Static Mapping Heuristics for Tasks with Dependencies, Priorities, Deadlines, and Multiple Versions in Heterogeneous Environments

Tracy D. Braun[†]
[†]Noemix
1425 Russ Blvd. Ste. T-110
San Diego, CA 92101-4717 USA
tdbraun@noemix.com

Howard Jay Siegel[‡*] and Anthony A. Maciejewski[‡]
[‡]Electrical and Computer Engineering Department
[*]Computer Science Department
Colorado State University
Fort Collins, CO 80523-1373 USA
{hj, aam}@colostate.edu

Abstract

Heterogeneous computing (HC) environments composed of interconnected machines with varied computational capabilities are well suited to meet the computational demands of large, diverse groups of tasks. The problem of mapping (defined as matching and scheduling) these tasks onto the machines of a distributed HC environment has been shown, in general, to be NP-complete. Therefore, the development of heuristic techniques to find near-optimal solutions is required. In the HC environment investigated, tasks had deadlines, priorities, multiple versions, and may be composed of communicating subtasks. The best static (off-line) techniques from some previous studies were adapted and applied to this mapping problem: a genetic algorithm (GA), a GENITOR-style algorithm, and a greedy Min-min technique. Simulation studies compared the performance of these heuristics in several overloaded scenarios, i.e., not all tasks executed. The performance measure used was a sum of weighted priorities of tasks that completed before their deadline, adjusted based on the version of the task used. It is shown that for the cases studied here, the GENITOR technique found the best results, but the faster Min-min approach also performed very well.

1. Introduction

Mixed-machine heterogeneous computing (HC) environments utilize a distributed suite of different machines, interconnected with high-speed links, to perform different computationally intensive tasks with diverse computational requirements (e.g., [2, 20, 22]). Such an environment coordinates the execution of tasks on machines within the system to exploit different capabilities (e.g., clock speeds, instruction sets, and memory sizes) and achieve increased performance. An HC system could also be part of a larger computational grid [14].

The act of assigning (matching) each task to a machine and ordering (scheduling) the execution of the tasks on each machine is key to coordinating and exploiting an HC system to its fullest extent. This matching and scheduling is known as mapping [8]. The general mapping problem has been shown to be NP-complete (e.g., [9, 16]).

In this study, tasks may be atomic or decomposable. Atomic tasks have no internal communications. Decomposable tasks consist of two or more communicating subtasks. Subtasks have data dependencies, but can be mapped to different machines. Tasks also have deadlines, priorities, and multiple versions. These additional task characteristics are common in military environments (e.g., [1, 10, 29]); however, they also add to the complexity of the HC mapping problem.

Three static (off-line) techniques were selected, adapted, and applied to this mapping problem: a greedy method (Min-min), an evolutionary method (a standard genetic algorithm (GA)), and the GENITOR approach [30]. The former two methods were also considered in a previous study and performed well [6]. This simulation study makes the following contributions:

- A new HC paradigm was used. Tasks had deadlines, priorities, multiple versions, and may have had subtasks. Multiple overloaded scenarios were considered, i.e., not all tasks met their deadline.
- Several heuristics were developed, adapted, and applied to this version of the mapping problem.
- Customized chromosome structures and operations were developed for the GA and GENITOR.
- The results show that GENITOR found the best solutions, but Min-min performed very competitively, and ran in less time.

Section 2 describes the details of the HC environment. The mapping heuristics are defined in Section

This research was supported in part by the DARPA/ITO Quorum Program under GSA subcontract number GS09K99BH0250 and a Purdue University Dean of Engineering Donnan Scholarship. Some of the equipment used was donated by Intel, Microsoft, and Noemix.

0-7695-1573-8/02/$17.00 (C) 2002 IEEE

3. In Section 4, the results from the simulation studies are examined. Section 5 summarizes this research.

2. HC Environment

Static mapping is performed when tasks are mapped in an off-line planning phase, e.g., planning the mapping for tomorrow in a production environment. Static mapping is also used in "what-if" predictive studies, e.g., justifying a new machine purchase *a priori*. The development of heuristic techniques to find near-optimal mappings is an active area of research, e.g., [6, 7, 13, 19, 28, 29, 31].

The static mapping heuristics in this study were evaluated using simulation experiments. It was assumed that an estimate of the execution time for each task on each machine was known *a priori* and contained within an *ETC* (expected time to compute) matrix [3, 4]. This is a common assumption when studying mapping heuristics (e.g., [18, 23]); approaches for doing this estimation are discussed in [20, 22].

