Available online at www.sciencedirect.com
ScienceDirect
J. Parallel Distrib. Comput. 67 (2007) 400–416
Journal of Parallel and Distributed Computing
www.elsevier.com/locate/jpdc

Robust static allocation of resources for independent tasks under makespan and dollar cost constraints

Prasanna Sugavanam[a,*], H.J. Siegel[a,b], Anthony A. Maciejewski[a], Mohana Oltikar[a], Ashish Mehta[a], Ron Pichel[c], Aaron Horiuchi[c], Vladimir Shestak[a], Mohammad Al-Otaibi[a], Yogish Krishnamurthy[a], Syed Ali[a], Junxing Zhang[e], Mahir Aydin[a], Panho Lee[a], Kumara Guru[a], Michael Raskey[c], Alan Pippin[d]

[a]*Electrical and Computer Engineering Department, Colorado State University, Fort Collins, CO 80523, USA*
[b]*Computer Science Department, Colorado State University, Fort Collins, CO 80523, USA*
[c]*Systems and VLSI Technology Division, Hewlett-Packard Company, Fort Collins, CO 80528, USA*
[d]*Linux and Open Source Lab, Hewlett-Packard Company, Fort Collins, CO 80528, USA*
[e]*School of Computing, University of Utah, Salt Lake City, UT 84112, USA*

Received 15 June 2005; received in revised form 18 December 2005; accepted 23 December 2005

Abstract

Heterogeneous computing (HC) systems composed of interconnected machines with varied computational capabilities often operate in environments where there may be inaccuracies in the estimation of task execution times. Makespan (defined as the completion time for an entire set of tasks) is often the performance feature that needs to be optimized in such systems. Resource allocation is typically performed based on estimates of the computation time of each task on each class of machines. Hence, it is important that makespan be robust against errors in computation time estimates. In this research, the problem of finding a static mapping of tasks to maximize the robustness of makespan against the errors in task execution time estimates given an overall makespan constraint is studied. Two variations of this basic problem are considered: (1) where there is a given, fixed set of machines, (2) where an HC system is to be constructed from a set of machines within a dollar cost constraint. Six heuristic techniques for each of these variations of the problem are presented and evaluated.

Keywords: Heterogeneous computing; Robustness; Resource allocation; Makespan; Cost constraint

1. Introduction

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., [10,18]). Such systems often operate in an environment where certain desired performance features degrade due to unpredictable circumstances, such as higher than expected work load or inaccuracies in the estimation of task and system parameters (e.g., [2,4,9,27,28,39,40]). Thus, when resources are allocated to tasks it is desirable to do this in a way that makes the system performance on these tasks robust against unpredictable changes.

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

* Corresponding author.
E-mail addresses: prasanna@colostate.edu (P. Sugavanam), hj@colostate.edu (H.J. Siegel), aam@colostate.edu (Anthony A. Maciejewski), mohana@colostate.edu (M. Oltikar), ammehta@colostate.edu (A. Mehta), rgp@fc.hp.com (R. Pichel), akh@fc.hp.com (A. Horiuchi), shestak@colostate.edu (V. Shestak), motaibi@colostate.edu (M. Al-Otaibi), yogi@colostate.edu (Y. Krishnamurthy), sdamjad@colostate.edu (S. Ali), junxing@cs.utah.edu (J. Zhang), mahir@colostate.edu (M. Aydin), leepanho@colostate.edu (P. Lee), higuru@colostate.edu (K. Guru), michael.raskey@hp.com (M. Raskey), ajp@fc.hp.com (A. Pippin).

0743-7315/$ - see front matter © 2006 Published by Elsevier Inc.
doi:10.1016/j.jpdc.2005.12.006

system features against perturbations in system parameters [3]. The general problem of optimally mapping tasks to machines in an HC environment has been shown to be NP-complete (e.g., [12,19,23]). Thus, the development of heuristic techniques to find near-optimal solutions for the mapping problem is an active area of research (e.g., [1,7,8,11,18,20,21,26, 30,33,41,42,47,49]).

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 considered to be *robust* with respect to specified system performance features against perturbations in given system parameters if degradation in these features is within acceptable limits when certain perturbations occur [3]. *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 research, the problem of finding a static (off-line) mapping of tasks to maximize the robustness of makespan against errors in task execution time estimates is studied. It is a *static* (off-line) mapping because it is assumed that this HC system will be used to regularly execute predetermined metatasks in a production environment. The system is considered robust if the actual makespan under the perturbed conditions does not exceed the required time constraint, denoted by τ. Two variations to this basic problem are considered.

For the first problem variation, the goal is to find a static mapping of all tasks to a given, dedicated, fixed set of machines so that the robustness of the mapping is maximized within a given makespan constraint. Specifically, the goal is to maximize the collective allowable error in execution time estimation for the tasks that can occur without the makespan exceeding the constraint.

The second variation is a study of the problem of how to select (purchase) a fixed set of machines, within a given dollar cost constraint, to comprise an HC system. It is assumed that this fixed HC system will be used to regularly execute predetermined metatasks in a production environment, where the metatasks are from a known problem domain with known estimated computational characteristics. The machines to be purchased for the HC suite are to be selected from different classes of machines, where each class consists of machines of the same type. The machines of different classes differ in dollar costs depending upon their performance. The dollar cost of machines within a class is the same. To be able to use a machine for executing tasks, a one time dollar cost is incurred (i.e., to purchase the machines). Only a subset of all of the machines available can be chosen to execute tasks as there is a dollar cost constraint, denoted by δ. The objectives of this variation are to: (1) select a subset of all the machines available so that the cost constraint for the machines is satisfied, and (2) find a static mapping of all tasks to the subset. The goal of the subset selection and associated mapping is to maximize robustness. That is, the tasks to be executed are known, and the goal is to build a robust system.

In the next section, the research problem investigated is formally stated. Section 3 describes the simulation setup used for each of the problem variations studied in this research. Section 4 provides literature related to this work. In Section 5, the heuristics for the fixed suite variation are presented and evaluated. In Section 6, the heuristics for the selected suite variation are presented and evaluated.

2. Problem statement

In both variations of this research, a set of T tasks in the metatask is required to be allocated to a (given or chosen) set of M machines. The estimated time to compute (ETC) value for each task on each class of machines is assumed to be known *a priori*. This assumption is commonly made (e.g., [25]). Approaches for doing this estimation are discussed in [22,34]. It is assumed that unknown inaccuracies in the ETC values are expected (e.g., a task's actual exact execution time may be data dependent). Hence, it is required that the mapping, denoted by μ, be robust against them.

Let C^{est} be the vector of *estimated* computation times for the T tasks on the machine where they are allocated. Let C be the vector of *actual* computation times (C^{est} plus the estimation error for each task). The finish time of a given machine j, denoted by F_j, depends only on the actual computation times of the tasks mapped to that machine. The performance feature (ϕ) that determines if the makespan is robust is the finish times of the machines. That is, $\varphi = \{F_j | 1 \leqslant j \leqslant M\}$. The FePIA procedure from [3] is applied to determine the robustness metric for this problem.

The robustness radius [3] for machine j of F_j against C for mapping μ, denoted by $r_\mu(F_j, C)$, is defined as the largest Euclidean distance by which C can change in any direction from the assumed point C^{est} without the finish time of machine j exceeding the tolerable variation. This can be equivalently stated as the robustness radius is the minimum Euclidean difference in C from the assumed value of C^{est} within which the finish time of machine j can reach the tolerable variation. Mathematically,

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

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 finish time of the machine j will be at most the makespan constraint τ.

