Processor Allocation for Tasks that is Robust Against Errors in Computation Time Estimates

Prasanna V. Sugavanam[1], H. J. Siegel[1,2], Anthony A. Maciejewski[1], Syed Amjad Ali[1], Mohammad Al-Otaibi[1], Mahir Aydin[1], Kumara Guru[1], Aaron Horiuchi[3], Yogish G. Krishnamurthy[2], Panho Lee[1], Ashish Mehta[1], Mohana Oltikar[1], Ron Pichel[3], Alan J. Pippin[3], Michael Raskey[4], Vladimir Shestak[1], and Junxing Zhang[5]

Colorado State University
[1]Department of Electrical & Computer Engineering
[2]Department of Computer Science
Fort Collins, CO 80523-1373
{prasanna, hj, aam, sdamjad, motaibi, mahir, higuru, yogi, leepanho, ammehta, mohana, shestak}@colostate.edu

Hewlett-Packard Company
[3]Systems & VLSI Technology Division
[4]Linux & Open Source Lab
Fort Collins, CO 80528
{akh, rgp, ajp}@fc.hp.com
michael.raskey@hp.com

[5]University of Utah
School of Computing
Salt Lake City, UT 84112
junxing@cs.utah.edu

Abstract

Heterogeneous computing systems composed of interconnected machines with varied computational capabilities often operate in environments where there may be sudden machine failures, higher than expected load, or inaccuracies in estimation of system parameters. Makespan (defined as the completion time for an entire set of tasks) is often the performance feature that is optimized in such systems. It is important that the makespan of a resource allocation (mapping) be robust against errors in task computation time estimates. The problem of optimally mapping tasks onto machines of a heterogeneous computing environment has been shown, in general, to be NP-complete. Therefore, heuristic techniques to find near optimal solutions to this mapping problem are required. The goal of this research is to find a static mapping of tasks so that the robustness of the desired system feature, makespan, is maximized against the errors in task execution time estimates. Seven heuristics to derive near-optimal solutions and an upper bound to this problem are presented and evaluated.

This research was supported by the Colorado State University Center for Robustness in Computer Systems (funded by the Colorado Commission on Higher Education Technology Advancement Group through the Colorado Institute of Technology), and by the Colorado State University George T. Abell Endowment.

1. Introduction and Problem Statement

Heterogeneous computing (HC) systems utilize various resources with different capabilities to satisfy the requirements of diverse task mixtures and to maximize the system performance (e.g., [11, 18]). Such systems often operate in an environment where certain desired performance features degrade due to unpredictable circumstances such as sudden machine failures, higher than expected load, or inaccuracies in the estimation of system parameters (e.g., [3, 4, 9, 24, 25, 33]). Thus, it becomes necessary to allocate resources to tasks to maximize the robustness of the allocation. This study focuses on this aspect of resource allocation.

The act of assigning (matching) each task to a machine and ordering (scheduling) the execution of the tasks on each machine is known as mapping, resource allocation, or resource management. An important research problem is how to determine a mapping so as to maximize the robustness of desired system features against perturbations in system parameters [4]. In both cases, the general problem of optimally mapping tasks to machines in an HC environment has been shown to be NP-complete (e.g., [13, 19, 21]). Thus, the development of heuristic techniques to find near-optimal solutions for the mapping problem is an active area of research (e.g., [1, 2, 7, 8, 18, 20, 28, 41]).

Static mapping is performed when the applications are mapped in an off-line planning phase such as in a production environment (e.g., [1, 10, 12, 35, 36, 39]). Static mapping techniques take a fixed set of applications, a fixed set of machines, and generate a mapping. These heuristics determine a mapping off-line, and must use estimated values of task computation times. These techniques may be used to plan the execution of a set of tasks for a future time period (e.g., the production tasks to execute on the following day). Static mapping is also used for "what-if" predictive studies. For example, a system administrator might want to know the benefits of adding a new machine to the HC suite to justify purchasing it. By conducting a static mapping and then deriving the estimated system performance for the set of applications, the impact of the new machine can be approximated.

In this research, a metatask composed of a number of independent tasks (i.e., no communication between tasks are needed) is considered. Makespan is defined as the completion time for the entire metatask. A mapping is defined to be robust with respect to specified system performance features against perturbations in specified system parameters if degradation in these features is limited when certain perturbations occur [4]. The degree of robustness is the maximum amount of collective uncertainty in perturbed system parameters within which a user-specified level of system performance can be guaranteed. In this system, it is required that the makespan be robust against errors in task execution time estimates. Specifically, the system is considered robust if the actual makespan under the perturbed conditions does not exceed the required time constraint. The goal of this study is to find a static mapping of all tasks to machines so that the robustness of the mapping is maximized; i.e., to maximize the collective allowable error in execution time estimation for the tasks that can occur without the makespan exceeding the constraint.

A description of the system model is now given. A set of $\underline{T}$ tasks in the metatask is required to be allocated to a set of $\underline{M}$ machines. Each machine executes a single task at a time (i.e., no multitasking), in the order in which the tasks are assigned. The estimated time to compute (ETC) value for each task on each machine is assumed to be known *a priori*. This assumption is commonly made while studying mapping heuristics (e.g., [23]). Approaches for doing this estimation are discussed in [29]. Assume that unknown inaccuracies in the ETC values are expected. Hence, it is required that the mapping $\underline{\mu}$ must be robust against them. Specifically, the actual makespan of the mapping (calculated considering the effects of ETC errors) must be less than $\underline{\tau}$.

The FePIA procedure that was developed in [4] is applied to determine the robustness metric for this problem. Let $\underline{C_i^{est}}$ be the ETC value for task i on the machine where it is allocated. Let $\underline{C_i}$ be equal to the actual computation time for task i. The finishing time of a given machine j, $\underline{F_j}$, depends only on the actual computation times of the tasks mapped to that machine. Additionally, let $\underline{C^{est}}$ be the vector of the C_i^{est} values such that $C^{est} = \left[C_0^{est}\ C_1^{est} \ldots C_{|T|-1}^{est} \right]$. Similarly, let $\underline{C}$ be the actual computation time vector such that $C = \left[C_0\ C_1 \ldots C_{|T|-1} \right]$. The performance feature ($\underline{\phi}$) that should be limited in variation to ensure the makespan robustness is the finishing times of the machines. That is, $\phi = \left\{ F_j \mid 1 \le j \le |M| \right\}$. The vector C is the perturbation parameter for this study and F_j is a function of C. To be robust (i.e., meet the time constraint τ), it is required that $F_j(C) \le \tau$ for all j. That is, $F_j(C) = \tau$ is the maximum allowed value for any F_j.