Each task t_i had one of four possible weighted priorities, p_i. Two different priority scenarios were investigated, high-weighted priorities, $p_i \in \{1, 4, 16, 64\}$, and low-weighted priorities, $p_i \in \{1, 2, 4, 8\}$. Each value was equally likely to be assigned, where 64 or 8 represented the most important tasks.

This research assumed an oversubscribed system, i.e., not all tasks could finish by their deadline. To model this, let the arrival time for task t_i be denoted a_i, and let the deadline for task t_i be denoted d_i. To simulate different levels of oversubscription, two different arrival rates were used, based on a Poisson distribution [17]. For high-arrival rates, the average arrival rate was 0.150 tasks per (arbitrary) unit of time and for moderate-arrival rates the average was 0.075.

Deadlines were assigned to each task as follows. First, for task t_i, the median execution time of the task, med_i, for all machines was found. Next, each task t_i was randomly assigned a deadline factor, δ_i, where $\delta_i \in \{1, 2, 3, 4\}$ (each value was equally likely to be assigned). Finally, the deadline for task t_i, d_i, was assigned as $d_i = (\delta_i \times med_i) + a_i$. All the subtasks within a task had just one deadline and one arrival time – those assigned to the task. A deadline achievement function, D_i, was also defined. Based on the mapping used, $D_i = 1$ if t_i finished at or before d_i, otherwise $D_i = 0$.

In this work, each task had three versions. At most one version of any task was executed, with version v_k always preferred over version v_{k+1}, but also requiring more execution time. Each version of a task (or subtask) had the same dependencies, priority, and deadline. The execution time for each task (or subtask) and the user preference for that version (defined shortly) were the only parameters that varied among different versions.

The estimated execution time for task t_i, on machine m_j, using version v_k is denoted $ETC(i, j, k)$. Thus, based on the previous assumption, $ETC(i, j, 0) > ETC(i, j, 1) > ETC(i, j, 2)$. The HC environment in the simulation study had $M = 8$ machines and $V = 3$ versions. To generate simulated execution times for version v_0 of each task in the ETC matrix, the coefficient-of-variation-based (CVB) method from [3] was used with means of 1000, and coefficients of variation of 0.6. Times for version v_{k+1} of t_i are randomly assigned values of 50% to 90% of $ETC(i, j, k)$. These parameters were based on previous research, experience in the field, and feedback from colleagues.

Task versions of lower preference were considered because, with their reduced execution times, they might have been the only version possible to execute. Let r_{ik} be the normalized user-defined preference for version v_k of task t_i, and let $U(w, x)$ be a uniform random (floating point) number greater than w and less than x. For the simulation studies: $r_{i0} = 1$ (most preferred), $r_{i1} = r_{i0} \times U(0, 1)$ (medially preferred), and $r_{i2} = r_{i1} \times U(0, 1)$ (least preferred). All subtasks of a task were required to use the same version.

The size and structure of the subtasks within a decomposable task were generated randomly, with between two and five subtasks per task, each with a maximum fanout of two. Subtask data transfer times were also generated using the CVB method [3], taking 5% to 12% of the average subtask execution time. The machines were assumed to be connected via a high-speed network hub (one input and one output port); subtask data transfers were scheduled and not multiplexed.

Atomic tasks and subtasks are called m-tasks (mappable tasks). The number of m-tasks to map in the simulation study was $T = 2000$, divided randomly into approximately 1000 atomic m-tasks and 1000 subtasks. Thus, there were approximately 1000 atomic tasks and 285 decomposable tasks. For evaluation purposes, there were $T_{eval} \approx 1285$ tasks. More details about the generation of the HC environment are in [7].

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

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

0-7695-1573-8/02/$17.00 (C) 2002 IEEE

Higher E values represent better mappings. Based on the definitions of d_i and r_{ik}, the upper bound (UB) of E is $UB = \sum_{i=0}^{T_{eval}-1} p_i$.

To determine D_i above, the task's completion time had to be determined. An m-task could note begin executing until after its arrival time (and after all input data had been received, if necessary). Let the machine availability, $mav(i,j,k)$, be the earliest time (after m-task t_i's arrival time) at which machine m_j (1) was not reserved, (2) could receive all subtask input data, and (3) was available for a long enough interval to execute version v_k of t_i. Then, the completion time of m-task t_i, version v_k, on machine m_j, denoted $ct(i,j,k)$, is $ct(i,j,k) = mav(i,j,k) + ETC(i,j,k)$.