Because the finish time of a machine is simply the sum of the execution times of the tasks mapped to that machine, the makespan constraint is represented by a hyperplane. As described in [3], Eq. (1) can be interpreted as the perpendicular distance from C^{est} to the hyperplane described by the equation $\tau - F_j(C^{\text{est}}) = 0$. Hence, Eq. (1) can be rewritten using the point-to-plane distance formula from [43]:

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

The *robustness metric*, denoted by $\rho_\mu(\varphi, C)$, for the mapping is simply the minimum of all robustness radii [3]. Mathematically,

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

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

The goal for the first problem variation (*fixed suite*) of 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 $\rho_\mu(\phi, C)$. The goal for the second problem variation (*selected suite*) is to determine the set of machines such that: (1) the makespan is within τ, (2) the dollar cost for the chosen set of machines is within δ, and (3) the robustness metric is maximized. The emphasis of the second variation is on selecting the set of machines to accomplish the above stated goal. Simulations are used to evaluate and compare the heuristics studied in this research.

3. Simulation setup

An HC system with 1024 independent tasks is simulated for both problem variations 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]. The estimated execution time of task i on machine j is given by $ETC(i, j)$. A task mean and coefficient of variation (COV) are used to generate the ETC matrices.

The fixed suite variation consisted of eight machines in the HC suite. Two different cases of ETC heterogeneities are used in this research, the high-task and high-machine heterogeneity (*high–high*) case and the low-task and low-machine heterogeneity (*low–low*) case. For both cases, the ETCs are of the inconsistent type [5], i.e., a machine that executes faster for one task does not necessarily execute faster for all tasks. 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.

The high–high case uses a mean task execution time of 30 s 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 ith 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 s, 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 s 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 establish the value of τ.

For the selected suite variation, the HC system simulated has five different classes of machines, with eight machines in each class. All the machines in a given class are homogeneous (the execution time of any given task on all the machines is the same). The ETCs used in this research are of the high-task and low-machine (across various classes) heterogeneity (*high–low*). In this variation, the ETCs are *consistent* [5] across different classes of machines; i.e., if a class i machine is faster than a class j machine for one task, it is faster for all tasks. These assumptions are made to represent a realistic environment. The machines with higher dollar cost typically are equipped with faster processors, larger memory, etc., and in general, execute tasks faster than the low-end cheaper machines. For this study, heuristics are run for a total of 100 high–low trials.

The high–low case uses a mean task execution time of 180 s, a COV of 0.9 for task heterogeneity, and a COV of 0.3 for machine heterogeneity. The ETC values are sorted in ascending order to obtain the consistent heterogeneity. Class 1 is the fastest machine, Class 2 is the second fastest, and so on. It is assumed that all machines in a class use the same software environment. The dollar cost per machine is in accordance with their execution speeds: Class 1—1800, Class 2—1500, Class 3—1200, Class 4—800, and Class 5—500. These values are based on specific configurations of DELL desktop, workstation, and server products.

The cost constraint δ is chosen so that not all machines in the suite can be used, and the actual makespan constraint τ is chosen so that it adds significant mapping challenge to the problem. Experiments with simple greedy heuristics were used to decide the value of the cost constraint to be 34,800 dollars and the time constraint to be 12,000 s. Choosing different values for any of the above parameters will not affect the general approach of the heuristics used in this research. Because the tasks are independent, there is no communication between tasks. The time and resources required for loading the task executable file is simply assumed to be the same on all the machines. Hence, the network characteristics will not affect the solution of the problem and so it is ignored.

For both variations of this study, the wall clock time for the mapper itself to execute is arbitrarily required to be less than or equal to 60 min for any trial on a typical unloaded 3 GHz Intel Pentium 4 machine. This was done to establish a basis for comparing the different heuristic approaches.

4. Related work

The work presented in this paper is built upon the four-step FePIA procedure detailed in [3]. The FePIA procedure describes a way to derive a generalized robustness metric and it is applied to the problem studied here. In the literature, a number of papers have studied the issue of robustness in distributed systems (e.g., [9,13,14,17,27,32,44]). Robust decision making formulations presented in [13,27, 28] 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 [13] 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 [13] determine a 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 one 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 work described in [14,17,27,28,32] considers robust resource allocation in job-shop environments. The central idea in [14] is to provide each job with extra time (defined as slack) to complete 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. They define repair steps if a job takes longer than expected so that the new evaluation of proximity to constraint violation would be as good as or better than the old evaluation. In [27,28], the authors assume a scenario-based approach to represent the input data uncertainty to their robustness decision model. In [32], the authors assume a certain random distribution of the machine breakdowns and a certain rescheduling policy in the event of breakdowns. 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 and no mathematical characterization of the possible uncertainties in ETC values is assumed. This lack of a mathematical characterization is common in many current problem domains. Thus, our problem differs in many ways from scheduling machines in a job-shop environment.

In [44], the stability radius of an optimal schedule in a job-shop environment is calculated. The stability radius of an optimal schedule is defined as the radius of a closed sphere in the space of the numerical input data such that, within that sphere, the schedule remains optimal. Outside the sphere, which is centered at the assumed input, some other schedule would outperform the optimal schedule. In terms of the framework presented in [3], the robustness requirement would be the existence of an optimal schedule in the face of perturbations in the input data. Thus, the stability radius can be considered as a special case of the robustness metric that is used in this work.

The literature was examined to select a set of heuristics appropriate for the HC environments 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., [11,23,49]). 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 [48]. 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. The memetic algorithm (MA) [6,35,36], also referred to as a hybrid GA, applies a separate local search process (*hill-climbing*) to refine chromosomes. Combining global and local search is a strategy used by many successful global optimization approaches [6]. The HereBoy evolutionary algorithm used here is a combination of GA and simulated annealing (SA) and is an adaptation of the work in [31] 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.

All of the heuristics in the selected suite variation use as a component machine assignment heuristics from the fixed suite variation. The partition/merge methods used in [29] are adapted to our environment to find the set of machines to be used for the greedy iterative maximization heuristic. The research in [15,37] has used variations of GA for the synthesis of heterogeneous multiprocessors in embedded systems. Some of the other techniques used for machine selection are designed specifically for this environment.

5. Heuristics descriptions for the fixed machine suite problem

This section describes six heuristics for the problem of finding a robust static allocation for a given, fixed, dedicated set of machines. Also, a mathematical upper bound on performance is derived.

5.1. Max–Max

The Max–Max heuristic (see Fig. 1) is based on the Min–Min (greedy) concept in [23]. 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 Eq. (2) is evaluated based on the tasks already assigned to machine j and the possible assignment of task i to machine j.

5.2. Greedy iterative maximization

Both of the 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 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.

The greedy iterative maximization (GIM) heuristic (see Fig. 2) 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 [23] based on task completion

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.

Fig. 1. Pseudocode for the Max–Max heuristic.

1. An initial mapping is generated with Min-Min or MCT.
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.
4. A task is chosen arbitrarily from the task list.
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.
8. A task is chosen arbitrarily from the task list.
9. The chosen task from the task list is swapped with the first target 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.

Fig. 2. Pseudocode for the GIM heuristic.

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 minimum completion time (first "Min") is determined (ignoring other unmapped tasks).
3. Among all the task/machine pairs found in the above step, find the pair that gives the minimum completion time (second "Min").
4. Remove the above 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.

Fig. 3. Pseudocode for the Min–Min heuristic.

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.