The robustness radius of F_j against C for mapping μ, $\underline{r_\mu(F_j, C)}$, is defined as the Euclidean distance that C can change from the assumed value of C^{est} without the finishing time of machine j exceeding the tolerable variation. Mathematically,

$$r_\mu(F_j, C) = \min_{C:\, F_j(C) = \tau} \left\| C - C^{est} \right\|_2. \quad (1)$$

That is, if the Euclidean distance between any vector of actual computation times and the vector of estimated computation times is no larger than $r_\mu(F_j, C)$, then the finishing time of the machine j will be at most the makespan constraint τ. As described in [4], equation (1) can be interpreted as the perpendicular distance from C^{est} to the hyperplane described by the equation $\tau - F_j(C^{est}) = 0$. Hence, equation (1) can be rewritten as [34],

$$r_\mu(F_j, C) = \frac{\left(\tau - F_j\left(C^{est} \right) \right)}{\sqrt{\text{number of tasks mapped to machine } j}}. \quad (2)$$

The robustness metric, $\underline{\rho_\mu(\phi, C)}$, for the mapping is simply the minimum of all robustness radii over all machines [4]. If the Euclidean distance between any vector of the actual execution times and the vector of the estimated execution times is no larger than $\rho_\mu(\phi, C)$, then the actual makespan will be at most the constraint τ. Mathematically,

$$\rho_\mu(\phi, C) = \min_{F_j \in \phi} r_\mu(F_j, C). \quad (3)$$

The performance metric that is used to evaluate the mapping is $\rho_\mu(\phi, C)$. It is obvious that the larger the robustness metric, the better the mapping.

The goal for this study is to map all tasks to machines such that the makespan for the entire metatask is within the time constraint τ while maximizing $\rho_\mu(\phi, C)$. Seven static mapping schemes are studied in this paper: Max-Max, Greedy Iterative Maximization, Overhead Iterative Maximization, GENITOR, Memetic Algorithm, Ant Colony Optimization, and Hereboy Evolutionary Algorithm. The wall clock time for the mapper itself to execute is arbitrarily required to be less than or equal to 60 minutes on a typical unloaded 3GHz Intel Pentium 4 machine. Simulations are used to evaluate and compare the seven static heuristics studied in this paper.

The remainder of this paper is organized in the following manner. The next section describes the simulation setup used for this research. Section 3 provides literature related to this work. In Section 4, the heuristics studied in this research and an upper bound are presented. Section 5 discusses the results, and the last section gives a brief summary of this research work.

2. Simulation Setup

An HC system with eight machines and 1024 independent tasks is simulated in this study. This large number of tasks is chosen to present a significant mapping challenge for each heuristic.

The estimated execution times of all tasks taking heterogeneity into consideration are generated using the gamma distribution method described in [5]. Two different cases of ETC heterogeneities are used in this research, the high task and high machine heterogeneity (<u>high-high</u>) case and the low task and low machine heterogeneity (<u>low-low</u>) case. For both the cases, the ETCs are of "inconsistent" type [5]. The estimated execution time of task i on machine j is given by <u>*ETC*(*i*, *j*)</u>. For this study, a total of 100 trials (50 trails for each of the cases) are run, where each trial corresponds to a different ETC matrix.

A task mean and coefficient of variation (<u>COV</u>) are used to generate the ETC matrices. The high-high case uses a mean task execution time of 30 seconds and a COV of 0.9 (task heterogeneity) to calculate the values for all the elements in a task vector (where the number of elements is equal to the number of tasks). Then using the i^{th} element of the vector as the mean and a COV of 0.9 (machine heterogeneity), the ETC values for task i on all the machines are calculated. The low-low heterogeneity case uses a mean task execution time of 30 seconds and a COV of 0.3 for task heterogeneity and 0.3 for machine heterogeneity.

The value of the time constraint τ is chosen to be 5000 seconds so that it presents a feasible mapping problem for the heuristics to solve. A simple greedy mapping heuristic that minimized the makespan was used to determine the value of τ. The performance of each heuristic is studied across all 100 different scenarios, where each scenario is an ETC matrix of either the high-high or low-low case.

3. Related Work

The work presented in this paper is built upon the four step FePIA procedure detailed in [4]. The FePIA procedure describes a way to derive a generalized robustness metric and it is applied for the problem studied here. In the literature, a number of papers have studied the issue of robustness in distributed systems (e.g., [9, 14, 15, 17, 25, 27, 37]). Robust decision making formulations presented in [14, 24, 25] motivate building a robust suboptimal solution over a better performing solution that is less robust.

In [9], given an allocation for an augmented dependency graph, an analytic measure of the vulnerability of the allocation to hazards (uncertainties in estimated execution times of tasks) is devised. They introduced the concept of critical component in the execution path based on the spare time and slack. Their robustness metric is problem specific and cannot be applied to our system.

The research in [14] considers a single machine scheduling environment where the processing times of individual jobs are uncertain. Given the probabilistic information about processing time for each job, the authors in [14] determine the normal distribution that approximates the flow time associated with a given schedule. The risk value is calculated by using the approximate distribution of flow time. The robustness of a given schedule is then given by 1 minus the risk of achieving substandard flow time performance. In our work, no such stochastic specification of the uncertainties is assumed. Furthermore, our environment involves multiple machines.

The study in [15] explores slack-based techniques for producing robust resource allocations in a job-shop environment. The central idea is to provide each task with extra time (defined as slack) to execute so that some level of uncertainty can be tolerated without having to reallocate. The study uses slack as its measure of robustness, which is simpler and different from our measure.

The research in [17] considers reactive scheduling to unexpected events that may cause a constraint violation in a shop floor environment. They define repair steps if a job takes longer than expected so that the new evaluation of constraint would be as good as or better than the old evaluation. Our work explores robust resource allocation techniques to maximize the cumulative errors in ETCs so that the specified performance is guaranteed in a heterogeneous computing environment; thus, our problem differs in many ways from scheduling machines in a shop.

In [24], the authors assume a scenario based approach to represent the input data uncertainty to their robustness decision model job shop environments. In their robust decision making framework, they present three critical elements: (a) use of scenario planning approach to structure data uncertainty for the decision situation; (b) choice of appropriate robustness criterion; and (c) the formal development of a decision model. Our work differs from the types discussed in [24] because our environment is heterogeneous computing and no mathematical characterization of the possible uncertainties in ETC values is assumed.

The study in [25], considers a two-stage flow shop where the processing times of individual jobs are uncertain. The authors structure the uncertainty in two ways: the first approach assumes a set of discrete processing time scenarios, each of which specifies the processing time for each job on each machine under that scenario; the second approach assumes a set of independent processing time intervals, which define the minimum and maximum processing time that can be realized for each job on each machine. Our work differs from [25] for reasons similar to those stated for [24].

The work in [27] develops a mathematical definition for the robustness against machine breakdowns in a job-shop environment. The authors assume a certain random distribution of the machine breakdowns and a certain rescheduling policy in the event of breakdowns. In our paper, we consider uncertainties in ETCs of tasks for heterogeneous computing.

In [37], it is attempted to calculate the stability radius of an optimal schedule in a job-shop environment. The work in [37] is specific to their environment, but our work is based on [4] because it is more general and considers system requirements to generate the robustness metric. In our paper, heuristic approaches are explored to allocate resources in a robust manner based on the metric derived using the work done in [4].

