Original mapping for MCT example

$P(1 \leq r \leq k)$ is true and prove that $P(k+1)$ is true. The assumption that P($1 \leq r \leq k$) is true implies that the ready times, when the $(k+1)^{th}$ task is mapped, are equal in the original mapping and the iterative mapping.

Case 1: The $(k+1)^{th}$ task is assigned to machine μ in the original mapping. $P(k+1)$ for the first case is clearly true because the tasks assigned to machine μ in the original mapping remain assigned to machine μ in the iterative mapping .

Case 2: The $(k+1)^{th}$ task, denoted t^{k+1}, to be mapped in the original mapping is not assigned to machine μ. Assume t^{k+1} is assigned to machine β_{k+1} ($\beta_{k+1} \in M$ and $\beta_{k+1} \neq \mu$) in the original mapping. This implies that t_{k+1} on machine β_{k+1}, in the original mapping, has the minimum completion time. The inductive step assumes the ready times, when t_{k+1} is mapped, of the original mapping and the iterative mapping are identical. Thus, the completion time of t_{k+1} on machine β_{k+1} is identical in the original mapping and the iterative mapping; therefore, t_{k+1} is assigned to machine β_{k+1} in the iterative mapping, and $P(k+1)$ is clearly true.

Generalization: The above can be generalized for iterations i and $i+1$ by considering i to be the original mapping and $i+1$ to be the first iterative mapping.

Example of MCT increasing makespan by breaking ties randomly

For this example, we can consider the initial ready times to be 0. Consider the following mapping order for the MCT heuristic: t_1, t_2, t_3, and t_4. The ETC matrix used for this example is shown in Table 4.

Original Mapping

The example of MCT increasing the makespan is easier to demonstrate than the Min-Min case. This example relies on a tie in the mapping of task t_1 between m_2 and m_3. In the original mapping, t_1 is mapped to m_2. The resource allocations are shown in Table 5 and a visual representation is shown in Figure 6. The following are the completion times: m_1: 4, m_2: 3, and m_3: 3. The makespan machine is m_1.

	machines		
task	m_1	m_2	m_3
t_1	3	2	2
t_2	4	1	4
t_3	5	4	3
t_4	4	5	4

Table 4. ETC matrix for MCT example

	machines		
assignment	m_1 CT	m_2 CT	m_3 CT
$t_1 \rightarrow m_2$	3	$\underline{2}$	$\underline{2}$
$t_2 \rightarrow m_2$	4	$\underline{3}$	4
$t_3 \rightarrow m_3$	5	7	$\underline{3}$
$t_4 \rightarrow m_1$	$\underline{4}$	8	7

Table 5. Original mapping for MCT example

First iterative mapping

For the first iterative mapping, we assign task t_1 to m_3. This will cause the makespan machine to change. The resource allocations for the first iterative mapping are shown in Table 6 and a visual representation is shown in Figure 7. The following are the final completion times: m_1: 4 (same as original), m_2: 1, and m_3: 5. The new makespan machine is m_3.

3.4 Minimum Execution Time (MET)

The details of the Minimum Execution Time (MET) heuristic [3] are shown in Figure 8. The MET heuristic will not change its mapping from iteration to iteration. The following trivial proof shows that if ties are broken deterministically, e.g., using the machine with the lowest reference number, then the mapping will not change from iteration to iteration.

Proof MET mapping will not change from iteration to iteration

The statement we want to prove is that for any task $t \in T$ the original mapping and the iterative mapping are the same for non-makespan machines given that a tie between two or more machines (i.e., different machines have the same execution time) will always be broken in a deterministic way. The makespan machine of the original mapping is referred to as μ.

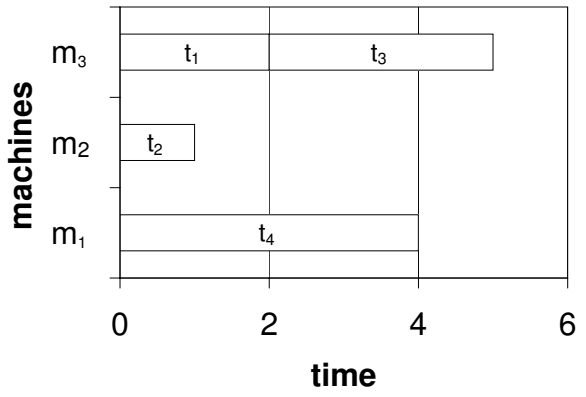

Figure 7. First iterative mapping for MCT example

assignment	machines m_2 CT	machines m_3 CT
$t_1 \rightarrow m_3$	$\underline{2}$	$\underline{1}$
$t_2 \rightarrow m_2$	$\underline{1}$	6
$t_3 \rightarrow m_3$	$\underline{5}$	$\underline{5}$