Fig. 4. Pseudocode for the MCT heuristic.

times. The other initial mappings are generated using the minimum completion time (MCT) heuristic that was used in [11] so that the makespan constraint is satisfied. Tasks are considered in a different random order every time a new mapping is generated for MCT. The Min–Min and MCT mapping generation procedures are shown in Figs. 3 and 4, respectively. 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 Eq. (2). Swapping

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. Each task on the min-radius machine is considered to be reassigned to all other machines.
4. If reassignment will increase the robustness metric, the task is reassigned to the machine that maximizes the robustness improvement the most.
5. Repeat steps 2-4 until no task can be reassigned from the current min-radius machine to improve the robustness metric.
6. The robustness metric and min-radius machine of the current mapping are determined.
7. Each task on the current min-radius machine is considered to be swapped with any task on other machines.
8. If swapping will increase the robustness metric, the relocation that has the maximum robustness improvement is made.
9. Repeat steps 6-8 until no task swapping can be done.

Fig. 5. Pseudocode for the SIM heuristic.

can be interpreted as a fine tuning procedure where the number of tasks on each machine is unaltered.

One variation tried was to select the "best" target task that improves the robustness the maximum during swapping in step 9 and was found to perform slightly worse than the "arbitrary order" swap method. 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 the min-radius machine is the minimum execution time (MET) machine is chosen such that $ETC(x, \text{min-radius machine})$ is less than $ETC(i, \text{min-radius machine})$.

5.3. Sum iterative maximization

The sum iterative maximization (SIM) heuristic (see Fig. 5) starts with the MET mapping that was used in [11], where all the tasks are mapped to their MET machines. During an iteration, the *robustness improvement*, 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, SIM reassigns it to the machine that maximizes the robustness improvement 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.

5.4. 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 Genitor approach used in [48]. Each chromosome represents a possible complete mapping of tasks to machines. Specifically, the chromosome is a vector of length T. The ith element of the vector is the identification 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) in decreasing order based on their fitness (robustness metric) values. Chromosomes that do not meet the makespan constraint are included in the population, and have negative robustness values.

A linear bias function (with a value of 1.5) [48] is used to select two chromosomes to act as parents. These two parents perform a crossover operation, where 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. The two offspring are inserted in sorted order in the population, and the two poorest chromosomes are removed.

After each crossover, the linear bias function is applied again to select a chromosome for mutation. A random task is chosen from the chromosome and reassigned to a random 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 (see Fig. 6).

5.5. Memetic algorithm

The MA metaheuristic [35] (see Fig. 7) 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 a Genitor chromosome. 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

```
1. generate initial population
2. evaluate robustness metric for each chromosome
3. rank population based on robustness metric
4. while (stopping criteria (i.e., 250,000 iterations) not met)
   {
   a. select two chromosomes to act as parents using a linear bias function
      to perform crossover
      i.  cut the chromosomes at a random spot into top and bottom parts
      ii. exchange the machine assignments of the tasks in the bottom of the
          chosen chromosomes
      iii. insert the offspring in the sorted population based on its robustness
          metric
   b. select one chromosome using a linear bias function in the mutation step
      i.  for the chosen chromosome, choose a random task and change its
          machine assignment arbitrarily
      ii. insert the offspring in the sorted population based on its robustness
   c. the population size stays fixed at the best 200 chromosomes
   }
5. output the best solution
```

Fig. 6. Pseudocode for Genitor.

```
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.
```

Fig. 7. Pseudocode for the memetic algorithm.

individuals in the initial population and also on the offspring generated after crossover and mutation.

5.6. HereBoy evolutionary algorithm

HereBoy is an evolutionary algorithm that combines the features of GA and SA [31] (see Fig. 8). 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. Mutated chromosomes 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, based on an arbitrary order of the tasks. An adaptive mutation scheme is employed by the HereBoy heuristic. Mutation is applied by randomly selecting a task on the chromosome, and mapping it 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 percentage of tasks to be mutated during each iteration or the mutation rate (γ) (Eqs. (4) and (5)) is determined by two terms: (a) the maximum mutation rate, denoted by α, which is user defined fraction, and (b) the fraction β that reduces the number of tasks mutated as the current robustness approaches the upper bound (UB) value on the robustness metric (the UB calculation is described in Section 5.7). Mathematically, the fraction β is calculated based on the equation given

1. Generate an MCT mapping and evaluate the robustness metric.
2. Compute the percentage of tasks to mutate (γ) using equations (7) and (8).
3. Randomly select γ% of the tasks and mutate each task by assigning it to the machine that maximizes robustness metric.
4. Evaluate robustness metric of the current mapping.
5. Accept the new mapping if better or perform a probability test to accept poorer solutions.
6. For the current mapping, repeat steps 2 to 5 until 10^7 iterations.
7. Output the best solution.

Fig. 8. Pseudocode for the HereBoy evolutionary algorithm.

below:

$$\beta = \frac{(\mathrm{UB} - \rho_\mu(\varphi, C))}{\mathrm{UB}}, \tag{4}$$

$$\gamma = \alpha \times \beta. \tag{5}$$

The chromosome mapping solution is evaluated at the end of each mutation. A probabilistic 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 [11]. Typically *cooling schedules* are predefined, although it has been shown that adaptive schedules produce better results [31].

An adaptive scheme is employed by HereBoy to reduce the probability (η) of accepting a poorer solution. The probability is given by Eq. (6) that is similar to the adaptive mutation rate formula. The probability is the product of the user defined value maximum probability (π) and the fractional term β defined earlier. Notice that the probability of accepting poor solutions reduces as better solutions are produced:

$$\eta = \pi \times \beta. \tag{6}$$

As a result of experimentation, HereBoy is run with a 5% maximum mutation rate α and the maximum probability π is set to 1% for this problem. The stoping criterion for the heuristic is a total of 10^7 iterations.

5.7. Upper bound

The method developed for estimating an UB 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 MET of task i, denoted by MET_i, is given by the following equation:

$$MET_i = \min ETC(i, j) \text{ over all } j. \tag{7}$$

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

Let $N = \lfloor T/M \rfloor$. The first N tasks in the sorted order are stored in a list S. The total execution time any N tasks can have is greater than or equal to the sum of the execution times of the first N tasks. Thus, the UB for robustness is given by

$$\mathrm{UB} = \frac{\left(\tau - \sum_{i=0}^{|S|-1} MET_i\right)}{\sqrt{N}}. \tag{8}$$

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, $r_\mu(F_x, C) < \mathrm{UB}$. 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 Eq. (2), the finish 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 mathematical UB results in a loose UB. Furthermore, the greater the heterogeneity, the looser the bound.

5.8. Experimental results

The simulation results are shown in Figs. 9 and 10. All the heuristics are run for 50 different trials for the two cases of heterogeneities, and the average values and 95% confidence intervals [24] are plotted. The running times of the heuristics averaged over all trials, mapping 1024 tasks onto eight machines, are shown in Table 1.