The literature was examined to select a set of heuristics appropriate for the HC environment considered here. The Max-Max is a variation of the Min-Min that has proven to be a good heuristic for static and dynamic mapping problems (e.g., [12, 21, 41]). The Iterative Maximization (IM) techniques are a variation of the iterative deepening and random search techniques used in [16]. The GENITOR-style genetic algorithm used here is an adaptation of [40]. GENITOR is a steady-state genetic algorithm (GA) that has been shown to work well for several problem domains, including resource allocation, and job shop scheduling and hence, chosen for this problem. Memetic Algorithm (MA) [6, 30, 31], also called the hybrid GA, applies a separate local search process (hill-climbing) to refine individuals. Combining global and local search is a strategy used by many successful global optimization approaches [6]. The Ant Colony Optimization (ACO) [31, 32, 38] metaheuristic has been used previously to map tasks onto heterogeneous machines, like Max-Max and GENITOR. In [32], the performance of ACO is compared to some of the heuristics described in [12]. The authors proved that the ACO heuristic perform well for a similar problem setup given in [12] although it is very time consuming to build good solutions. The heuristic formulation of ACO used here is a variation of [32]. The HereBoy Evolutionary Algorithm used here is a combination of GA and Simulated Annealing (SA) and is an adaptation of [26] that was applied to the evolvable hardware problem. This fast evolutionary algorithm is shown to be well suited for exploring large spaces and can be applied to a wide range of optimization problems.

4. Heuristics Descriptions

Both of the Iterative Maximization (IM) heuristics start with an initial solution and try to improve the solution by "local" modifications similar to the iterative improvement techniques used in [16]. The initial solution can be constructed randomly or by using some heuristic method. To modify a mapping, a number of different techniques can be applied, e.g., exchange of machine assignments of two tasks, the move of a task from one machine to another machine. The choice of how to modify a mapping can be made randomly or with some "constructive" criteria-based method. The Greedy Iterative Maximization (GIM) and Overhead Iterative Maximization (OIM) heuristics predominantly use the following two procedures during an iteration: reassignment and swapping. In reassignment, a task is picked from a machine i arbitrarily or procedurally, and moved to the target machine j such that the robustness metric of the mapping is improved. In swapping, a task is chosen from a machine i and another task is chosen from a different machine j, both tasks are chosen arbitrarily or procedurally, and then swapped such that the robustness of the mapping is improved.

In the IM heuristics and in Ant Colony Optimization (ACO), the term min-radius machine is the machine that determines the robustness metric of the mapping, that is, the one that has the minimum robustness radius over all machines. Execution of the reassignment procedure followed by swapping was used in both the IM heuristics because it yielded better results than performing them in reverse order and also was better than using only one of the two.

Reassignment aggressively tries to maximize robustness radius by increasing the numerator and simultaneously reducing the denominator of equation (2). Swapping can be interpreted as a fine tuning procedure where the number of tasks on each machine is unaltered.

This section describes seven heuristics for the problem of finding a robust static allocation. Also, a mathematical upper bound on performance is derived.

4.1. Max-Max

The Max-Max heuristic is based on the Min-Min (greedy) concept in [21]. In step 2 of the Max-Max heuristic, to find the fitness function for assigning a given task i to a given machine j, the robustness radius of machine j given by equation (2) is evaluated based on the tasks already assigned to machine j and the possible assignment of task i to machine j.

The Max-Max heuristic can be summarized by the following procedure:

1. A task list is generated that includes all the unmapped tasks.
2. For each task in the task list, the machine that gives the task its maximum fitness value (first “Max”) is determined (ignoring other unmapped tasks).
3. Among all the task/machine pairs found in the above step, the pair that gives the maximum fitness value (second “Max”) is chosen.
4. The task found in step 3 is then removed from the task list and is mapped to its paired machine.
5. Repeat steps 2 to 4 until all the tasks are mapped.

4.2. Greedy Iterative Maximization

The Greedy Iterative Maximization (GIM) heuristic loops through the sequence of initial mapping generation and robustness improvement until the wall clock time of one hour expires. The first initial mapping for GIM is generated using the Min-Min heuristic similar to [21] based on the completion times. The other initial mappings are generated using the Minimum Completion Time (MCT) heuristic that was used in [12] so that the makespan constraint is satisfied. Tasks are considered in a different order every time a new mapping is generated for MCT. The Min-Min and MCT mapping generation procedures are shown in Figures 1 and 2, respectively.

The GIM heuristic can be summarized by the following procedure.

1. An initial mapping is generated as described above.
2. The robustness metric and min-radius machine for the current mapping is found.
3. Generate a task list containing all tasks on the min-radius machine not yet considered for reassignment.
4. A task is chosen arbitrarily from the task list and considered for reassignment to all other machines.
5. Reassign the task to the machine that improves the robustness metric the most and go to step 2; if the reassignment does not improve the mapping, remove the task from the task list and go to step 4 until there are no tasks in the task list.
6. The robustness metric and min-radius machine for the current mapping is determined.
7. Generate a task list containing all tasks on the min-radius machine not yet considered for swapping.
8. A task is chosen arbitrarily from the task list and considered to be swapped to all tasks on all other machines.
9. The chosen task from the task list is swapped with the first task that will increase the robustness metric by traversing through all the tasks in arbitrary order on all other machines and go to step 6; if the chosen task could not be swapped with any other task, remove the task from the task list and go to step 8 until the task list is empty.
10. Repeat steps 1-9 until the one hour time constraint has expired.

```
1. A task list is generated that includes all
   unmapped tasks.
2. Find the completion time of each
   unmapped task on each machine (ignoring
   other unmapped tasks).
3. Find the machine that gives the minimum
   completion time for each task.
4. Among all the task/machine pairs found in
   3, find the pair that gives the minimum
   completion time.
5. Remove the above task from the task list
   and map it to the chosen machine.
6. Update the available time of the machine
   on which the task is mapped.
7. Repeat steps 2-6 until all the tasks have
   been mapped.
```

Figure 1: Pseudo-code for the Min-Min heuristic.

One variation tried was to select the “best” task that improves the robustness during swapping in step 9 and was found to perform slightly worse than the “arbitrary order” swap method. It is observed that, in general, the robustness of the initial mapping did not impact the robustness of the final mapping; however, if the robustness of the initial mappings are good, more iterations of steps 1 through 9 can be performed in the given time constraint.

```
1. A task list is generated that includes all
   unmapped tasks.
2. Choose a task arbitrarily from the task list.
3. Find the machine that gives the minimum
   completion time for the chosen task.
4. Remove the task from the task list and map
   it to the chosen machine.
5. Update the available time of the machine
   on which the task is mapped.
6. Repeat steps 2-5 until all the tasks have
   been mapped.
```

Figure 2: Pseudo-code for the MCT heuristic.

In another variation, GIM is initialized with the Max-Max heuristic. For this variation, the reassignment scheme is the same as before and swapping is done in the following way. For an arbitrary task *i* on the min-radius machine, a task *x* that is mapped on any other machine for which min-radius machine is the minimum execution time (MET) machine is chosen such that *ETC*(*x*, min-radius machine) is less than *ETC*(*i*, min-radius machine).

4.3. Overhead Iterative Maximization

The Overhead Iterative Maximization (OIM) heuristic starts with the MET mapping that was used in [12], where all the tasks are mapped to their fastest execution time machines. During an iteration, the robustness overhead, defined as the change in the sum of robustness radii of the machines after task reassignment or swapping, is maximized. For each task on the min-radius machine, OIM reassigns it to the machine that maximizes the robustness overhead if it will improve the robustness metric. Similar to the task reassignment procedure, each task on the min-radius machine is considered for swapping with a task on another machine.