Table 6. First iterative mapping for MCT example

Case 1: Task *t* is assigned to the makespan machine (μ)
Original mapping:
Task *t* is assigned to machine μ
Iterative mapping:
Task *t* is not available for resource allocation, because it was already assigned to machine μ in the original mapping.
Case 2: Task *t* is not assigned to machine μ
Original mapping:
Task *t* is assigned to its MET machine β ($\beta \neq \mu$)
Iterative mapping:
The MET machine is dependent on the ETC values of *t* for all machines. This implies that *t* is assigned to its MET machine β because the ETC values do not change.

Example of MET increasing makespan by breaking ties randomly

For this example, we can consider the initial ready times to be 0. Consider the ETC matrix shown in Table 4 and the following mapping order: t_1, t_2, t_3, and t_4.

1 A task list is generated that includes all unmapped tasks in a given arbitrary order.

2 The first task in the list is mapped to its minimum execution time machine.

3 The task selected in step 2 is removed from the task list

4 Steps 2-3 are repeated until all tasks have been mapped.

Figure 8. Procedure for MET

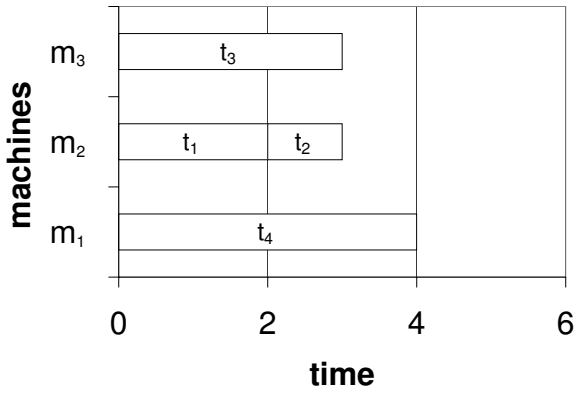

Figure 9. Original mapping for MET example

Original Mapping

The resource allocations of the original mapping are shown in Table 7 and a visual representation is shown in Figure 9. There are two MET machines for t_1. In the original mapping, task t_1 will be assigned to m_2. The following are the completion times after the MET heuristic finishes mapping: m_1: 4, m_2: 3, and m_3: 3. The makespan machine is m_1.

First iterative mapping

The makespan machine from the original mapping (m_1) and the task assigned to it (t_4) are not considered for the iterative mapping. In the iterative mapping, t_1 is mapped to m_3. This change causes the makespan to increase. The resource allocations are shown in Table 8 and a visual representation is shown in Figure 10. The completion times for the first iterative mapping are: m_1: 4 (same as original mapping), m_2: 1, and m_3: 5. The new makespan machine is m_3.

assignment	machines m_1 ETC	m_2 ETC	m_3 ETC
$t_1 \rightarrow m_2$	3	$\underline{2}$	$\underline{2}$
$t_2 \rightarrow m_2$	4	$\underline{1}$	4
$t_3 \rightarrow m_3$	5	4	$\underline{3}$
$t_4 \rightarrow m_1$	$\underline{4}$	5	$\underline{4}$

Table 7. Original mapping for MET example

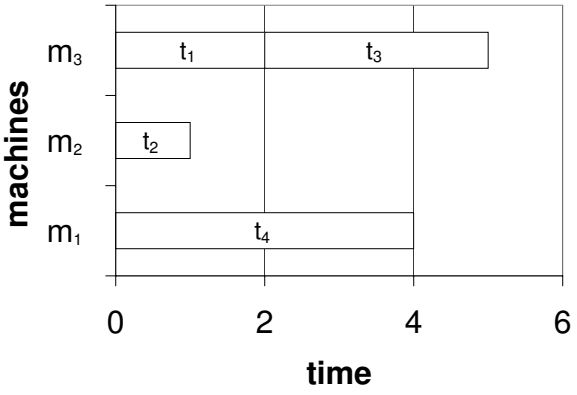

Figure 10. First iterative mapping for MET example

3.5 Switching Algorithm (SWA)

The Switching Algorithm (SWA) is adapted from [14]. It was designed for use in dynamic environments. The procedure for SWA is shown in Figure 13. The switching algorithm is a hybrid of the MET and MCT heuristics. An example of SWA increasing its makespan using the iterative approach can be found even for the case when ties are broken deterministically.