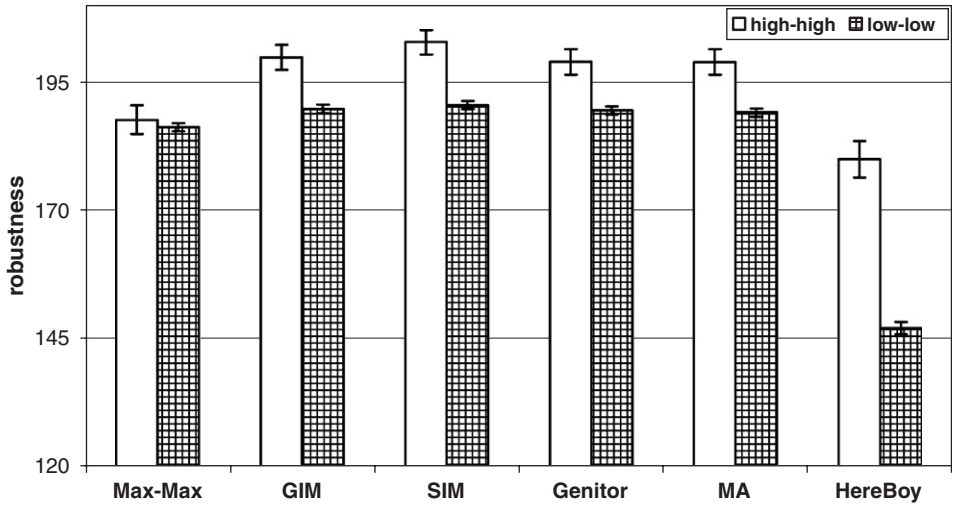


Fig. 9. The simulation results for robustness. The average UB values are 416.54 for high–high heterogeneity and 313.83 for low–low heterogeneity.

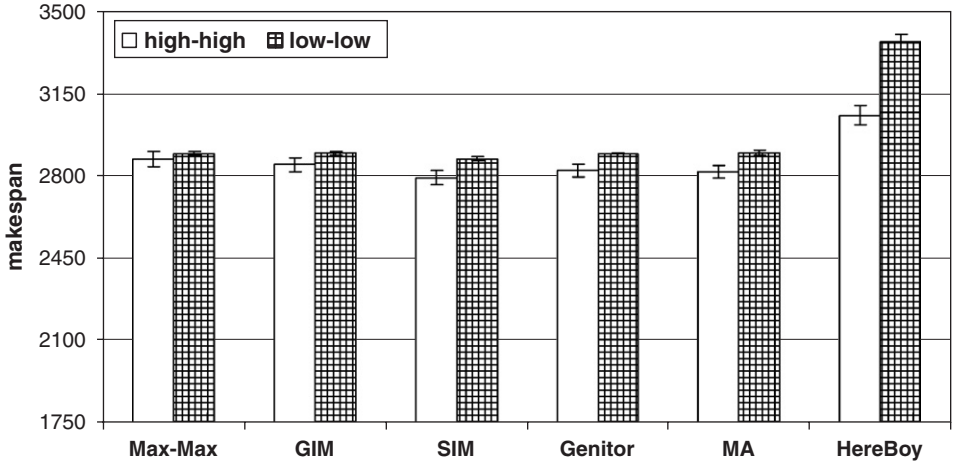


Fig. 10. The simulation results for makespan (the makespan constraint $\tau = 5000$).

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

Heuristic	Average execution times (s)
Max–Max	0.52
Greedy IM	3600
Sum IM	600
Genitor	3000
Memetic algorithm	3000
HereBoy	1200

The GIM and SIM are among the best heuristics in terms of robustness 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, ultimately surpassing the "best" swap variation solution. For "arbitrary order" and "best" swap variations, it was observed that, in general, the robustness of the initial mapping did not impact the robustness of the final mapping. The variation of the GIM heuristic that is seeded with the Max–Max solution is on average less by 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.

The Max–Max heuristic was the fastest among all the heuristics implemented for this research. 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. Genitor has less than 1% overall improvement in robustness after 6000 generations. Allowing Genitor to run more than 250,000 generations was observed to be insignificant. MA showed similar characteristics and was stopped after 100,000 iterations.

In previous work for a different problem domain [31], a HereBoy heuristic was able to outperform a GA and required much less execution time. However, for the problem specified here, HereBoy had the worst performance out of all the heuristics. In addition, allowing HereBoy to run for a longer time did not improve the solution. One possible cause may be that the UB used here is relatively loose (considering all the assumptions made in the derivation), and hence, the adaptive mutation technique that uses the UB value did not prove useful.

Another heuristic tried for this environment was ant colony optimization [36,38,45]. The ACO heuristic performed well for a related problem [38] although it is very time consuming to build good solutions. For our implementation (see [46] for the details), ACO was allowed to run for an hour and did not perform well. Hence, ACO is not included in this study.

Notice that for a similar makespan, the GIM heuristic showed 6% better robustness for the high–high 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. This is also demonstrated in [3], but there it is for random mappings.

6. Heuristics descriptions for the selected machine suite problem

6.1. Overview

This section describes six heuristics for the problem of selecting machines to comprise a suite. Five of the six heuristics studied for this problem, negative impact greedy iterative maximization, partition/merge greedy iterative maximization, selection Genitor, Max–Max memetic algorithm, and Max–Max Hereboy evolutionary algorithm, involve two phases. In phase 1, a subset of machines is selected using specific heuristic techniques to meet the cost and makespan constraints, and to maximize robustness. In phase 2, tasks are mapped to the set of machines found in phase 1 to further maximize the robustness metric for the mapping. The cost and robustness sum iterative maximization heuristic involves only one phase where a robustness maximization criterion is used to select machines such that the cost constraint is always satisfied. Throughout the description of the heuristics, Class 1 of machines is referred to as the highest class and Class 5 of machines is referred to as the lowest class.

6.2. Negative impact greedy iterative maximization

The negative impact greedy iterative maximization (NI-GIM) heuristic used here is a modification of GIM described in the fixed machine suite variation. The NI-GIM heuristic performs a Min–Min [23] mapping (procedure described in Fig. 3) based on the completion times assuming all machines to be available, irrespective of the cost constraint.

The robustness radius of all the available machines is calculated for the Min–Min mapping. The negative impact of removing machine j is determined in the following way. Each of the tasks mapped onto machine j is evaluated for reassignment to all the other machines. The decrease in the robustness radius of each available machine i if a task t is reassigned from machine j is calculated; call this $\Delta_{i,t}$. Let A be the set of available machines in the suite. The *negative impact* of removing machine j, denoted by NI_j, is given by

$$NI_j = \sum_{t \in \text{ tasks on } j} \frac{1}{|A|} \sum_{i=0}^{|A|-1} \Delta_{i,t}. \tag{9}$$

The ratio of negative impact to cost is obtained by simply dividing the negative impact by the cost of the machine j. The machine that has the least value of the negative impact to cost ratio is then removed from the set of available machines. The procedure of performing the Min–Min mapping with only the available machines and the ratio calculation to remove another machine is repeated until the cost constraint is satisfied.

For the set of machines determined above that meets the cost constraint, the GIM heuristic (please refer to Section 5.2) is run to determine a mapping that maximizes robustness for the given machine set.

6.3. Partition/merge greedy iterative maximization

The phase 1 of partition/merge greedy iterative maximization (P/M-GIM) starts with a random number of machines chosen from each class. The tasks are then mapped to the selected machines using the Min–Min heuristic of Fig. 3. The makespan for the Min–Min mapping is calculated. It was observed that the makespan constraint in this study is such that if the cost constraint is violated, the makespan constraint is always satisfied using Min–Min. Hence, either both of the constraints are satisfied or only one of the two constraints is violated using Min–Min. If the cost constraint is violated, then the *task-merge* (machine removal) [29] technique is executed. Otherwise, the *task-partition* (machine addition) [29] technique is executed to improve the makespan. Merging is stopped once the cost constraint is satisfied and partitioning is stopped if addition of another machine will violate the cost constraint.