The OIM heuristic can be summarized by the following procedure.

1. The MET mapping is generated and the robustness metric of the mapping is determined.
2. The min-radius machine for the mapping is found.
3. For each task on the min-radius machine, it is considered to be reassigned to all other machines.
4. If reassignment will increase the robustness metric, robustness overhead is recorded in a list *C* for each reassignment.
5. The task is reassigned to the machine that maximizes the robustness overhead the most and the list *C* is emptied.
6. Repeat steps 2-5 until no task can be reassigned from the current min-radius machine to improve the robustness metric.
7. The robustness metric and min-radius machine of the current mapping are determined.
8. For each task on the current min-radius machine, it is considered to be swapped with any task on other machines.
9. If swapping will increase the robustness metric, robustness overhead is recorded in a list *C* for each swap.
10. The relocation in *C* that has the maximum overhead is made and the list *C* is emptied.
11. Repeat steps 7-10 until no task swapping can be done.

4.4. GENITOR

GENITOR is a general optimization technique that is a variation of the genetic algorithm approach. It manipulates a set of possible solutions. The method studied here is similar to the standard GENITOR approach used in [40]. Each chromosome represents a possible complete mapping of tasks to machines. Specifically, the chromosome is a vector of length $|T|$. The i^{th} element of the vector is the number of the machine to which task *i* is assigned. The GENITOR operates on a fixed population of 200 chromosomes. The population includes one chromosome (seed) that is the Max-Max solution and the rest of the chromosomes are generated by randomly assigning tasks to machines. The entire population is sorted (ranked) based on their fitness (robustness metric) values. Chromosomes that do not meet the makespan constraint are allowed to be included in the population. The ranking is constructed so that all chromosomes that meet the constraint are listed first, ordered by their robustness metric value (highest first). The chromosomes that do not meet the makespan constraint are then listed, again ordered by their robustness metric value.

Next, a special function (described later) is used to select two chromosomes to act as parents. These two parents perform a crossover operation, and two new offspring are generated. The offspring are then evaluated and must immediately compete for inclusion in the population. If the new offspring has a higher fitness than the poorest member in the population, the offspring is inserted in sorted order in the population, and the poorest chromosome is removed. Otherwise, the new offspring is discarded.

The special function for selecting parent chromosomes is a linear bias function, used to provide a specific selective pressure [40]. For example, a bias of 1.5 implies that the top ranked chromosome in the population is 1.5 times more likely to be selected for a crossover or mutation than the median chromosome. The linear bias value of 1.5 was used to select chromosomes for crossover and mutation. Elitism, the property of guaranteeing the best solution remains in the population, is implicitly implemented by always maintaining the ranked list.

In the crossover step, for the pair of the selected parent chromosomes a random cut-off point is generated that divides the chromosomes into top and bottom parts. For the parts of both chromosomes from that point to the end of each chromosome, crossover exchanges machine assignments between corresponding tasks producing two new offspring.

After each crossover, the linear bias function is applied again to select a chromosome for mutation. A mutation operator generates a single offspring by perturbing the original chromosome. A random task is chosen for the chromosome and the mutation operator randomly reassigns it to a new machine. The resultant offspring is considered for inclusion in the population in the same fashion as for an offspring generated by crossover.

This completes one iteration of the GENITOR. The heuristic stops when the criterion of 250,000 total iterations is met.

4.5. Memetic Algorithm

The Memetic Algorithm (<u>MA</u>) metaheuristic [30] combines population-based global search with local search made by each of the individuals. Each individual represents a complete mapping of tasks to machines, and is the same as GENITOR chromosomes. The local search hill climbing is a process that starts at a certain solution, and moves to a neighboring solution if it is better than the current solution until a stopping criterion is reached. The interactions between individuals are made with the use of a crossover operator. Later, an individual is mutated by partly modifying an existing solution. Hill climbing is done on all individuals in the initial population and also on the offspring generated after crossover and mutation.

The MA heuristic can be summarized by the following procedure.

```
1. Generate initial population, as in GENITOR.
2. Hill climb on each member of the population.
   While (stopping criteria (i.e., 500 iterations) not met)
   {
   a. Select two tasks arbitrarily and their machine assignments are swapped.
   b. If (robustness metric of offspring > robustness metric of original individual),
      replace the original individual, otherwise ignore the offspring.
   }
3. Evaluate robustness metric for each individual.
4. Rank population based on robustness metric, as in GENITOR.
5. While (stopping criteria (i.e., 100,000 iterations) not met)
   {
   a. Perform crossover, as in GENITOR.
   b. After crossover operation, perform step 2 (hill climb) on the offspring.
   c. Perform mutation, as in GENITOR.
   d. After mutation operation, perform step 2 (hill climb) on the offspring.
   e. The population size stays fixed at the best 200 individuals, as in GENITOR.
   }
6. Output the best solution.
```

4.6. Ant Colony Optimization

The Ant Colony Optimization (<u>ACO</u>) metaheuristic has been shown to be an effective strategy for several problems closely related to scheduling jobs in an HC environment [32]. The ACO algorithm implemented here is a variation of the ACO algorithm design described in [32]. The first step in the ACO algorithm design is to define the pheromone trail. The pheromone trail will enable the ants to share useful information about good solutions. Similar to [32], because of the nature of the problem, it would be useful to store information about good machines for each task. Here, the pheromone value $\underline{\tau(i, j)}$ represents the "goodness" of a particular machine *j* for a particular task *i*. Hence, the pheromone table will have an entry for each task on each machine similar to the ETC matrix. The information in the pheromone table will be shared by all the ants. At a high level, the ACO heuristic works in the following way. A certain number of ants are released to find different complete mapping solutions. Based on the mapping of the individual ants, the global pheromone trail is updated (procedure described later). This procedure is iterated for a predefined number of times. The final mapping solution is determined by mapping each task to its highest pheromone value machine.

The ants build their solution based on (a) problem-specific information (robustness radius) and (b) pheromone table information, in a heuristic fashion. The ant procedure is as follows. Similar to the Max-Max two phase greedy heuristic, the ant procedure involves two phases. Initially, a task list is created that has all the unmapped tasks. Let $\underline{n}$ be the number of tasks in the task list at any given instance. In Phase 1, for each task in the task list, the machine that gives the maximum robustness radius is determined just as in step 2 of the Max-Max heuristic. The robustness radius of each task is then normalized with respect to the robustness radius of the "best" task (found by looping through all the tasks in the task list). This value for task *i* is termed as the <u>worth</u> of the task and represented as

$\eta(i)$. In Phase 2, an unmapped task is stochastically selected (procedure described later) and assigned to the machine that gave its maximum robustness radius. The mapped task is removed from the task list. The procedure is repeated until all the tasks in the task list are mapped. The fitness function of the ant s, $f(s)$, is the robustness metric at the end of the ant procedure.