In addition to the upper bound, UB, a second, tighter upper bound, TUB was also used. TUB is based on the concept of value per unit time of task t_i, $VT_i = \max_{(0 \le j < M,\ 0 \le k < V)} (p_i \times r_{ik})/ETC(i,j,k)$. TUB sorted the tasks by VT_i, and then allocated time for tasks with the highest VT_i first. By considering the most valuable tasks first, an upper bound was achieved. Let ϵ be the arrival time of the last arriving task, and the end of the simulation interval. TUB did not assign tasks to machines, but they were allocated time from 0 to $\mathcal{E} = M \times \epsilon$, based on the execution time of the machine and version in VT_i. If a task was allocated time, then $D_i = 1$ for E. To achieve this upper bound, TUB was based on unrealistic assumptions, e.g., there were no subtask communications and all $d_i = \mathcal{E}$.

3. Mapping Heuristics

3.1. Greedy Mapping Heuristics

As an approximate lower bound, consider a greedy FIFO technique, referred to as the Minimum Current Fitness (MCF) technique. MCF considered m-tasks for mapping in ascending order of arrival time; it mapped m-tasks to the machine that could complete the best (lowest) version possible by that task's deadline. If no machine/version combination could complete the m-task before its deadline, the task was not mapped.

A two-phase greedy technique, Min-min, performed well in many situations (e.g., [6, 16, 19, 31]). The Min-min heuristic was adapted and applied to this mapping problem. To describe Min-min, it is useful to define f_i, the task fitness for m-task t_i, $f_i = -(D_i \times p_i \times r_{ik})$, where t_i is executed using version v_k. The task fitness f_i represents the negative of the contribution of each task to E.

Let U be the set of all unmapped m-tasks. Let $UP \subset U$ consist of all unmapped subtasks whose predecessors have been mapped and all unmapped atomic tasks. The first phase of Min-min found the best machine (i.e., minimum f_i) for each m-task in UP, and then stored these m-task/machine pairs in the set CT. The m-tasks within CT are referred to as candidate tasks. If two machines gave the same best f_i, the one with the minimum completion time was used.

Phase two of Min-min selected the candidate task with the minimum f_i over all CT, and mapped this m-task to its corresponding machine. If two candidate tasks had the same minimum f_i, the one with the minimum execution time was used. This task was then removed from U. Phases one and two were repeated until all m-tasks were mapped (or removed because they could not meet their deadline). Several other variations of the Min-min heuristic were also investigated [7].

3.2. Generational Genetic Algorithm

The steps of a general genetic algorithm (GA) are in Figure 1. One iteration of the loop in Figure 1 is considered one generation, i.e., life-cycle. The approach described in this subsection will be referred to as a generational GA, as it may replace the entire population at each generation [15]. This is in contrast to a steady-state GA [25], where one member of the population is replaced at a time, e.g., GENITOR, presented in the next subsection.

```
initial population generation;
evaluation;
while (stopping criteria not met)
    selection;
    crossover;
    mutation;
    evaluation;
end while
output best solution;
```

Figure 1. General GA procedure, based on [24].

In GAs, a chromosome is used to represent a solution to the given problem (in this case, a mapping). The chromosome structure implemented for this study was composed of two data structures, the mapping table and the version vector. Figure 2 shows an example of this chromosome structure. The mapping table stored matching, scheduling, and subtask dependency information. In the version vector, if $V[i] = k$, then version v_k was used for task t_i. The GA used a population size, P, of 100 chromosomes.

The mapping table is a dynamic array structure; each row could change in length as necessary. The number of rows was M. Each column, or index position,

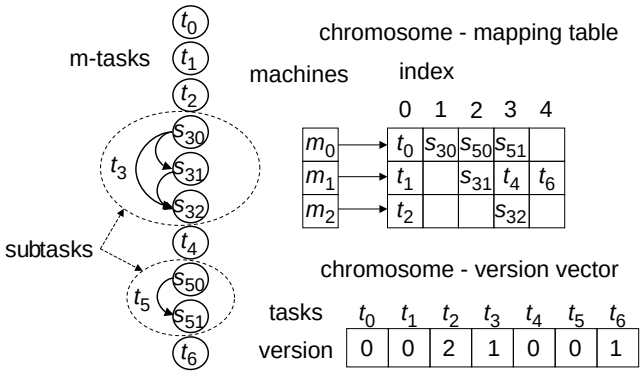

Figure 2. Example chromosome structure for a seven-task, three-machine mapping.

represented the order in which the m-tasks on each machine were to be considered for scheduling (but not necessarily the order in which they executed). Empty positions were used to maintain subtask dependencies.