Five different methods for partitioning and merging are implemented: (a) cheap, (b) expensive, (c) even distribution, (d) most common, and (e) random. In the *cheap* variation, the merge step removed a machine in the most expensive class, or the partition step added the cheapest available machine. The *expensive* variation did exactly the opposite (removed a cheapest machine or added the most expensive). *Even distribution* attempted to remove from the class that already had the most number of machines or to add to the class that had the least machines (ties were broken arbitrarily). The *most common* approach attempted to remove from the class that had the least number of machines or to add machines to the class that already had the most machines (ties were broken arbitrarily). The *random* variation simply involved partitioning or merging an available machine from a randomly selected class.

After generating a valid mapping that satisfies the cost and makespan constraints using one of the above techniques, phase 2 is entered, where reassignment and swapping of the

1. Begin with the CLB mapping.
2. Find the makespan machine (the machine that finishes last) for the current mapping.
    a. For each task on the makespan machine, find the machine that gives the maximum task-execution improvement.
    b. From the subset of these task/machine pairs that improve the makespan and meet the cost constraint, find the pair that gives the maximum task-execution improvement and assign the task to the paired machine.
3. Repeat step 2 until makespan $\leq \tau$ or no task can be relocated.
4. If makespan $\leq \tau$, go to step 7, otherwise go to step 5.
5. Find the makespan machine for the current mapping.
    a. For each task on the makespan machine, consider swapping it with a task on a different machine. Find the target task that gives the maximum task-execution improvement.
    b. From the subset of these target tasks, for each task on the makespan machine that improves the makespan, find the target task that gives the overall maximum task-execution improvement.
    c. Swap the task on the makespan machine with the selected target task.
6. Repeat step 5 until makespan $\leq \tau$. If this is not possible, the mapping procedure fails (for our study, this never happened).
7. Find the min-radius machine for the current mapping.
    a. For each task on the min-radius machine, find the machine that gives the maximum robustness improvement.
    b. From the subset of these task/machine pairs that improve the robustness and meet the cost constraint, find the pair that gives the maximum robustness improvement and assign the task to the paired machine.
8. Repeat step 7 until no task can be relocated.
9. Find the min-radius machine for the current mapping.
    a. For each task on the min-radius machine, consider swapping it with a task on a different machine. Find the target task that gives the maximum robustness improvement.
    b. From the subset of these target tasks, for each task on the min-radius machine that improves the makespan, find the target task that gives the overall maximum robustness improvement.
    c. Swap the task on the min-radius machine with the selected target task.
10. Repeat step 9 until no task can be swapped.

Fig. 11. Pseudocode for the CR-SIM heuristic.

GIM heuristic are executed in an attempt to improve the robustness metric of the mapping. The reassignment and swapping of the GIM heuristic is executed for 20 unique machine combinations (found using phase 1) and the best solution is output.

6.4. Cost and robustness sum iterative maximization

The cost and robustness sum iterative maximization (CR-SIM) heuristic (see Fig. 11) starts with a cost *lower bound* (CLB) mapping where all the tasks are mapped onto a single lowest cost machine (step 1). There cannot be a mapping that has a lower cost than the CLB mapping. However, because this mapping is not guaranteed to have a makespan less than τ, reassignment of some tasks to other machines (steps 2 and 3) may be necessary. It is assumed that all the machines are available for the reassignment of tasks. When a machine is used for the first time, the cost for using the machine is paid and the total cost of all the machines used in the suite must be less than δ. After the reassignment procedure, if τ is still violated, a task swapping procedure is executed (steps 5 and 6). A similar procedure is used to maximizing robustness (steps 7–10).

For this heuristic, the *task-execution improvement*, defined as the decrease in the sum of the completion times of the machines after reassignment or swapping, and the *robustness improvement*, defined as the increase in the sum of the robustness radius of the machines after reassignment or swapping, are maximized. Recall that the min-radius machine is defined as the machine with the smallest robustness radius.

A variation of this heuristic uses a predetermined set of minimum cost machines such that adding another machine will violate the cost constraint. For this set of lowest cost machines chosen that meets the cost constraint, relocations are made based on the task-execution or robustness improvement as before. For another variation, define *cost performance index* (CPI) of machine j as the product of the cost of machine j and the average ETC of all tasks on machine j. The machines with the lowest CPI are selected until the cost is less than or equal to δ for mapping tasks. For this machine set, the relocation and swapping are done as explained above.

6.5. Selection Genitor

In phase 1 of Selection Genitor (S-Genitor) a chromosome is a vector of length equal to the number of machine classes (five), where the ith element is the number of machines in the ith class. Phase 1 of Genitor operates on a fixed population of 100 chromosomes. The entire population is generated randomly such that the cost constraint is met. The chromosomes are evaluated using the robustness metric based on a machine assignment using the Max–Max mapping from Section 5.1. The entire population is sorted in descending order based on the robustness metric of the Max–Max heuristic.

In the crossover step, for the pair of the selected parent chromosomes (chosen with a linear bias function of 1.5), a random cut-off point is generated that divides the chromosomes into top and bottom parts. A new chromosome is formed using the top of one and bottom of another. An offspring is inserted in the population after evaluation only if the cost constraint is satisfied (the worst chromosomes of the population are discarded to maintain a population of only 100).

After each crossover, the linear bias function is applied again to select a chromosome for mutation. Two random classes are chosen for the chromosome and the mutation operator increments the number of machines of the first chosen class by one and decrements the number of machines of the other by one. If the chromosome is infeasible, that is, if it violates the cost constraint or the possible number of machines in each class, it is discarded. Otherwise, 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 phase 1 of S-Genitor. The heuristic stops when the criterion of 500 total iterations is met. The relatively small number of iterations was found experimentally to be sufficient for the solution space. The machine combination found from phase 1 is used in phase 2, which derives a mapping using this combination of machines to maximize robustness based on the Genitor implementation in Section 5.4 (a total of 100,000 iterations is used here to stop the phase 2 of Genitor).

6.6. Max–Max memetic algorithm

For the Max–Max memetic algorithm (MMMA) metaheuristic, in phase 1, 100 random combinations of machines from each class are chosen such that the cost constraint is satisfied. Each of the 100 combinations is evaluated using the Max–Max heuristic and the machine combination that has the highest robustness metric is selected. In phase 2, for the best machine combination found in phase 1, the MA heuristic identical to that described in Section 5.5 is executed, the only difference being the stopping criterion. A total of 40,000 iterations is used here in phase 2 of MA.

6.7. Max–Max HereBoy evolutionary algorithm

In Max–Max HereBoy (MM-HereBoy), phase 1 starts by adding one machine to each class (starting from the lowest class) in a round robin fashion until the cost constraint is violated. The current machine combination is evaluated using the robustness metric based on a machine assignment made by the Max–Max mapping of Section 5.1.

Now, starting from the highest class, a new machine is considered to be included in the existing machine set in a round robin fashion (unless no more machines from a particular class can be added). Adding another machine will violate the cost constraint. Hence, to be able to accommodate the inclusion of a machine, one or more machines from other classes should be removed. Machines are considered to be removed from a single class or from two different classes (this is sufficient to add a machine of any class). All such combinations are considered and if removing a particular combination of machines allows adding another machine of a lower class (after adding the higher class machine under consideration), then an additional machine is added. For each combination of machines that is removed, and replaced by other machines, a new set of working machines is formed. All machine sets are evaluated using the mapping produced by Max–Max and the set that gives the highest robustness metric is stored as the best. For the current best machine set, the above described procedure is repeated until addition of a machine from any class will not improve the robustness metric.