To allow all the ants to share information about good solutions, the pheromone trail must be updated. Instead of allowing all the ants to leave pheromone values, only the best ant of the iteration is allowed. This variation is shown to improve the performance of ACO algorithms significantly [32, 38]. Only the best ant s_{best} of the iteration is allowed to update the pheromone table. The global best ant, s_{gb}, of the ACO procedure is the one that has the maximum robustness radius among all the ants so far (including all the previous iterations). To allow the ants to "forget" poor information, each pheromone value is decayed with a parameter ρ that takes values between 0 and 1. The pheromone trail update at the end of each iteration is given by equation (4). It is done for all i and j.

$$\tau(i,j)=\begin{cases}\rho\,.\,\tau(i,j)+\dfrac{f(s_{best})}{f(s_{gb})} & \text{if task } i \text{ is allocated to machine } j \text{ in } s_{best}\\ \rho\,.\,\tau(i,j) & \text{otherwise}\end{cases} \quad (4)$$

The stochastic task selection rule used by traditional ACOs is called the random-proportional rule. For each task i, let p^{i}_{best} be the machine that maximizes the robustness radius for task i. Let α be the pheromone value weight and β be the weight value given to the heuristic information (robustness radius). For this study, the values of α and β are constrained such that $\alpha + \beta = 1$. The probability of selecting task i, $prob(i)$, to map next is given by the following equation.

$$prob(i)=\frac{[\tau(i,p^{i}_{best})]^{\alpha}\,.\,[\eta(i)]^{\beta}}{\sum_{k=0}^{n-1}[\tau(k,p^{k}_{best})]^{\alpha}\,.\,[\eta(k)]^{\beta}} \quad (5)$$

That is, the next task is selected randomly such that tasks with a higher $prob(i)$ have a higher probability of selection. The pheromone table is initialized to 1 and then seeded with the Max-Max solution. Seeding means that

$$\tau(i,j)=\begin{cases}\rho+1 & \text{if task } i \text{ is allocated to machine } j \text{ by Max-Max}\\ \rho & \text{otherwise}\end{cases} \quad (6)$$

A local search strategy, similar to the reassignment procedure of the IM heuristics, is incorporated. This strategy tries to move a task from the min-radius machine to any other machine that gives the best improvement in the robustness metric. This local search is applied to each of the ant solutions before the pheromone update stage to try to take the ant solution to its local optimum in the search space.

The α and β values determine the extent to which the pheromone information and heuristic information, respectively, will be used by the ants to build their solution. The α value was determined experimentally by incrementing from 0 to 1 in steps of 0.1. The α value of 0.1 and β value of 0.9 was found to give good results. By similar experiments, the pheromone decay factor ρ of 0.75 was selected. There are 250 iterations with 10 ants in each iteration (kept constant). These values were chosen to a give good solution in the wall clock time constraint of one hour for the heuristic running time.

4.7. HereBoy Evolutionary Algorithm

HereBoy is a fast evolutionary algorithm that combines the features of GA and SA [26]. Unlike GA, there is only a single individual undergoing optimization, not a population. The individual or the chromosome is a task to machine mapping similar to the GENITOR and MA. Because there is only one individual, the search space is explored only using chromosome mutation. Mutations are kept if they produce an individual that performs better than its parent. The poor performers are discarded although some can be kept based on a probability test analogous to the SA approach.

HereBoy starts with an MCT mapping. An adaptive mutation scheme is employed by the HereBoy heuristic. In this scheme, the number of tasks to be mutated at each iteration or the mutation rate (γ) is given by (8). Mutation is applied by randomly selecting a task on the chromosome and changing its machine assignment. The task chosen is first unmapped from its currently assigned machine, and mapped to the machine that maximizes the robustness metric. Randomly assigning the chosen task to a new machine was also tried, but it performed poorly and so was not used. The mutation rate is determined by two terms: the maximum mutation rate, α, which is the product of the user defined fraction and the number of tasks $|T|$, and the fraction β that reduces the number of tasks mutated as the current robustness approaches the upper bound (UB) value on the robustness metric. Mathematically, the fraction β is calculated based on the equation given below.

$$\beta=\frac{\left(\mathrm{UB}-\rho_{\mu}(\phi,C)\right)}{\mathrm{UB}} \quad (7)$$

$$\gamma=\alpha\,.\,\beta \quad (8)$$

The chromosome mapping solution is evaluated at the end of each mutation. A probability test is performed to accept poorer solutions so that the surrounding neighborhood is searched for better opportunities. The test probability starts with a high value and reduces over time and is referred to as the cooling schedule [12]. Typically cooling schedules are predefined, although it has been shown that adaptive schedules produce better results [26].

An adaptive scheme is employed by HereBoy to reduce the search probability (η). The search probability is given by equation (9) that is similar to the adaptive mutation rate formula. The search probability is the product of the user defined value maximum search probability (ρ) and the fractional term β defined earlier. Notice that the probability of accepting poor solutions reduces as better solutions are produced.

$$\eta = \rho \,.\, \beta \tag{9}$$

As a result of experimentation, HereBoy is run with a 5% maximum mutation rate α. The maximum search probability ρ is set to 0 for this problem because accepting poor solutions did not improve the final mapping. The stopping criteria for the heuristic are a total of 100,000 iterations or no change in the chromosome for 10,000 successive iterations.

4.8. Upper Bound

The method developed for estimating an upper bound on the robustness metric for this study assumes a homogeneous MET system in which the execution time for each task on all machines is the same and equal to the minimum time that the task would take to execute across the original set of machines. The minimum execution time of task i, MET_i, is given by the following equation.

$$MET_i = \min ETC(i, j) \text{ over all } j \tag{10}$$

The upper bound for the robustness metric of the homogeneous MET system is equal to or better than the upper bound for the robustness metric of the original system because of the impact of the MET values in equation (2). The tasks in the MET system are arranged in ascending order of their execution times. Then, the robustness upper bound is calculated as follows.

Let $N = \lfloor |T|/|M| \rfloor$. The first N tasks in the sorted order are stored in a list S. For the purposes of this mathematical upper bound derivation, the same N tasks in S are assumed to be on all the machines so that $F_j = F_i$, $1 \le i,j \le |M|$. Thus, the upper bound for robustness is given by equation (11).

$$\text{UB} = \frac{\left(\tau - \sum_{i=0}^{|S|-1} MET_i\right)}{\sqrt{N}} \tag{11}$$

Proof by contradiction:

Assume that there is another solution whose robustness metric is greater than UB and has machines with fewer tasks than N. If there is a machine with tasks fewer than N, then there must be a machine m_x with more than N tasks mapped on to it. So, $\sqrt{\text{number of tasks on } m_x} > \sqrt{N}$. Because the list S consists of the N tasks with the smallest ETC values, and machine m_x has more than N tasks, its completion time must be greater than the sum of the execution time of all tasks in S. Thus, $F_x > \sum_{i=0}^{|S|-1} MET_i$. Therefore, $r_\mu(F_x, C) <$ UB. Because the machine with the least robustness radius determines the robustness metric of the entire system, there cannot be a mapping without tasks equally distributed to have robustness greater than UB.

Now, assume a different solution Sol^* has N tasks on each of the machines and has a robustness metric greater than UB. Thus, by equation (2), the finishing time of all machines for Sol^* must be less than $\sum_{i=0}^{|S|-1} MET_i$. But this summation is the smallest possible F_j for any j. Hence, there cannot be a mapping with N tasks on each machine and a robustness metric larger than UB.