A task in row m_j of the mapping table was matched to machine m_j. To obey data dependency constraints, a subtask's ancestors always have a lower index in the same or different row, and a subtask's descendents have a higher index.

Scheduling information was indirectly represented by the columns of the mapping table. The m-tasks were considered for actual scheduling on the machines (and network, if necessary) beginning from index 0, machine m_0, and then going top to bottom, left to right. That is, for column i, m-tasks were considered in order from m_0 to m_{M-1}, then m-tasks in column $i+1$ were considered, and so on. The m-task positions in the actual schedule (computed whenever the chromosome was evaluated) could differ from their positions in the mapping table (e.g., because of different arrival times).

Initial population generation. The initial population was generated using one chromosome that was the Min-min mapping and 99 random solutions (valid mappings). The GA executed a minimum of four times, and conditionally up to eight times, unless the same overall best solution was found again. The overall best mapping found was used as the final solution.

Evaluation. A chromosome's fitness refers to the quality of the solution, i.e., mapping, in the chromosome. A higher quality solution has a higher fitness value. The fitness function used was E.

Selection. After evaluation, selection occurs. Stochastic universal sampling (SUS) [5] was used for the selection phase of the GA. Chromosomes in the population were sorted (ranked) based on their fitness value. If c_i was the ith chromosome in the population, then c_0 was the most fit chromosome, and c_{P-1} was the least fit (ties were broken arbitrarily).

Each chromosome was then allocated a sector of a roulette wheel. The size of the sector is based on the fitness, i.e., more fit chromosomes will have larger sectors. Let A_i be the angle of the sector for chromosome c_i. Then A_0 (the sector for the most fit chromosome) has the largest angle, and A_{P-1} (the sector for the least fit chromosome) has the smallest angle. Let R, the ratio of two adjacent sector angles, be a constant (greater than 1). Then $R = \frac{A_i}{A_{i+1}}$ with $0 \leq i < P-1$. Given R, the angles for each sector can be explicitly stated in terms of the smallest sector $A_i = R^{(P-1)-i}A_{P-1}$ where $0 \leq i < P-1$. The sum of all angles for the roulette wheel, normalized to one, can then be computed as

$$\sum_{i=0}^{P-1} A_i = R^{(P-1)}A_{P-1} \sum_{i=0}^{P-1} \frac{1}{R^i} = 1, \qquad (2)$$

with $A_i = \frac{1}{R^i}A_0$. The value used for R was $R = 1 + 1/P$. For $P = 100$, this approximately gives the ratio of 1.5 between the best sector and median sector in the roulette wheel, as was used in [27, 30].

SUS then generated P pointers around the circumference of the roulette wheel, each $1/P$ apart. (Recall the angles around the roulette wheel were normalized to one, and $P \times (1/P) = 1$.) The chromosomes represented by the sectors on which pointers landed were then included in the next generation. The first pointer was generated from a random (floating point) number between 0 and $1/P$, inclusive, $x \in U[0, 1/P]$. This simulates a spin of the roulette wheel, where x was a random distance around the circumference of the first sector. Starting from x, the additional $P-1$ pointers were then placed, each $1/P$ apart.

Elitism, which guarantees that the the best solution found by the GA always remains in the population, was also employed [12]. Therefore, the quality of the solutions found was monotonically non-decreasing.

Crossover. Crossover combines elements of two parent chromosomes to generate a new offspring chromosome. To reduce the execution time of the GA, but not limit the range of possible offspring, a modification called gyre crossover was implemented. It is a form of single offspring crossover [21]. A brood of size $\mathcal{B}$ is a group of chromosomes selected from the entire population at random that was used for breeding. Each chromosome could be selected only once. Next, a random segment in each chromosome was selected. Then, in a gyral fashion, each member of the brood imposed the selected portion of itself on its neighbor chromosome in the brood. The result was $\mathcal{B}$ new offspring that replaced the parents in the population.