For the best combination of machines from the phase 1 procedure, the HereBoy evolutionary algorithm (see Section 5.6) is executed as phase 2 to determine the task to machine mapping for that combination of machines.

6.8. Upper bound

The UB on the robustness metric for this study is similar to that for the fixed machine suite problem variation. It assumes a homogeneous MET system. For the selected suite problem, there cannot be more than 33 machines in the system for the given cost constraint. This includes the 33 machines of the lowest class possible in the entire HC suite. Following Eq. (2) and our assumption of the homogeneous MET system, having more machines in the suite gives a better robustness metric than having fewer machines in the suite (due to the impact of number of tasks on each machine). Thus, a loose UB on robustness is Eq. (11) with $M = 33$.

6.9. Results

The simulation results are shown in Figs. 12 and 13. All the heuristics are run for 100 different high-low trials. The average values and 95% confidence intervals [24] are plotted. The running times of the heuristics averaged over 100 trials, mapping 1024 tasks in each trial, are shown in Table 2.

The S-Genitor and "cheap" variation of the P/M-GIM heuristic are the best among all the heuristics studied for this robustness maximization problem (the cheap variation is shown in the figures). Both of these heuristics, on average, had all of the available machines from Class 4 and 5. The "cheap" variation of the P/M-GIM heuristic always removed machines from Class 1 if the cost constraint was violated. But Genitor explored

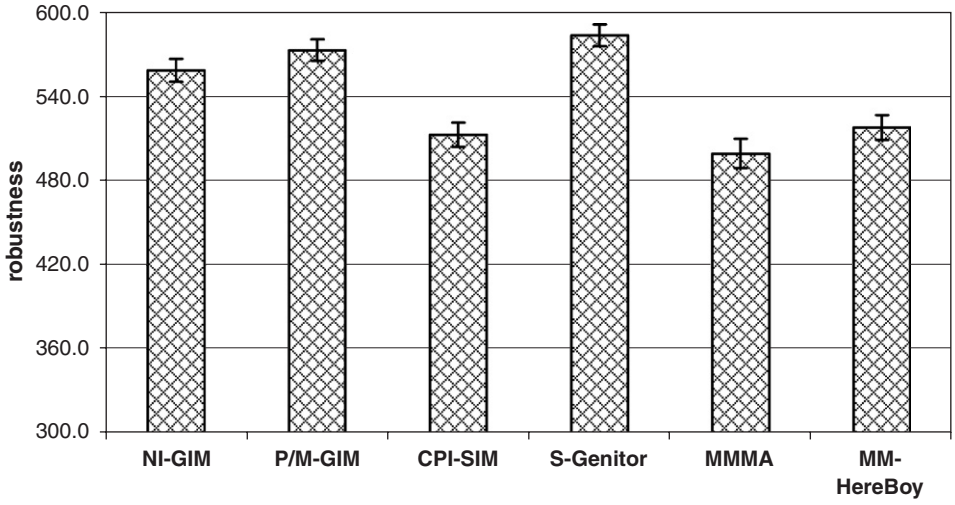


Fig. 12. The simulation results for robustness. The average UB value is 2019.3.

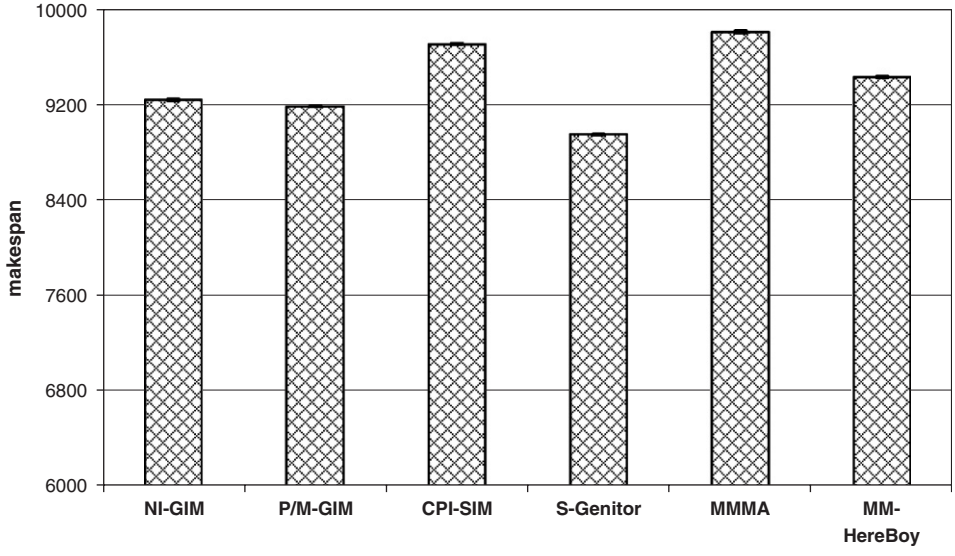


Fig. 13. The simulation results for makespan (the makespan constraint $\tau = 12,000$).

Table 2
The average execution times of the heuristics averaged over 100 trials (using a typical unloaded 3 GHz Intel Pentium 4 machine)

Heuristic	Average execution times (s)
NI-GIM	3600
P/M-GIM	3600
CPI-SIM	780
S-Genitor	3420
MMMA	3000
MM-HereBoy	1380

the search space more generally and on average used more machines in Class 1 than in Class 2. The "most common" and "random" variations of P/M-GIM heuristic were within 10% of the "cheap" variation. The "expansive" variation performed the worst among all the variations of P/M-GIM and "even distribution" was slightly better than the "expensive" variation. These two variations did not have as many machines in the suite as compared to the other variations. For this problem, having a good balance between the execution speed of machines and the number of machines in the HC suite proved to be important for maximizing the robustness of the mapping.

The NI-GIM heuristic performed slightly worse (on average) than P/M-GIM. The negative impact calculation always forced removal of machines from either Class 2 or 3. All machines from Class 1, 4, and 5 (i.e., the fastest class and the two cheapest classes of machines) were used in more than 90% of the trails.

The CR-SIM heuristic by itself did not perform well (an average of 252 for the robustness metric across 100 trials). The poor performance is because it always selected machines for relocation that will maximize task-execution or robustness

improvement. Therefore, CR-SIM typically picked machines in the order of the highest class to the lowest. The CR-SIM heuristic does not consider replacing a fast machine with multiple slower machines. The CPI variation of CR-SIM (CPI-SIM) performed within 12% of S-Genitor. The lowest cost variation also performed similarly and is within 2% of the CPI-SIM variation.

The robustness metric of the MM-HereBoy is within 12% of S-Genitor. The search technique used for selecting the machines for HereBoy used all of the machines of Class 1, 4, and 5.

The MMMA heuristic that made use of the random search approach to find the set of machines in phase 1 performed the worst among all the heuristics. The MA optimization heuristic has proved to work well for a similar environment in Section 4. However, the machine selection by the random approach proved to be ineffective for this kind of an environment.

The SIM heuristic performed well for the fixed suite problem, where inconsistent heterogeneity between machines is considered. However, due to consistent heterogeneity considered in the selected suite study, the sum of the task-execution or robustness improvement of machines did not help to find a good solution. The phase 2 of the other heuristics discussed in this research is similar to the heuristics studied in Section 4. The GIM heuristic performed well here because it focused on maximizing the robustness metric itself, unlike CR-SIM. The discussion on the performance of phase 2 of S-Genitor, MMMA, and MM-HereBoy are similar to those discussed in Section 4.