The method used to construct this mathematically sound upper bound results in a loose upper bound. Furthermore, the greater the heterogeneity, the looser the bound.

5. Experimental Results

The simulation results are shown in Figures 3 and 4. All the heuristics are run for 100 different scenarios and the average values and 95% confidence intervals [22] are plotted. The running times of the heuristics averaged over 100 trials, mapping 1024 tasks onto eight machines, are shown in Table 1.

The GIM and OIM are among the best heuristics for both of the high-high and low-low cases studied here. The IM heuristics that make use of the tailored search technique (as opposed to the general search used by GENITOR) proved to be very effective. The “best” swap variation of the GIM arrived at a good solution faster than the “arbitrary order” swap; however, the latter performed more beneficial swaps and showed a gradual increase in the robustness better than the former. The third variation of the GIM heuristic that is seeded with the Max-Max solution is less than 2% of the “arbitrary swap” variation. In this approach, not many beneficial swaps could be made and hence, a poor initial solution did not perform comparably to the other variations.

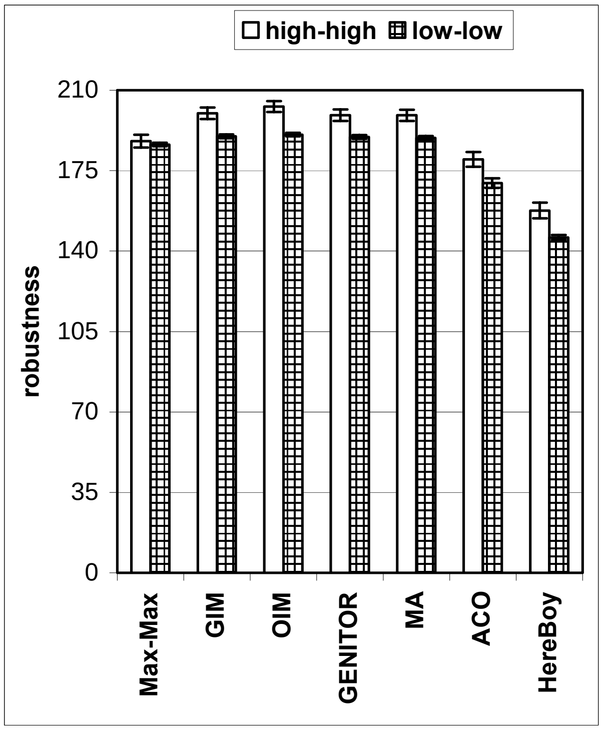

Figure 3: The simulation results for robustness. The average UB values are 416.54 for high-high heterogeneity and 313.83 for low-low heterogeneity.

The GENITOR and MA performed comparably to the IM heuristics. Both of the heuristics are seeded with the Max-Max solution and used the concept of elitism. The ACO solution was within 12% of the best heuristic (OIM) solution. In the ACO heuristic, seeding the pheromone trial with the Max-Max mapping and the use of the local search technique improved the solution on average by 27%.

The makespan results of all the heuristics are similar and in agreement with the robustness metric results. Notice that for a similar makespan, the GIM heuristic showed 6% better robustness for the hi-hi case and 2% better robustness for the low-low case over the Max-Max heuristic. This clearly implies that even though the makespan and robustness of a mapping are related, minimizing the makespan does not automatically maximize the robustness.

The Max-Max and HereBoy are the fastest among all of the heuristics studied here. Among the faster heuristics, the two-phase greedy heuristic (Max-Max) performed better than the fast evolutionary algorithm (HereBoy). The HereBoy heuristic performed the worst among all of the heuristics. The upper bound used here is a loose upper bound (considering all the assumptions made in the derivation), and hence, the adaptive mutation technique that uses the upper bound value did not prove useful.

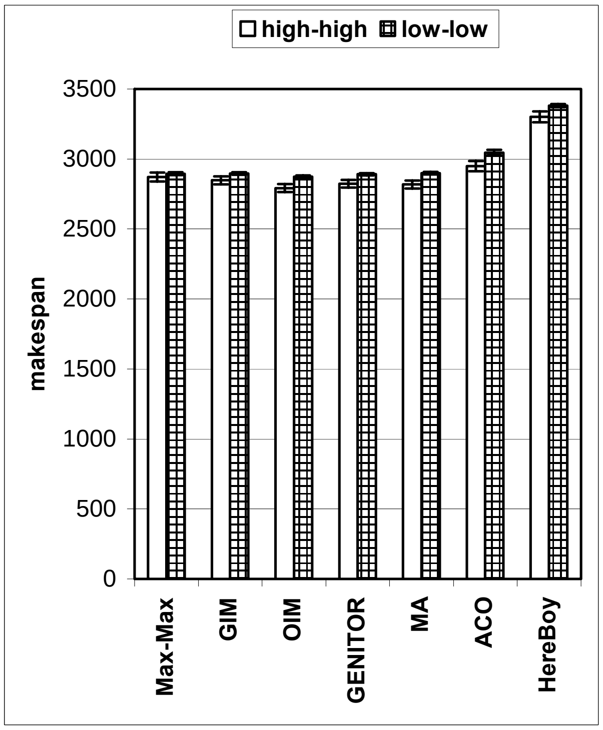

Figure 4: The simulation results for makespan, which is constrained to be ≤ 5,000 (= τ).

heuristic	average execution times (seconds)
Max-Max	0.52
Greedy IM	3600
Overhead IM	600
GENITOR	3000
Memetic Algorithm	3000
Ant Colony Optimization	3200
HereBoy	2.2

Table 1: The average execution times of the heuristics averaged over 100 trials (using a typical unloaded 3 GHz Intel Pentium 4 machine).

6. Summary

This paper presents seven static heuristics for maximizing the robustness of a mapping against errors in the ETC values using the HC environments presented. A system of independent tasks is mapped onto a set of heterogeneous machines using the heuristics described in this research.

The best robustness metric is obtained by using the Overhead Iterative Maximization heuristic. The Greedy Iterative Maximization, GENITOR, and Memetic Algorithm performed comparably with their robustness metric within 2% of the Overhead Iterative Maximization. However, the execution times for the heuristics themselves are much higher as compared to the Overhead Iterative Maximization heuristic. Thus, Overhead Iterative Maximization is a good choice for the given problem.

Acknowledgments: The authors thank Shoukat Ali and Jay Smith for their valuable comments.

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 resource allocation heuristics for heterogeneous computing systems," *Computer Architecture*, A. R. Hurson, ed., a volume of Advances in Computers, Elsevier, New York, NY, to appear in 2005.

[2] S. Ali, J.-K. Kim, H. J. Siegel, A. A. Maciejewski, Y. Yu, S. B. Gundala, S. Gertphol, and V. Prasanna, "Greedy heuristics for resource allocation in dynamic distributed real-time heterogeneous computing systems," *2002 International Conference on Parallel and Distributed Processing Techniques and Applications* (*PDPTA 2002*), June 2002, pp. 519–530.

[3] S. Ali, A. A. Maciejewski, H. J. Siegel, and J.-K. Kim, "Robust resource allocation for sensor-actuator distributed computing systems," *The 2004 International Conference on Parallel Processing (ICPP 2004)*, Aug. 2004, pp. 174–185.