Let P_c be the probability of crossover for each chromosome. Then, the initial step of each crossover pro-

0-7695-1573-8/02/$17.00 (C) 2002 IEEE

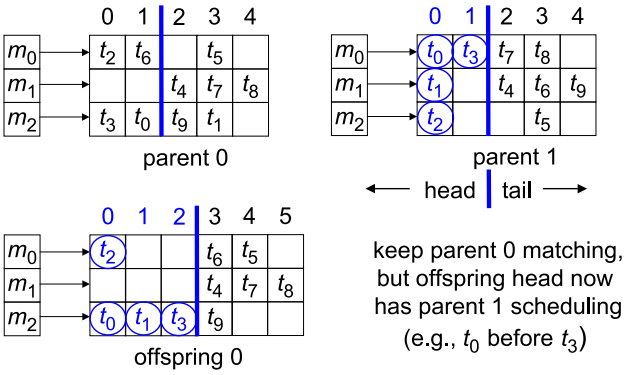

Figure 3. Example of scheduling crossover.

cedure was to process the entire population, and for each chromosome generate a random (floating point) number $x \in U[0,1]$. If $x \leq P_c$, that chromosome was part of a brood. If $x > P_c$, that chromosome was not in this particular crossover procedure. Each crossover procedure, i.e., matching, scheduling, and versions, selected chromosomes independently. Based on results from [27] and initial testing, $P_c = 0.5$ was used. Brood sizes of $\mathcal{B} = 10$ were used.

The first crossover operation was matching crossover. The procedure selected one machine queue in one chromosome of the brood at random. Then, for the m-tasks in that queue, the same m-tasks were found in the partner chromosome. The m-tasks in the partner chromosome were then moved so that they had the same matching as the first chromosome. If there were conflicts, a new (initially empty) column was inserted.

The next crossover procedure, scheduling crossover, is illustrated in Figure 3. First, two adjacent parents from the brood were selected to create an offspring together. Then, a random cut for both parents was selected, dividing both chromosomes into a head and tail. Offspring 0 was initialized with a copy of parent 0. Then, the tasks in the head of offspring 0 were deleted. Tasks were extracted from the head of parent 1, and while maintaining their scheduling, were placed into the head of offspring 0 using the matching from parent 0. Extra tasks were replaced as appropriate (e.g., t_6 in Figure 3). Thus, the matching information of parent 0 was preserved, but the scheduling information from the head of parent 1 was imposed. This procedure works because a valid subtask ordering was maintained at all times. This example represents one simple case of scheduling crossover; other cases are discussed in [7]. These crossover procedures are similar to schemes in [11, 26].

For version crossover, a gyre crossover of version vectors was used. In each parent, two crossover points were selected at random, and the middle segments were imposed on neighbors in a gyral fashion.

Mutation. After all three crossover procedures were completed, the mutation procedures were performed. Let P_m be the probability of mutation for each chromosome. Based on results from [27] and initial testing, $P_m = 0.5$ was used. Then, similar to crossover, the initial step of each mutation procedure was to process the entire population (which had been changed by the three types of crossover), and for each chromosome generate a random (floating point) number $x \in U[0,1]$. If $x \leq P_m$, that chromosome was mutated. Each mutation procedure, i.e., matching, scheduling, and versions, selected chromosomes independently.

The method used to perform a matching mutation first selected a target m-task at random. Next, a new machine to which the target m-task was matched was selected at random. Finally, the contents of the two designated mapping table positions (within the same column) were exchanged.

The method used to perform a scheduling mutation only affected the ordering of tasks on a single machine. The first step selected a target m-task. Next, the valid range for the target m-task was determined. The valid range is the set of index positions to which the target m-task can be moved and not violate any dependency constraints [27]. Next, a new position from within the valid range was selected at random, and the contents of these two positions were exchanged. The procedures defined for matching mutations and scheduling mutations are similar to the order-based mutation reported in [21, 26].

For a version mutation, a random task was selected. Then a new, different value for that task's version was randomly generated.

Stopping Criteria. Four stopping criteria were used for the generational GA. The GA was stopped as soon as one of these conditions was met: (1) 1000 generations had been computed, (2) the elite chromosome had not changed for 150 generations [27], (3) all chromosomes had converged to the same mapping, or (4) the elite chromosome represented the TUB solution.

3.3. GENITOR-Based Mapping Heuristic

GENITOR is a steady-state genetic search algorithm that has been shown to work well for several problem domains, including job shop scheduling and parameter optimization (e.g., [28, 30]). A typical GENITOR-style GA would be implemented as follows. First, an initial population is generated and evaluated, as it was with the generational GA. Next, the entire population is sorted (ranked) by each chromosome's fitness value, and stored in this sorted order. Then, a special function is used to select two chromosomes to

0-7695-1573-8/02/$17.00 (C) 2002 IEEE

act as parents. The two parents perform a crossover, and a (single) new offspring is generated. This offspring is then evaluated, and must immediately compete for inclusion in the population. If the offspring has a higher fitness than the poorest member of the population, the offspring is inserted in sorted order in the population, and the poorest chromosome is removed. Otherwise, the offspring is discarded. This would continue until a stopping condition is reached.