7. Summary

Two variations of robust mapping of independent tasks to machines were studied in this research. In the fixed suite variation, six static heuristics were presented that will maximize the robustness of a mapping against errors in the ETC when a set of machines was given. The best robustness metric was obtained by using the SIM heuristic. The GIM, GA, and MA performed comparably with their robustness metric within 2% of the SIM. However, the execution times for the heuristics themselves were much higher as compared to the SIM heuristic. Thus, SIM is a good choice for the fixed suite problem.

This study also presented six static heuristics for selecting a set of machines, under a given dollar cost constraint that will maximize the robustness of a mapping against errors in the ETC. The best average robustness metric was obtained by using the S-Genitor heuristic. The P/M-GIM heuristic performed comparably with its robustness metric within 2% of S-Genitor. The execution times for both of the heuristics themselves were also comparable. Thus, both S-Genitor and P/M-GIM are a good choice for the selected suite problem variation. In this study, a suite of at most 33 machines from five classes were used to execute 1024 tasks. Future work could include examining bigger scenarios, where all of the above parameters are larger.

Acknowledgments

Preliminary portions of this material were presented at the 14th IEEE Heterogeneous Computing Workshop (HCW 2005) and at the 4th International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks (HeteroPar 2005). The authors thank Shoukat Ali, Adele Howe, and Jay Smith for their valuable comments. This research was supported by NSF under grand No. CNS-0615170, 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), by the DARPA Information Exploitation Office under Contract no. NBCHC030137, and by the Colorado State University George T. Abell Endowment. Approved for public release; distribution unlimited.

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, B. Yao, Characterizing resource allocation heuristics for heterogeneous computing systems, in: A.R. Hurson (Ed.), Advances in Computers, vol. 63: Parallel, Distributed, and Pervasive Computing, Elsevier, Amsterdam, The Netherlands, 2005, pp. 91–128.

[2] S. Ali, J.-K. Kim, H.J. Siegel, A.A. Maciejewski, Y. Yu, S.B. Gundala, S. Gertphol, V. Prasanna, Utilization-based techniques for statically mapping heterogeneous applications onto the HiPer-D heterogeneous computing system, Parallel and Distributed Computing Practices, Special Issue on Parallel Numeric Algorithms on Faster Computers vol. 5 (4) (December 2002).

[3] S. Ali, A.A. Maciejewski, H.J. Siegel, J.-K. Kim, Measuring the robustness of a resource allocation, IEEE Trans. Parallel Distrib. Systems 15 (7) (July 2004) 630–641.

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

[5] S. Ali, H.J. Siegel, M. Maheswaran, D. Hensgen, S. Ali, Representing task and machine heterogeneities for heterogeneous computing systems, Special 50th Anniversary Issue, Tamkang J. Sci. Eng. 3 (3) (November 2000) 195–207 (invited).

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

[7] H. Barada, S.M. Sait, N. Baig, Task matching and scheduling in heterogeneous systems using simulated evolution, in: 10th IEEE Heterogeneous Computing Workshop (HCW 2001), in the Proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS 2001), April 2001.

[8] I. Banicescu, V. Velusamy, Performance of scheduling scientific applications with adaptive weighted factoring, in: 10th IEEE Heterogeneous Computing Workshop (HCW 2001), in the Proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS 2001), April 2001.

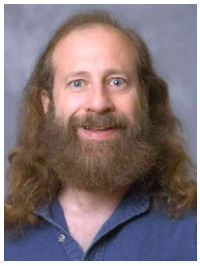
[9] L. Bölöni, D.C. Marinescu, Robust scheduling of metaprograms, J. Scheduling 5 (5) (September 2002) 395–412.

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

[11] 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, B. Yao, A comparison of eleven static heuristics for mapping a class of independent