[4] S. Ali, A. A. Maciejewski, H. J. Siegel, and J.-K. Kim, "Measuring the robustness of a resource allocation," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 15, No. 7, July 2004, pp. 630–641.

[5] S. Ali, H. J. Siegel, M. Maheswaran, D. Hensgen, and S. Ali, "Representing task and machine heterogeneities for heterogeneous computing systems," *Tamkang Journal of Science and Engineering*, Special 50th Anniversary Issue, Vol. 3, No. 3, Nov. 2000, pp. 195–207 (invited).

[6] S. Areibi, M. Moussa, and H. Abdullah, "A comparison of genetic/memetic algorithms and heuristic searching," *International Conference on Artificial Intelligence (IC-AI 2001)*, June 2001.

[7] H. Barada, S. M. Sait, and N. Baig, "Task matching and scheduling in heterogeneous systems using simulated evolution," *10th IEEE Heterogeneous Computing Workshop (HCW 2001)*, Apr. 2001.

[8] I. Banicescu and V. Velusamy, "Performance of scheduling scientific applications with adaptive weighted factoring," *10th IEEE Heterogeneous Computing Workshop (HCW 2001)*, Apr. 2001.

[9] L. Bölöni and D. C. Marinescu, "Robust scheduling of metaprograms," *Journal of Scheduling*, Vol. 5, No. 5, Sep. 2002, pp. 395–412.

[10] C. Banino, O. Beaumont, L. Carter, J. Ferrante, A. Legrand, Y. Robert, "Scheduling strategies for master-slave tasking on heterogeneous processor platforms", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 15, No. 4, Apr. 2004, pp. 319–330.

[11] T. D. Braun, H. J. Siegel, and A. A. Maciejewski, "Heterogeneous computing: Goals, methods, and open problems," *2001 International Conference on Parallel and Distributed Processing Techniques and Applications* (*PDPTA 2001*), June 2001, pp. 1–12 (invited keynote paper).

[12] 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 Bin 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*, Vol. 61, No. 6, June 2001, pp. 810–837.

[13] E. G. Coffman, Jr. ed., *Computer and Job-Shop Scheduling Theory*, John Wiley & Sons, New York, NY, 1976.

[14] R. L. Daniels and J. E. Carrilo, "β-Robust scheduling for single-machine systems with uncertain processing times," *IIE Transactions*, Vol. 29, No. 11, Nov. 1997, pp. 977–985.

[15] A. J. Davenport, C. Gefflot, and J. C. Beck, "Slack-based techniques for robust schedules," *6th European Conference on Planning*, Sep. 2001, pp. 7–18.

[16] J. Dorn, M. Girsch, G. Skele, and W. Slany, "Comparison of iterative improvement techniques for schedule optimization," *European Journal on Operations Research*, Vol. 94, No. 2, Oct. 1996, pp. 349–361.

[17] J. Dorn, R. M. Kerr, and G. Thalhammer, "Reactive scheduling: Improving the robustness of schedules and restricting the effects of shop floor disturbances by fuzzy reasoning," *International Journal on Human-Computer Studies*, Vol. 42, No. 6, June 1995, pp. 687–704.

[18] M. M. Eshaghian, ed., *Heterogeneous Computing*, Norwood, MA, Artech House, 1996.

[19] D. Fernandez-Baca, "Allocating modules to processors in a distributed system," *IEEE Transaction*

on Software Engineering, Vol. SE-15, No. 11, Nov. 1989, pp. 1427–1436.

[20] I. Foster and C. Kesselman, eds., *The Grid: Blueprint for a New Computing Infrastructure*, San Fransisco, CA, Morgan Kaufmann, 1999.

[21] O. H. Ibarra and C. E. Kim, "Heuristic algorithms for scheduling independent tasks on non-identical processors," *Journal of the ACM*, Vol. 24, No. 2, Apr. 1977, pp. 280–289.

[22] R. Jain, *The Art of Computer Systems Performance Analysis Techniques for Experimental Design, Measurement, Simulation, and Modeling*, Wiley, New York, 1991.

[23] M. Kafil and I. Ahmad, "Optimal task assignment in heterogeneous distributed computing systems," *IEEE Concurrency*, Vol. 6, No. 3, July-Sep. 1998, pp. 42–51.

[24] P. Kouvelis and G. Yu, *Robust Discrete Optimization and Its Applications*, Dordrecht, Kluwer, 1997.

[25] P. Kouvelis, R. Daniels, and G. Vairaktarakis, "Robust scheduling of a two-machine flow shop with uncertain processing times," *IIE Transactions*, Vol. 38, No. 5, May 2000, pp. 421–432.

[26] D. Levi, "Hereboy: A fast evolutionary algorithm," *2nd NASA/DoD Workshop on Evolvable Hardware (EH '00)*, July 2000, pp. 17–24.

[27] V. J. Leon, S. D. Wu, and R. H. Storer, "Robustness measures and robust scheduling for job shops," *IIE Transactions*, Vol. 26, No. 5, Sep. 1994, pp. 32–43.

[28] 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*, Vol. 59, No. 2, Nov. 1999, pp. 107–121.

[29] M. Maheswaran, T. D. Braun, and H. J. Siegel, "Heterogeneous distributed computing," *Encyclopedia of Electrical and Electronics Engineering*, J. G. Webster, ed., Vol. 8, John Wiley & Sons, New York, NY, 1999, pp. 679–690.

[30] P. Moscato, *On Evolution, Search, Optimization, Genetic Algorithms, and Martial Arts: Towards Memetic Algorithms,* Technical Report, Caltech Concurrent Computation Program C3P 826, California Institute of Technology, Pasadena, CA, 1989.

[31] G. C. Onwubolu and B. V. Babu, *New Optimization Techniques in Engineering*, Springer-Verlag, New York, NY, 2004.

[32] G. Ritchie and J. Levine, "A hybrid ant algorithm for scheduling independent jobs in heterogeneous computing environments," *3rd Workshop of the UK Planning and Scheduling Special Interest Group* (PLANSIG 2004), Dec. 2004.

[33] M. Sevaux and K. Sörensen, "Genetic algorithm for robust schedules," *8th International Workshop on Project Management and Scheduling (PMS 2002)*, Apr. 2002, pp. 330–333.

[34] G. F. Simmons, *Calculus with Analytic Geometry, Second Edition*, New York: McGraw-Hill, 1995.

[35] S. Shivle, R. Castain, H. J. Siegel, A. A. Maciejewski, T. Banka, K. Chindam, S. Dussinger, P. Pichumani, P. Satyasekaran, W. Saylor, D. Sendek, J. Sousa, J. Sridharan, P. Sugavanam, and J. Velazco, "Static mapping of subtasks in a heterogeneous ad hoc grid environment," *13th IEEE Heterogeneous Computing Workshop (HCW 2004)*, Santa Fe, NM, Apr. 2004.