Example of SWA increasing makespan

Consider the ETC matrix shown in Table 9 and the following mapping order: t_1, t_2, t_3, t_4, and t_5. The intial ready times for all the machines is 0. SWA will switch from MCT to MET when the balance index (BI) is more than the high threshold of 0.49 and will switch from MET to MCT when the BI goes below the low threshold of 0.48.

Original Mapping

The original mapping is shown in Table 10 and a visual representation is shown in Figure 11. The completion times for the original mapping are: m_1: 6, m_2: 5, and m_3: 5. The makespan machine is m_1.

assignment	machines m_2 ETC	m_3 ETC
$t_1 \rightarrow m_3$	$\underline{2}$	$\underline{2}$
$t_2 \rightarrow m_2$	$\underline{1}$	4
$t_3 \rightarrow m_3$	4	$\underline{3}$

Table 8. First iterative mapping for MET example

task	machines m_1	m_2	m_3
t_1	6	10	12
t_2	1	2	4
t_3	5	2.5	4
t_4	6	3	2.5
t_5	4	2	1

Table 9. ETC matrix for SWA example

First Iterative Mapping

The first iterative mapping is shown in Table 11 and a visual representation is shown in Figure 12. The makespan machine (m_1) and the task assigned to it (t_1) are not considered for resource allocation in the first iterative mapping. Tasks t_2 and t_3 are assigned to the same machines in the original mapping and the first iterative mapping, however, t_4 is assigned to m_3 because the resource allocation of t_4 has a different BI. The completion times for the first iterative mapping are: m_1: 6, m_2: 4, and m_3: 6.5. The new makespan machine is m_3.

3.6 K-percent Best Algorithm

The K-percent Best Algorithm is adapted from [14]. The K-percent Best Algorithm, similar to the SWA, is a hybrid of MET and MCT. The procedure to implement the K-percent Best is shown in Figure 14. If the percentage is (100/number of machines)% then the K-percent Best is identical to the MET heuristic, however, if the percentage is 100% then it is identical to the MCT heuristic. Proof of the K-percent Best Algorithm increasing its makespan can be found even for the case when ties are broken deterministically.

Example of K-percent Best increasing makespan

For this example, we can consider the initial ready times of machines to be 0. Consider the ETC matrix shown in Table 12 and the following mapping order: t_1, t_2, t_3, t_4, and t_5. The percent, for this example, is set to 70%. This implies that for the original mapping the best two machines are used for mapping, and for the first iterative mapping only

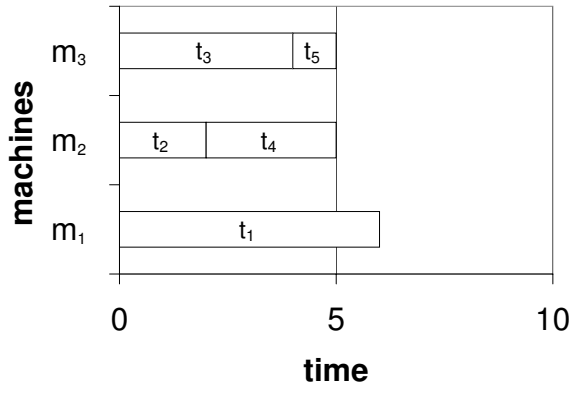

Figure 11. Original mapping for SWA example

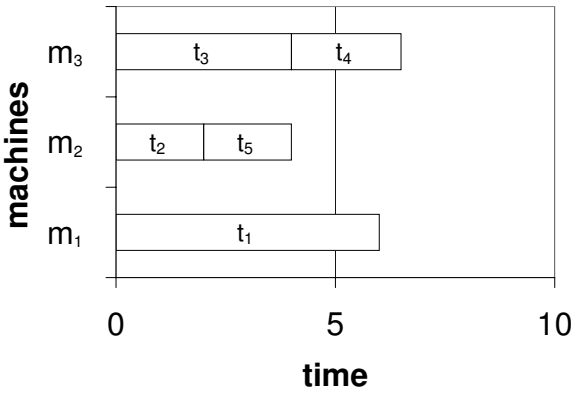

Figure 12. First iterative mapping for SWA example

BI	assignment	CT (m_1,m_2,m_3)	heuristic
x	$t_1 \rightarrow m_1$	6, 0, 0	MCT
0	$t_2 \rightarrow m_2$	6, 2, 0	MCT
0	$t_3 \rightarrow m_3$	6, 2, 4	MCT
$1/3$	$t_4 \rightarrow m_2$	6, 5, 4	MCT
$2/3$	$t_5 \rightarrow m_3$	6, 5, 5	MET