tasks onto heterogeneous distributed computing systems, J. Parallel Distrib. Comput. 61 (6) (June 2001) 810–837.
[12] E.G. Coffman Jr., (Ed.), Computer and Job-Shop Scheduling Theory, Wiley, New York, 1976.
[13] R.L. Daniels, J.E. Carrilo, ß-Robust scheduling for single-machine systems with uncertain processing times, IIE Trans. 29 (11) (November 1997) 977–985.
[14] A.J. Davenport, C. Gefflot, J.C. Beck, Slack-based techniques for robust schedules, in: Sixth European Conference on Planning, September 2001, pp. 7–18.
[15] R.P. Dick, N.K. Jha, MOGAC: a multiobjective genetic algorithm for the co-synthesis of hardware–software embedded systems, IEEE Trans. Comput.-Aided Design 17 (10) (October 1998) 920–935.
[16] J. Dorn, M. Girsch, G. Skele, W. Slany, Comparison of iterative improvement techniques for schedule optimization, European J. Oper. Res. 94 (2) (October 1996) 349–361.
[17] J. Dorn, R.M. Kerr, G. Thalhammer, Reactive scheduling: improving the robustness of schedules and restricting the effects of shop floor disturbances by fuzzy reasoning, Internat. J. Human–Comput. Stud. 42 (6) (June 1995) 687–704.
[18] M.M. Eshaghian (Ed.), Heterogeneous Computing, Artech House, Norwood, MA, 1996.
[19] D. Fernandez-Baca, Allocating modules to processors in a distributed system, IEEE Trans. Software Eng. SE-15 (11) (November 1989) 1427–1436.
[20] I. Foster, C. Kesselman (Eds.), The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, San Fransisco, CA, 1999.
[21] R.F. Freund, H.J. Siegel, Heterogeneous processing, IEEE Comput. 26 (6) (June 1993) 13–17.
[22] A. Ghafoor, J. Yang, A distributed heterogeneous supercomputing management system, IEEE Comput. 26 (6) (June 1993) 78–86.
[23] O.H. Ibarra, C.E. Kim, Heuristic algorithms for scheduling independent tasks on non-identical processors, J. ACM. 24 (2) (April 1977) 280–289.
[24] R. Jain, The Art of Computer Systems Performance Analysis Techniques for Experimental Design, Measurement, Simulation, and Modeling, Wiley, New York, 1991.
[25] M. Kafil, I. Ahmad, Optimal task assignment in heterogeneous distributed computing systems, IEEE Concurrency 6 (3) (July 1998) 42–51.
[26] A. Khokhar, V.K. Prasanna, M.E. Shaaban, C. Wang, Heterogeneous computing: challenges and opportunities, IEEE Comput. 26 (6) (June 1993) 18–27.
[27] P. Kouvelis, R. Daniels, G. Vairaktarakis, Robust scheduling of a two-machine flow shop with uncertain processing times, IIE Trans. 38 (5) (May 2000) 421–432.
[28] P. Kouvelis, G. Yu, Robust Discrete Optimization and its Applications, Kluwer Academic Publisher, Dordrecht, 1997.
[29] S.M. Kroumba, G. Bois, Y. Savaria, A synthesis approach for the generation of parallel architectures, in: 37th Midwest Symposium on Circuits and Systems, vol. 1, 3–5 August 1994, pp. 323–326.
[30] Y.-K. Kwok, A.A. Maciejewski, H.J. Siegel, I. Ahmad, A. Ghafoor, A semi-static approach to mapping dynamic iterative tasks onto heterogeneous computing systems, J. Parallel Distrib. Comput. 66 (1) (January 2006) 77–98.
[31] D. Levi, Hereboy: a fast evolutionary algorithm, in: Second NASA/DoD Workshop on Evolvable Hardware (EH '00), July 2000, pp. 17–24.
[32] V.J. Leon, S.D. Wu, R.H. Storer, Robustness measures and robust scheduling for job shops, IIE Trans. 26 (5) (September 1994) 32–43.
[33] M. Maheswaran, S. Ali, H.J. Siegel, D. Hensgen, R.F. Freund, Dynamic mapping of a class of independent tasks onto heterogeneous computing systems, J. Parallel Distrib. Comput. 59 (2) (November 1999) 107–121.
[34] M. Maheswaran, T.D. Braun, H.J. Siegel, Heterogeneous distributed computing, in: J.G. Webster (Ed.), Encyclopedia of Electrical and Electronics Engineering, vol. 8, Wiley, New York, 1999, pp. 679–690.
[35] 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.
[36] G.C. Onwubolu, B.V. Babu, New Optimization Techniques in Engineering, Springer, New York, 2004.
[37] A. Rae, S. Parameswaran, Application-specific heterogeneous multiprocessor synthesis using differential-evolution, in: 11th International Symposium on System Synthesis, December 1998, pp. 83–88.
[38] G. Ritchie, J. Levine, A hybrid ant algorithm for scheduling independent jobs in heterogeneous computing environments, in: Third Workshop of the UK Planning and Scheduling Special Interest Group (PLANSIG 2004), December 2004.
[39] M. Sevaux, K. Sörensen, Genetic algorithm for robust schedules, in: Eighth International Workshop on Project Management and Scheduling (PMS 2002), April 2002, pp. 330–333.
[40] V. Shestak, E.K.P. Chong, A.A. Maciejewski, H.J. Siegel, L. Benmohamed, I.J. Wang, R. Daley, Resource allocation for periodic applications in a shipboard environment, in: 14th IEEE Heterogeneous Computing Workshop (HCW 2005), in the Proceedings of the 19th International Parallel and Distributed Processing Symposium (IPDPS 2005), April 2005, pp. 122–127.
[41] S. Shivle, P. Sugavanam, 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, J. Velazco, Mapping of subtasks with multiple versions on an ad hoc grid environment, Parallel Computing, Special Issue on Heterogeneous Computing vol. 31 (7) (July 2005) 671–690.
[42] S. Shivle, H.J. Siegel, A.A. Maciejewski, P. Sugavanam, T. Banka, R. Castain, K. Chindam, S. Dussinger, P. Pichumani, P. Satyasekaran, W. Saylor, D. Sendek, J. Sousa, J. Sridharan, J. Velazco, Static allocation of resources to communicating subtasks in a heterogeneous ad hoc grid environment, Special Issue on Algorithms for Wireless and Ad-hoc Networks, J. Parallel Distrib. Comput. 66 (4) (April 2006) 600–611.
[43] G.F. Simmons, Calculus with Analytic Geometry, second ed., McGraw-Hill, New York, 1995.
[44] Y.N. Sotskov, V.S. Tanaev, F. Werner, Stability radius of an optimal schedule: a survey and recent developments, Industrial Appl. Combinat. Optimiz. 16 (1998) 72–108.
[45] T. Stützle, H. Hoos, Max–min ant system, Future Generation Comput. Syst. 16 (8) (2000) 889–914.
[46] P. Sugavanam, Robust resource allocation of independent tasks and resource allocation for communicating subtasks on ad hoc grids, Masters Thesis, Electrical and Computer Engineering, Colorado State University, 2005.
[47] L. Wang, H.J. Siegel, V.P. Roychowdhury, A.A. Maciejewski, Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach, J. Parallel Distrib. Comput. 47 (1) (November 1997) 8–22.
[48] D. Whitley, The GENITOR algorithm and selective pressure: Why rank based allocation of reproductive trials is best, in: Third International Conference on Genetic Algorithms, June 1989, pp. 116–121.
[49] M.-Y. Wu, W. Shu, H. Zhang, Segmented min–min: a static mapping algorithm for meta-tasks on heterogeneous computing systems, in: Nineth IEEE Heterogeneous Computing Workshop (HCW 2000), May 2000, pp. 375–385.

Prasanna Sugavanam received his M.S. degree in Electrical and Computer Engineering from Colorado State University in 2005, where he was a Graduate Research Assistant. He received his Bachelor of Engineering in Electrical and Electronics from Kumaraguru College of Technology, India in 2001. He is currently working as a senior software developer for a leading biotech company in California.

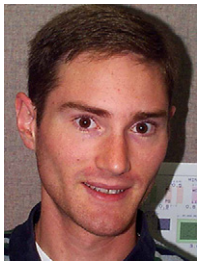


H.J. Siegel was appointed the George T. Abell Endowed Chair Distinguished Professor of Electrical and Computer Engineering at Colorado State University (CSU) in August 2001, where he is also a Professor of Computer Science. In December 2002, he became the first Director of the university-wide CSU Information Science and Technology Center (ISTeC). From 1976 to 2001, he was a professor at Purdue University. He received two B.S. degrees from MIT, and the MA, M.S.E., and Ph.D. degrees from Princeton University. Prof. Siegel has co-authored over 300 published papers on parallel and distributed computing and communication. He is a Fellow of the IEEE Fellow and a Fellow of the ACM. He was a Coeditor-in-Chief of the Journal of Parallel and Distributed Computing, and was 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 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 from Ohio State University in 1982, 1984, and 1987. From 1988 to 2001, he was a professor of Electrical and Computer Engineering at Purdue University, West Lafayette. He is currently the Department Head of Electrical and Computer Engineering at Colorado State University. Tony is a Fellow of the IEEE. A complete vita is available at: www.engr.colostate.edu/∼aam.

Mohana Oltikar is pursuing her M.S. degree in Electrical and Computer Engineering at Colorado State University, where she is currently a Graduate 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.

Ashish Mehta 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 Electronics from University of Mumbai, India. His research interests include resource management in distributed computing systems, computer architecture, computer networks, and embedded 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.

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

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 3 years in industry as a network engineer working for CISCO, Russia. He received his B.S. 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.

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.

Yogish G. Krishnamurthy graduated from the Department of Computer Science at Colorado State University, where he received his Masters in Computer Science in December 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.

Syed Amjad Ali is currently a graduate student at Colorado State University pursuing his Master's in Computer Information Systems. He received his Bachelors in Computer Science and Engineering from Dr. Babasaheb Ambedkar Marathwada University, Maharashtra, India. He is involved in a project with US Navy for implementing a real time information retrieval system in Internet relay chat servers. He was also involved with IBM in setting up a grid at Colorado State University's College of Business. He manages the Apple and Linux clusters at Atmospheric Science Department at CSU. His research interests include heterogeneous systems, parallel computing, grid computing, and information retrieval algorithms.

Junxing Zhang is 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. He has publications in the areas of distributed and heterogeneous computing, data management systems, and formal verification. His current research focuses on computer networking, especially wide area network measurement, characterization, and modeling.

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.

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 University, 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 fields of overlay networks, transport protocols, sensor networks, and distributed computing.

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.

Michael Raskey received a B.S. in Electrical Engineering from Valparaiso University in 2001, and a M.S. in Electrical Engineering from Colorado State University in 2005. He is currently employed by Hewlett-Packard Company in Fort Collins, Colorado, as a systems/software engineer.

Alan Pippin is currently pursuing an M.S. degree in Electrical Engineering at Colorado State University. He received his Bachelors in Electrical and Computer Engineering from Brigham Young University in 2001. He is currently employed in the systems-VLSI lab at Hewlett-Packard as a control and verification engineer working on core chipsets for enterprise computing systems. He is a member of IEEE.