The special function for selecting parents is a bias function, used to provide a specific selective pressure [30]. 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 than the median chromosome. Elitism is implicitly implemented by always removing the poorest chromosome.

In this study, instead of individual pairs performing crossover, broods of size $\mathcal{B} = 10$ were used, to take advantage of gyre crossover and remain consistent with the generational GA. A linear bias of 1.5 was used to select ten chromosomes for crossover. The same three crossovers from the generational GA were used.

After each type of crossover, the offspring were considered for that same type of mutation. For example, after a brood underwent a matching crossover, each new offspring was considered for matching mutation. A probability of mutation of $P_m = 0.5$ was used. After (possibly) being mutated, the new offspring were evaluated and considered for insertion into the population. The fitness of each chromosome was evaluated using E.

The stopping conditions for the GENITOR-style GA were: (1) 100,000 total offspring, (2) 15,000 offspring with no change in the upper 50% of the population, (3) the TUB was found, or (4) all chromosomes converged to the same solution. Condition (1) was based on a stopping condition from [30] and condition (2) approximated the no change in elite condition from the generational GA. Similar to the generational GA, between four and eight runs for each ETC matrix were performed. Although GENITOR is a steady-state GA, the GENITOR-style GA implemented in this study was not a true steady-state GA. The use of broods introduced a generational component.

4. Results from Simulation Studies

Three different HC mapping test case scenarios were examined: (1) high-priority weighting and high-arrival rate, (2) high-priority weighting and moderate-arrival rate, and (3) low-priority weighting and high-arrival rate. Each result reported is the average of 50 different trials, with $T = 2000$ m-tasks, $M = 8$ machines, and $V = 3$ versions. The 95% confidence interval for each

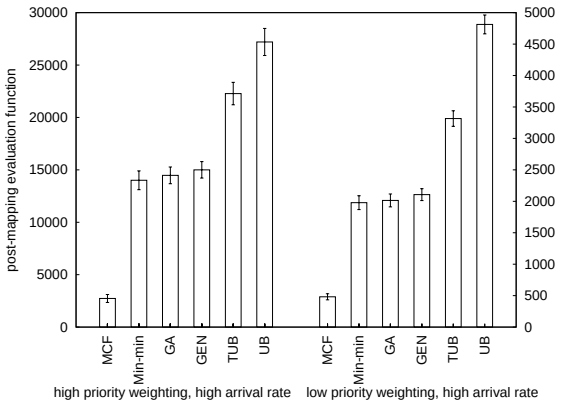

Figure 4. Comparison of heuristics for high-priority weighting and low-priority weighting.

heuristic is shown at the top of each bar [17], GENITOR is abbreviated as GEN.

The average execution time on a single trial for MCF was 2 minutes, whereas Min-min took, on average, 23 minutes. The GA and GENITOR heuristics were designed with similar stopping conditions; so both averaged 12 hours per trial. Timings are from Pentium III 700 MHz processors with 512 MB RAM and 256 KB L2-cache, running Linux RedHat version 6.2.

The left side of Figure 4 shows the results for the high-priority weighting, high-arrival rate scenario. MCF performed the poorest, achieving only 20% of the performance of Min-min. Min-min performed well and achieved 62% of TUB. GA was able to improve the initial Min-min seeded mappings by 2%, achieving 64% of TUB. GENITOR did the best of all the mapping heuristics, finding mappings that were 5% higher than Min-min and achieving 67% of TUB.

The right side of Figure 4 shows the average performance that MCF, Min-min, GA, and GENITOR achieved for the low-priority weighting, high-arrival rate scenario. In general, the results were similar to the high-priority weighting, high-arrival rate scenario.

The right side of Figure 5 shows the results for the high-priority weighting, moderate-arrival rate scenario. The lower arrival rate presented an easier mapping problem than the other scenarios. Thus, each heuristic performed better at the slower arrival rate. For example, on the right side of Figure 5, Min-min is 76% of TUB, and GENITOR is 82% of TUB. Compare this with 62% and 67%, respectively, for the high-priority weighting, high-arrival rate case.