Table 10. Original mapping for SWA example

BI	assignment	CT (m_2,m_3)	heuristic
x	$t_2 \rightarrow m_2$	2, 0	MCT
0	$t_3 \rightarrow m_3$	2, 4	MCT
$1/2$	$t_4 \rightarrow m_3$	2, 6.5	MET
$4/13$	$t_5 \rightarrow m_2$	4, 6.5	MCT

Table 11. First iterative Mapping for SWA example

one machine is considered. This is the critical difference between the first iterative mapping and the original mapping. The option of only having one machine forces the K-percent Best Algorithm to perform like the MET heuristic in the first iterative mapping.

Original Mapping
The results of the original mapping are shown in Table 13 and a visual representation is shown in Figure 15. The K-percent Best machines are also shown. The completion times for the original mapping are: m_1: 6, m_2: 5, and m_3: 5.5. The makespan machine is m_1.

First Iterative Mapping
The makespan machine (m_1) and the task assigned to it (t_1) are not considered for resource allocation in the first iterative mapping. The results of the first iterative mapping are shown in Table 14 and a visual representation is shown in Figure 16. The completion times for the first iterative mapping are: m_1: 6, m_2: 7, and m_3: 3. The makespan machine after the first iteration becomes m_2. This example proves that for K-percent Best the makespan can increase if ties are broken deterministically.

3.7 Sufferage Algorithm

The Sufferage Algorithm (shown in Figure 17) is adapted from [4, 14]. The sufferage algorithm is a greedy algorithm that does a limited local search. The proof of the sufferage algorithm increasing makespan with the iterative approach can be found even for the case when ties are broken deterministically.

The example, however, is considerably more complex than the examples provided for K-percent Best and SWA.

Example of sufferage increasing makespan

For this example, we can consider the initial ready times of machines to be 0. Consider the ETC matrix shown in Table 15.

Original Mapping
The original mapping is shown in Table 16 and a visual representation is shown in Figure 18. The completion times of the original mapping using the Sufferage Algorithm are as follows: m_1: 10, m_2: 9.5, and m_3: 9.5. The makespan machine is m_1.

1 A task list is generated that includes all unmapped tasks in a given arbitrary order.

2 The first task in the list is mapped using the MCT heuristic.

3 The load balance index is calculated for the system (min. ready time / max. ready time).

4 The heuristic used to map the task is determined as follows:

- i If the load balance index $>$ high threshold, the MET heuristic is selected to map future tasks.
- ii If the load balance index $<$ low threshold, the MCT heuristic is selected to map future tasks.
- iii Otherwise, the current heuristic remains selected.

5 Steps 3-4 are repeated until all tasks have been mapped.

Figure 13. Description of SWA

	machines		
task	m_1	m_2	m_3
t_1	6	10	12
t_2	1	2	4
t_3	2	4	3
t_4	5	3	4
t_5	6	2	2.5