[36] S. Shivle, H. J. Siegel, A. A. Maciejewski, T. Banka, K. Chindam, S. Dussinger, A. Kutruff, P. Penumarthy, P. Pichumani, P. Satyasekaran, D. Sendek, J. Sousa, J. Sridharan, P. Sugavanam, and J. Velazco, "Mapping of subtasks with multiple versions in a heterogeneous ad hoc grid environment," *3rd International Workshop on Algorithms, Models, and Tools for Parallel Computing on Heterogeneous Networks (HetroPar 2004)*, Cork, Ireland, July 2004.

[37] Y. N. Sotskov, V. S. Tanaev, and F. Werner, "Stability radius of an optimal schedule: A survey and recent developments," *Industrial Applications of Combinatorial Optimization*, Vol. 16, 1998, pp. 72–108.

[38] T. Stützle and H. Hoos, "Max-min ant system," *Future Generation Computer Systems*, Vol. 16, No. 8, 2000, pp. 889–914.

[39] 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*, Special Issue on Parallel Evolutionary Computing, Vol. 47, No. 1, Nov. 25, 1997, pp. 8–22.

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

[41] M.-Y. Wu, W. Shu, and H. Zhang, "Segmented min-min: A static mapping algorithm for meta-tasks on heterogeneous computing systems," *9th IEEE Heterogeneous Computing Workshop* (*HCW 2000*), May 2000, pp. 375–385.

Biographies

Prasanna V. Sugavanam is pursuing his M.S. degree in Electrical and Computer Engineering at Colorado State University, where he is currently a Graduate Teaching Assistant. He received his Bachelor of Engineering in Electrical and Electronics from Bharathiar University, India in 2001. His research interests include resource management in distributed computing systems, computer architecture, and fault-tolerant computing systems. His previous projects include resource management for clusters for IBM Corporation, Boulder. He is student member of IEEE.

H. J. Siegel holds the endowed chair position of Abell Endowed 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 300 technical papers. His research interests include heterogeneous parallel and distributed computing, communication networks, 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 currently on the Steering Committees of three continuing conferences/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 sciences honor society. *An up-to-date vita is available at 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. Maciejewski's primary research interests relate to the analysis, simulation, and control of robotic systems and he has co-authored over 100 published technical articles in these areas. He is an Associate Editor for the *IEEE Transactions on Robotics and Automation*, a Regional Editor for the journal *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 serves 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.*

Syed Amjad Ali is currently pursuing a M.S. degree in Computer Information Systems at Colorado State University. He received his Bachelor's in Computer Science and Engineering from BAM University in India. His research interests include computer architecture, grid computing, and heterogeneous computing. He is currently involved with IBM to setup a grid system at the CSU College of Business.

Mohammad Al-Otaibi is currently pursing his Ph.D. in the Department of Computer Science at New Mexico Institute of Mining and Technology. He received his M.S. in Electrical and Computer Engineering from Colorado State University and B.S. in Computer Engineering from King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia. He worked with Lucent Technologies in Saudi Arabia as a Computer Network Engineer from 1998 to 1999. His research interests are in the field of computer networks, heterogeneous computing and reconfigurable computing.

Mahir Aydin is pursuing his Ph.D. degree in Electrical and Computer Engineering at Colorado State University. He is also working for Premiere Systems in Fairfax, Virginia as a software engineer. He received his Bachelor of Engineering degree in Computer Engineering and his Master of Science degree in computer science from Union College, Schenectady, New York. His current interests include computer architecture, software engineering, microprocessors, networks, database design, and VLSI design.

Kumara Guru is a graduate student of Colorado State University pursuing his M.S in Electrical and Computer Engineering. He received his B.E degree in Electronics and Communication from the University of Madras in 2003. His research interests include computer architecture, heterogeneous computing, and optics.

Aaron Horiuchi is currently a Masters of Engineering student at CSU and a ASIC R&D engineer at Hewlett Packard. He obtained a B.S.E. with an Electrical Specialty in Dec 2001 at the Colorado School of Mines. His research interests include signal integrity, analog circuit design, and VLSI systems.

Yogish G. Krishnamurthy graduated from the Department of Computer Science at Colorado State University, where he received his Masters in Computer Science in Dec. 2004. He received his Bachelor of Engineering in Computer Science and Engineering from Vishweshariah Technological University, India in June 2002. He is currently employed in Level 3 Communications as a Software developer working on core business applications.

Pan Ho Lee is a Ph.D. student in Electrical and Computer Engineering at Colorado State University. He received his B.S and M.S degrees in Computer Engineering from Kwang Woon Univeristy, Seoul, Korea in 1992 and 1994, respectively. From 1994 to 2003, he worked for Daewoo Telecom and LG Electronics as a research staff member. His current research interests are in the field of overlay transport and network protocols, sensor networks and distributed computing.

Ashish Mehta is a graduate student of Colorado State University pursuing his M.S. degree in Electrical and Computer Engineering. He received a B.E degree in Electronics Engineering from Fr. Conceicao Rodrigues College of Engineering, Mumbai, India. His fields of interest are heterogeneous computing, computer architecture, computer networks, and embedded systems.

Mohana Oltikar is pursuing a M.S. degree in Electrical and Computer Engineering at Colorado State University, where she is currently a Graduate Research Assistant. She has completed her bachelor’s degree in Electronics Engineering from University of Mumbai, India. She is currently working on the robustness of heterogeneous systems.

Ron Pichel received his B.S. degree in Electrical Engineering in 2001 from Valparaiso University in Indiana. He started graduate studies in computer engineering at Colorado State University. Currently, he is enrolled in National Technological University in pursuit of his M.S. degree in Computer Engineering. He is employed by Hewlett-Packard Company, where he works as a verification engineer for high-end server ASICs.

Alan Pippin is a member of the Systems-VLSI Lab at Hewlett-Packard in Fort Collins Colorado. He is involved in the design of chipsets for high end servers there. He received his bachelor's degree in Electrical & Computer Engineering from Brigham Young University in 2001. He is currently working on a Master's degree in Electrical Engineering at Colorado State University. He is also a member of the IEEE.

Michael Raskey received a B.S. in Electrical Engineering from Valparaiso University, and is currently a graduate student at Colorado State University pursing a M.S. in Electrical Engineering. Michael also currently works for Hewlett-Packard Company in Fort Collins, Colorado, as a Systems/Software Engineer working on Linux Quality Assurance for Integrity Servers. Michael's technical interests include computer architecture, internet engineering, and open source software.

Vladimir V. Shestak is pursuing a Ph.D. degree from the Department of Electrical and Computer Engineering at Colorado State University, where he has been a Research Assistant since August 2003. His current projects include resource management for clusters for IBM, Boulder. He received his M.S. degree in computer engineering from New Jersey Institute of Technology in May 2003. Prior to joining the New Jersey Institute of Technology he spent three years in industry as a Network Engineer working for CISCO Business Unit in Moscow, Russia. He received his BS degree in electrical engineering from Moscow Engineering Physics Institute, Moscow, Russia. His research interests include resource management within distributed computing systems, algorithm parallelization, and computer network design and optimization.

Junxing Zhang is currently pursuing his Ph.D. in the School of Computing at University of Utah. He received his M.S. in Computer Science from Colorado State University and B.E. in Computer Engineering from Beijing University of Posts and Telecommunications. His research interests include distributed and heterogeneous computing, software configuration management, computer networking, and data management systems.