The stopping conditions encountered by each of the GA and GENITOR did not vary much among the different scenarios. For the GENITOR experiments, the most common stopping condition was reaching 100,000

0-7695-1573-8/02/$17.00 (C) 2002 IEEE

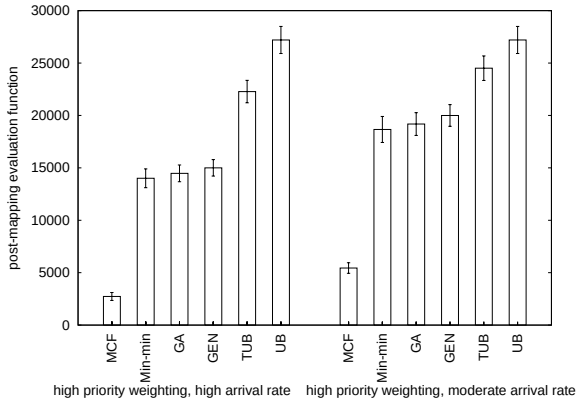

Figure 5. Comparison of heuristics for high and moderate arrival rates.

total offspring, occurring for 99% of all runs (the other 1% was no change in the upper 50% of the population). For GA, the most common stopping condition was 150 generations with no change in elite, occurring for 85% of all runs. The other 15% of these GA runs encountered the stopping condition of 1000 total generations. GA and GENITOR used an average of 6.9 and 7.2 initial populations, respectively.

In the previous experiments, a mapping that can achieve the UB or TUB was impossible to create. Thus, another set of experiments is introduced, based on a non-overloaded, contrived set of data, where the optimal solution was known, and the heuristics' best solutions could be compared to optimal solutions.

For this contrived class of mapping problems, the optimal mapping had the following properties. Each task was able to execute its most preferred version. Each task was able to meet its deadline. All subtasks in a task had the same best machine (so there were no inter-machine communications). This achievable optimal solution represents a perfect packing of tasks to machines in the HC environment. The arrival rate of tasks was much slower, to allow for the perfect packing solution. This set of experiments used the average of 25 *ETC* matrices with high priorities, 512 m-tasks, and 8 machines.

For these experiments, Min-min performed very well. On average, Min-min achieved 94.5% of the optimal solution. GA came slightly closer (94.6%), and GENITOR came the closest, 95.0% of the optimal. While not a mathematical proof, this would indicate that the heuristics may have performed similarly in the overloaded experiments because they came close to the (unknown) optimal solution.

Other experiments conducted during this research investigated the use of intron and exon chromosome regions, weighted evaluation functions, stochastic versions of MCF and Min-min, and other arrival rates and priority weights. More details are in [7].

5. Summary

This paper presented a new paradigm for an HC environment, where tasks had deadlines, priorities, multiple versions, and subtasks. Two upper bounds, two greedy heuristics, and two kinds of genetic algorithms were implemented and compared.

It was shown that the Min-min approach performed very well, achieving 59% to 76% of the performance of TUB. The generational GA approach, seeded with Min-min, improved these mappings by only 2% to 3%, achieving 61% to 78% of TUB's value. The GENITOR approach, also seeded with Min-min, found the best mappings with an improvement of 6% to 8% over Min-min, achieving 67% to 82% of TUB's performance.

The heuristics were then examined against achievable optimal solutions. Min-min, GENITOR, and GA were all within 5.5% of known optimal solutions in these cases.

This study presented one implementation and application of GAs for the HC mapping problem (many others are possible). In situations where a high quality assignment of machines to tasks is critical, their use is justified. However, the faster Min-min heuristic also provided very good results.

Acknowledgments – The authors thank Shoukat Ali, Jong-Kook Kim, Loretta Vandenberg, and Darrell Whitley for their comments.

References

[1] "Agile Information Control Environment Proposers Information Package," BAA 98-26, May 1998, http://webext2.darpa.mil/iso/aice/-AICE98-26.htm (viewed June 1998).

[2] S. Ali, T. D. Braun, H. J. Siegel, and A. A. Maciejewski, "Heterogeneous Computing," *Encyclopedia of Distributed Computing*, J. Urbana and P. Dasgupta, eds., Kluwer Academic Publishers, Norwell, MA, to appear 2002.

[3] 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 Engr.*, vol. 3, no. 3, Nov. 2000, pp. 195–207.

[4] R. Armstrong, *Investigation of Effect of Different Run-Time Distributions on SmartNet Performance,* master's thesis, Dept. of Comp. Science, Naval Postgraduate School, Monterey, CA, 1997.