Table 12. ETC matrix for the K-percent Best example

1 A task list is generated that includes all unmapped tasks in a given arbitrary order.

2 A subset is formed by picking the $M \cdot (\frac{k}{100})$ best machines based on the execution times for the task.

3 The task is assigned to a machine that provides the earliest completion time in the subset.

4 The task is removed from the unmapped task list.

5 The ready time of the machine on which the task is mapped is updated.

6 Steps 2-5 are repeated until all tasks have been mapped.

Figure 14. Procedure for K-percent Best

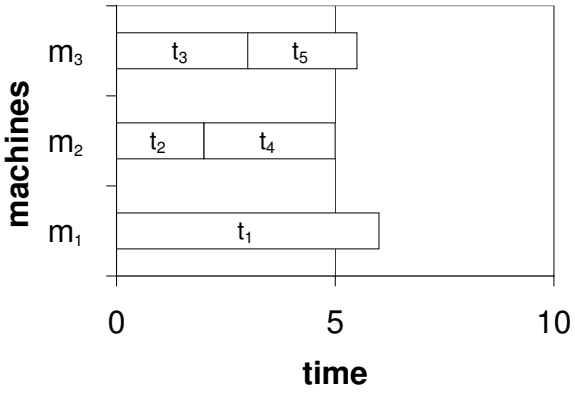

Figure 15. Original mapping for K-percent Best example

First Iterative Mapping

The first iterative mapping is shown in Table 17 and a visual representation is shown in Figure 19. The completion times of the first iterative mapping using the Sufferage Algorithm are as follows: m_1: 10, m_2: 10.5, and m_3: 8.5. The new makespan machine is m_2. This example shows that the makespan can increase when using the iterative approach.

4 Related Work

Makespan is often the performance feature to be optimized in the study of resource allocation in a heterogeneous computing system. Many studies explore different methods of reducing the makespan of the given set of tasks. The literature was examined to select a set of heuristics appropriate for the HC environment considered in this study. The MET, MCT and Min-Min heuristics implemented here are adapted from [8]. The K-percent Best and Switching Algorithm were adapted from [14] and the Sufferage Algorithm was adapted from [14]. The variation of Genitor implemented here is an adaptation of the Genitor heuristic introduced in [17].

In the static environment studied in [3], heuristics were used to minimize makespan. The goal of [3] was to reduce the makespan in a static environment. The goal of this study is to investigate whether the iterative procedure can reduce the finishing time of some machines compared to using only the original mapping. Our goal is clearly different from the goal of [3] because the reduction of makespan does not imply a reduction in the completion time of all machines. The work in [14] reduces the makespan of a set of tasks in a

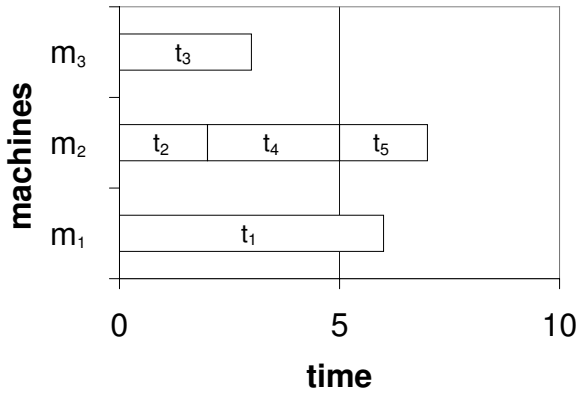

Figure 16. First iterative mapping for K-percent Best example

dynamic mapping environment, i.e., the arrival times of the tasks are not known *a priori*. The study in [8] attempts to minimize the makespan of a set of tasks on a heterogeneous multiprocessor system.

5 Conclusions

An iterative approach for minimizing the finishing times of machines in a heterogeneous computing environment was proposed. The performance of several heuristics was analyzed for such an approach. The greedy heuristics studied (Min-Min, MCT, MET, SWA, K-percent Best, and Sufferage) did not guarantee an improvement in the completion time among remaining machines. The mappings generated by MET, MCT, and Min-Min heuristics were proven to not change over successive iterations, if ties are broken deterministically. If ties are broken randomly, the makespan of MET, MCT and Min-Min can actually increase. When the iterative approach is used, the Switching Algorithm, K-percent Best and Sufferage heuristics all produced mappings that could increase their makespan when ties are broken deterministically. The Genitor-based approach will keep the same mapping or produce a better mapping because the best solution is preserved from one iteration (of the iterative approach) to the next iteration. Implementing a form of "seeding" similar to Genitor's seeding to other heuristics would guarantee that a heuristic can never increase makespan from one iteration to the next. This would cause the best solutions to be preserved across iterations, thus changing the mapping only if better mapping is found.

Acknowledgments: The authors thank Jay Smith and David Janovy for their valuable comments.

1 A task list (L) is generated that includes all unmapped tasks in a given arbitrary order.

2 While there are still unmapped tasks:

- i Mark all machines as unassigned.
- ii For each task $t_k \in L$.
  - a The machine m_j that gives the earliest completion time is found.
  - b The Sufferage value is calculated. (Sufferage value = second earliest completion time minus earliest completion time).
  - c If machine m_j is unassigned then assign t_k to machine m_j, delete t_k from L, and mark m_j as assigned. Otherwise, if the sufferage value of the task (t_i) already assigned to m_j is less than the sufferage value of task t_k then unassign t_i, add t_i back to L, assign t_k to machine m_j, and remove t_k from L.
- iii The ready times for all machines are updated.

Figure 17. Procedure for Sufferage

assignment	CT (m_1,m_2,m_3)	K-% machines
$t_1 \rightarrow m_1$	6, 0, 0	m_1,m_2
$t_2 \rightarrow m_2$	6, 2, 0	m_2,m_3
$t_3 \rightarrow m_3$	6, 2, 3	m_1,m_3
$t_4 \rightarrow m_2$	6, 5, 3	m_2,m_3
$t_5 \rightarrow m_3$	6, 5, 5.5	m_2,m_3

Table 13. Original mapping for K-percent Best example

assignment	CT (m_2,m_3)	K-% machines
$t_2 \rightarrow m_2$	2, 0	m_2
$t_3 \rightarrow m_3$	2, 3	m_3
$t_4 \rightarrow m_2$	5, 3	m_2
$t_5 \rightarrow m_2$	7, 3	m_2

Table 14. First iterative mapping for K-percent Best example

References

[1] S. Ali, T. D. Braun, H. J. Siegel, A. A. Maciejewski, N. Beck, L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao. Characterizing re-

task	machines m_1	m_2	m_3
t_0	4.5	3.5	3.5
t_1	1	3	4
t_2	3	5	3
t_3	4	3	5
t_4	4.5	2.5	2.5
t_5	4	2	4
t_6	3	5	4
t_7	5	4	1
t_8	4	4	2
t_9	4	5	5
t_{10}	2	3	3
t_{11}	2	2	2

Table 15. ETC matrix for Sufferage example

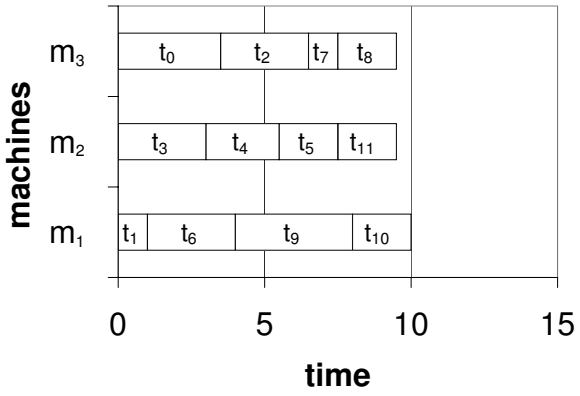

Figure 18. Original mapping for Sufferage example

1st pass	min. CT	Sufferage	machine
$t_1 \rightarrow m_1$	1	2	m_1
$t_5 \rightarrow m_2$	2	2	m_2
$t_7 \rightarrow m_3$	1	3	m_3
2nd pass	min. CT	Sufferage	machine
$t_6 \rightarrow m_1$	4	1	m_1
$t_8 \rightarrow m_3$	3	2	m_3
3rd pass	min. CT	Sufferage	machine
$t_2 \rightarrow m_3$	6	1	m_3
$t_3 \rightarrow m_2$	5	3	m_2
4th pass	min. CT	Sufferage	machine
$t_4 \rightarrow m_2$	7.5	1	m_2
$t_9 \rightarrow m_1$	9	2	m_1
5th pass	min. CT	Sufferage	machine
$t_0 \rightarrow m_3$	9.5	1.5	m_3
6th pass	min. CT	Sufferage	machine
$t_{10} \rightarrow m_1$	10	0.5	m_1
$t_{11} \rightarrow m_2$	9.5	0.5	m_2

Table 16. Original mapping for Sufferage example

source allocation heuristics for heterogeneous computing systems. In *Advances in Computers Volume 63: Parallel, Distributed, and Pervasive Computing*, pages 91–128, 2005.

[2] L. Barbulescu, L. D. Whitley, and A. E. Howe. Leap before you look: An effective strategy in an oversubscribed scheduling problem. In *19th National Conference on Artificial Intelligence*, pages 143–148, July 2004.

[3] T. D. Braun, H. J. Siegel, N. Beck, L. Boloni, R. F. Freund, D. Hensgen, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao. 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(6):810–837, June 2001.

[4] H. Casanova, A. Legrand, and F. Berman. Heuristics for scheduling parameter sweep applications in grid environments. In *9th IEEE Heterogeneous Computing Workshop (HCW 2000)*, pages 349–363, Mar. 2000.