[5] J. E. Baker, "Reducing Bias and Inefficiency in the Selection Algorithm," *2nd International Conference on*

0-7695-1573-8/02/$17.00 (C) 2002 IEEE

Genetic Algorithms, J.J. Grefenstette, ed., July 1987, pp. 14–21.

[6] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund, "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.

[7] T. D. Braun, *Heterogeneous Distributed Computing: Off-line Mapping Heuristics for Independent Tasks and for Tasks with Dependencies, Priorities, Deadlines, and Multiple Versions*, doctoral dissertation, School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, 2001.

[8] T. D. Braun, H. J. Siegel, N. Beck, L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao, "A Taxonomy for Describing Matching and Scheduling Heuristics for Mixed-Machine Heterogeneous Computing Systems," *17th IEEE Symposium on Reliable Distributed Systems,* Oct. 1998, pp. 330–335.

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

[10] "DARPA ITO - Quorum Vision," http://www.darpa.mil/ito/research/quorum/vision.html (viewed Aug. 2001).

[11] L. Davis, "Applying Adaptive Algorithms to Epistatic Domains," *9th International Joint Conference on Artificial Intelligence*, Aug. 1985, pp. 162–164.

[12] K. A. DeJong, *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*, doctoral dissertation, Dept. of Computer and Communication Sciences, University of Michigan, Ann Arbor, MI, 1975.

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

[14] I. Foster and C. Kesselman, *The Grid : Blueprint for a New Computing Infrastructure*, Morgan Kaufman, New York, NY, 1999.

[15] J. H. Holland, *Adaptation in Natural and Artificial Systems,* University of Michigan Press, Ann Arbor, MI, 1975.

[16] O. H. Ibarra and C. E. Kim, "Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors," *Journal of the ACM*, vol. 24, no. 2, Apr. 1977, pp. 280–289.

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

[18] M. Kafil and I. Ahmad, "Optimal Task Assignment in Heterogeneous Distributed Computing Systems," *IEEE Concurrency*, vol. 6, no. 3, July–Sept. 1998, pp. 42–51.

[19] 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.

[20] 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.

[21] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, New York, NY, 1996.

[22] H. J. Siegel, H. G. Dietz, and J. K. Antonio, "Software Support for Heterogeneous Computing," *The Computer Science and Engineering Handbook*, A. B. Tucker, Jr., ed., CRC Press, Boca Raton, FL, 1997, pp. 1886–1909.

[23] H. Singh and A. Youssef, "Mapping and Scheduling Heterogeneous Task Graphs Using Genetic Algorithms," *5th IEEE Heterogeneous Computing Workshop (HCW '96)*, Apr. 1996, pp. 86–97.

[24] M. Srinivas and L. M. Patnaik, "Genetic Algorithms: A Survey," *IEEE Computer*, vol. 27, no. 6, June 1994, pp. 17–26.

[25] G. Syswerda, "Uniform Crossover in Genetic Algorithms," *3rd International Conference on Genetic Algorithms*, J. Schaffer, ed., Morgan-Kaufmann, San Mateo, CA, June 1989, pp. 2–9.

[26] G. Syswerda, "Schedule Optimization Using Genetic Algorithms," *Handbook of Genetic Algorithms*, L. Davis, ed., Van Nostrand Reinhold, New York, 1991, pp. 332–349.

[27] 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*, vol. 47, no. 1, Nov. 1997, pp. 1–15.

[28] J.-P. Watson, S. Rana, L. D. Whitley, and A. E. Howe, "The Impact of Approximate Evaluation on the Performance of Search Algorithms for Warehouse Scheduling," *Journal of Scheduling*, vol. 2, no. 2, Mar./Apr. 1999, pp. 79–98.

[29] L. R. Welch, P. V. Werme, L. A. Fontenot, M. W. Masters, B. A. Shirazi, B. Ravindran and D. W. Mills, "Adaptive QoS and Resource Management Using *A Posteriori* Workload Characterizations," *5th IEEE Real-Time Technology and Applications Symp.*, June 1999, pp. 266–275.

[30] D. Whitley, "The GENITOR Algorithm and Selective Pressure: Why Rank-Based Allocation of Reproductive Trials Is Best," *3rd International Conference on Genetic Algorithms*, D. Schaffer, ed., Morgan Kaufmann, San Francisco, CA, June 1989, pp. 116–121.

[31] 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.

0-7695-1573-8/02/$17.00 (C) 2002 IEEE