[5] A. Dogan and F. Ozguner. Genetic algorithm based scheduling of meta-tasks with stochastic execution times in heterogeneous computing systems. *Cluster Computing*, 7(2):177–190, Apr. 2004.

[6] R. F. Freund and H. J. Siegel. Heterogeneous processing. *IEEE Computer*, 26(6):13–17, June 1993.

[7] A. Ghafoor and J. Yang. A distributed heterogeneous supercomputing management system. *IEEE Computer*, 26(6):78–86, June 1993.

[8] O. H. Ibarra and C. E. Kim. Heuristic algorithms for scheduling independent tasks on non-identical processors. *Journal of the ACM*, 24(2):280–289, Apr. 1977.

[9] M. Kafil and I. Ahmad. Optimal task assignment in heterogeneous distributed computing systems. *IEEE Concurrency*, 6(3):42–51, July 1998.

[10] A. Khokhar, V. K. Prasanna, M. E. Shaaban, and C. Wang. Heterogeneous computing: Challenges and opportunities. *IEEE Computer*, 26(6):18–27, June 1993.

[11] A. Kumar and R. Shorey. Performance analysis and scheduling of stochastic fork-join jobs in a multicomputer system. *IEEE Transactions on Parallel and Distributed Systems*, 4(10):1147–1164, Oct. 1993.

[12] C. Leangsuksun, J. Potter, and S. Scott. Dynamic task mapping algorithms for a distributed heterogeneous computing environment. In *4th IEEE Heterogeneous Computing Workshop (HCW 1995)*, pages 30–34, Apr. 1995.

[13] M. Maheswaran, T. D. Braun, and H. J. Siegel. Heterogeneous distributed computing. In J. G. Webster, editor, *Encyclopedia of Electrical and Electronics Engineering*, volume 8, pages 679–690. John Wiley, New York, NY, 1999.

[14] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. *Journal of Parallel and Distributed Computing*, 59(2):107–121, Nov. 1999.

1st pass	min. CT	Sufferage	machine
$t_3 \rightarrow m_2$	3	2	m_2
$t_7 \rightarrow m_3$	1	3	m_3
2nd pass	min. CT	Sufferage	machine
$t_2 \rightarrow m_3$	4	4	m_3
$t_5 \rightarrow m_2$	5	0	m_2
3rd pass	min. CT	Sufferage	machine
$t_8 \rightarrow m_3$	6	3	m_3
4th pass	min. CT	Sufferage	machine
$t_0 \rightarrow m_2$	8.5	1	m_2
5th pass	min. CT	Sufferage	machine
$t_4 \rightarrow m_3$	8.5	2.5	m_3
6th pass	min. CT	Sufferage	machine
$t_{11} \rightarrow m_2$	10.5	0	m_2

Table 17. First iterative mapping for Sufferage example

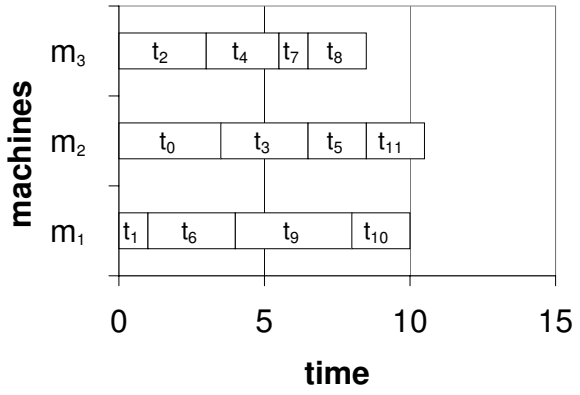

Figure 19. First iterative mapping for Sufferage example

[15] V. Shestak, E. K. P. Chong, A. A. Maciejewski, H. J. Siegel, L. Bemohamed, I. Wang, and R. Daley. Resource allocation for periodic applications in a shipboard environment. In *14th IEEE Heterogeneous Computing Workshop (HCW 2005)*, Apr. 2005.

[16] H. Singh and A. Youssef. Mapping and scheduling heterogeneous task graphs using genetic algorithms. In *5th IEEE Heterogeneous Computing Workshop (HCW 1996)*, pages 86–97, Apr. 1996.

[17] D. Whitley. The genitor algorithm and selective pressure: Why rank based allocation of reproductive trials is best. In *3rd International Conference on Genetic Algorithms*, pages 116–121, June 1989.

[18] M. Wu and W. Shu. Segmented min-min: A static mapping algorithm for meta-tasks on heterogeneous computing systems. In *9th IEEE Heterogeneous Computing Workshop (HCW 2000)*, pages 375–385, Mar. 2000.

[19] D. Xu, K. Nahrstedt, and D. Wichadakul. Qos and contention-aware multi-resource reservation. *Cluster Computing*, 4(2):95–107, Apr. 2001.

[20] J. Yang, I. Ahmad, and A. Ghafoor. Estimation of execution times on heterogeneous supercomputer architectures. In *International Conference on Parallel Processing*, volume I, pages 219–225, Aug. 1993.

Biographies

Luis D. Briceño is currently pursuing his Ph.D. degree in Electrical and Computer Engineering at Colorado State University. He obtained his B.S. degree in electrical and electronic engineering from the University of Costa Rica. His research interests include heterogeneous parallel and distributed computing.

Mohana Oltikar completed her M.S. degree in Electrical and Computer Engineering at Colorado State University in 2006, where she worked as a Graduate Assistant. She received her bachelor's degree in electronics engineering from the University of Mumbai, India. She is currently working as a digital design engineer in Hughes Network Systems.

H. J. Siegel holds the endowed chair position of Abell Distinguished Professor of Electrical and Computer Engineering at Colorado State University (CSU), where he is also a Professor of Computer Science. He is the Director of the CSU Information Science and Technology Center (ISTeC). ISTeC a university-wide organization for promoting, facilitating, and enhancing CSU's research, education, and outreach activities pertaining to the design and innovative application of computer, communication, and information systems. Prof. Siegel is a Fellow of the IEEE and a Fellow of the ACM. From 1976 to 2001, he was a professor in the School of Electrical and Computer Engineering at Purdue University. He received a B.S. degree in electrical engineering and a B.S. degree in management from the Massachusetts Institute of Technology (MIT), and the M.A., M.S.E., and Ph.D. degrees from the Department of Electrical Engineering and Computer Science at Princeton University. He has co-authored over 330 technical papers. His research interests include heterogeneous parallel and distributed computing, parallel algorithms, parallel machine interconnection networks, and reconfigurable parallel computer systems. He was a Coeditor-in-Chief of the *Journal of Parallel and Distributed Computing*, and has been on the Editorial Boards of both the *IEEE Transactions on Parallel and Distributed Systems* and the *IEEE Transactions on Computers*. He was Program Chair/Co-Chair of three major international conferences, General Chair/Co-Chair of six international conferences, and Chair/Co-Chair of five workshops. He is a member of the Eta Kappa Nu electrical engineering honor society, the Sigma Xi science honor society, and the Upsilon Pi Epsilon computing sci-

ences honor society. He has been an international keynote speaker and tutorial lecturer, and has consulted for industry and government. For more information, please see www.engr.colostate.edu/∼hj.

Anthony A. Maciejewski received the B.S.E.E, M.S., and Ph.D. degrees in Electrical Engineering in 1982, 1984, and 1987, respectively, all from The Ohio State University under the support of an NSF graduate fellowship. From 1985 to 1986 he was an American Electronics Association Japan Research Fellow at the Hitachi Central Research Laboratory in Tokyo, Japan where he performed work on the development of parallel processing algorithms for computer graphic imaging. From 1988 to 2001, he was a Professor of Electrical and Computer Engineering at Purdue University. In 2001, he joined Colorado State University where he is currently the Head of the Department of Electrical and Computer Engineering. Prof. Maciejewskis primary research interests relate to the analysis, simulation, and control of robotic systems and he has co-authored over 150 published technical articles in these areas. He has served in the editorial board of the *IEEE Transactions on Robotics and Automation*, *IEEE Transactions on Systems, Man, and Cybernetics*, *Pattern Analysis and Applications*, *Intelligent Automation and Soft Computing*, and was co-guest editor for a special issue on Kinematically Redundant Robots for the *Journal of Intelligent and Robotic Systems*. He is a fellow of IEEE, served as VP for Finance and on the IEEE Administrative Committee for the Robotics and Automation Society and was the Program Co-Chair (1997) and Chair (2002) for the International Conference on Robotics and Automation, as well as serving as the Chair and on the Program Committee for numerous other conferences. *An up-to-date vita is available at www.engr.colostate.edu/∼aam*.