ARTICLE IN PRESS

Available online at www.sciencedirect.com

SCIENCE DIRECT®

ELSEVIER

J. Parallel Distrib. Comput. ▮▮▮ (▮▮▮▮) ▮▮▮–▮▮▮

Journal of Parallel and Distributed Computing

www.elsevier.com/locate/jpdc

Static allocation of resources to communicating subtasks in a heterogeneous ad hoc grid environment☆

Sameer Shivle[a], H.J. Siegel[a,b], Anthony A. Maciejewski[a], Prasanna Sugavanam[a,*], Tarun Banka[a], Ralph Castain[c], Kiran Chindam[a], Steve Dussinger[d], Prakash Pichumani[a], Praveen Satyasekaran[e], William Saylor[a], David Sendek[a], J. Sousa[d], Jayashree Sridharan[a], José Velazco[f]

[a]*Electrical & Computer Engineering Department, Colorado State University, Fort Collins, CO 80523, USA*
[b]*Computer Science Department, Colorado State University, Fort Collins, CO 80523, USA*
[c]*Advanced Computing Laboratory, Los Alamos National Laboratory, Los Alamos, NM 87545, USA*
[d]*HP Technologies, Fort Collins, CO 80528-9544, USA*
[e]*Xilinx, Inc., Longmont, CO 80503, USA*
[f]*Stelex, San Juan, Puerto Rico*

Received 17 February 2005; received in revised form 6 October 2005; accepted 15 October 2005

Abstract

An ad hoc grid is a heterogeneous computing and communication system that allows a group of mobile devices to accomplish a mission, often in a hostile environment. Energy management is a major concern in ad hoc grids. The problem studied here focuses on statically assigning resources in an ad hoc grid to an application composed of communicating subtasks. The goal of the allocation is to minimize the average percentage of energy consumed by the application to execute across the machines in the ad hoc grid, while meeting an application execution time constraint. This pre-computed allocation is then used when the application is deployed in a mission. Six different heuristic approaches of varying time complexities have been designed and compared via simulations to solve this ad hoc grid allocation problem. Also, a lower bound based on the performance metric has been designed to compare the performance of the heuristics developed.

Keywords: Ad hoc grid; Communication scheduling; Mapping; Resource allocation; Task scheduling

1. Introduction

An ad hoc grid is a heterogeneous computing (HC) and communication system, all of whose components are mobile. Ad hoc grids allow a group of individuals to accomplish a mission that involves computation and communication among the grid components, often in a hostile environment. Examples of applications of ad hoc grids include: disaster management, wildfire fighting, and defense operations [MaM03]. In all of these cases, a grid-like environment is necessary to reliably support the coordinated effort of a group of individuals working under extreme conditions.

An important research problem is how to assign resources to the subtasks (matching) and order the execution of the subtasks that are matched (scheduling) to maximize some performance criterion of a HC system. This procedure of matching and scheduling is called mapping or resource allocation. The mapping problem has been shown, in general, to be NP-complete (e.g. [Cof76,Fer89,IbK77]). Thus, the development of heuristic techniques to find near-optimal solutions for the mapping problem is an active area of research (e.g. [AlK02,BaS01,BaV01,BrS01a,BrS01b,Esh96,MaA99,MiF00, WuS00]).

☆ This research was supported in part by the Colorado State University George T. Abell Endowment.

* Corresponding author.
E-mail addresses: ssameer@engr.colostate.edu (S. Shivle), hj@engr.colostate.edu (H.J. Siegel), aam@engr.colostate.edu (A.A. Maciejewski), prasanna@engr.colostate.edu (P. Sugavanam), tarunb@engr.colostate.edu (T. Banka), rhc@lanl.gov (R. Castain), kiran@engr.colostate.edu (K. Chindam), sjd@fc.hp.com (S. Dussinger), prkash@engr.colostate.edu (P. Pichumani), moses@engr.colostate.edu, moses@xilinx.com (P. Satyasekaran), sendekdm@lamar.colostate.edu (D. Sendek), jso@fc.hp.com (J. Sousa), jaya@engr.colostate.edu (J. Sridharan), jose.velazco@abbott.com (J. Velazco).

0743-7315/$ - see front matter © 2005 Published by Elsevier Inc.
doi:10.1016/j.jpdc.2005.10.005

For this research, a known large application task composed of communicating subtasks with data dependencies among them is to be mapped to machines in an ad hoc grid. We statically (offline) find a resource allocation for the application task that will be needed later in the field for a mission to be completed (e.g., response to a specific wildfire). For each mission that uses this known application, mission-specific input data will be processed by the subtasks. The goal is to map subtasks to machines in such a way as to minimize the average of the percentage of the energy that is consumed by the machines in the grid, while meeting an application execution time constraint.

The contributions of this paper are the design and comparison via simulations of six different heuristic approaches, of varying time complexities, to solve this ad hoc grid allocation problem. Also, a lower bound (LB) based on the performance metric has been derived to evaluate the performance of the heuristics developed.

The next section describes the problem statement for this paper. Section 3 discusses some of the literature related to this work. In Section 4, heuristics studied in this research and the LB developed for the problem are presented. Section 5, explains the simulation setup used for this research. Section 6 describes the results, and the last section gives a brief summary of this research.

2. Problem statement

Each application task generated for this study is composed of a set S of communicating subtasks with data dependencies among them. The data dependencies among the subtasks are represented by a directed acyclic graph (DAG). There is a set M of machines in the ad hoc grid. As is typical in static mapping studies, the estimated execution time for each subtask on each machine is assumed to be known a priori (e.g. GhY93,KaA98,KhP93,MaB99,SiD97,SiY96). The estimated time to compute (ETC) values are used by the mapping heuristics. The ad hoc grid considered for this study is composed of two classes of machines: "fast machines" (e.g., laptops) and "slow machines." (e.g., palmtops). Each machine j has four energy parameters associated with it:

(a) initial battery energy: $B(j)$;
(b) rate at which it consumes energy for executing a subtask, per execution time unit: $E(j)$;
(c) rate at which it consumes energy for sending subtask communication, per communication time unit: $C(j)$; and
(d) the machine's communication bandwidth: $BW(j)$.

Parameters (b) and (c) are a simplified model of real energy consumption. A more complex model of energy consumption may be considered in the future work. The details of the simulation environment are presented in Section 5.

Let the estimated execution time of subtask i on machine j be $ETC(i, j)$. Then the energy consumed for executing a single subtask i on machine j is $ETC(i, j) \times E(j)$.

The time required to transfer one bit of a data item between machine j and machine k is the inter-machine communication time called $CMT(j, k)$ and is given by

$$CMT(j,k) = 1/\min\left(BW(j), BW(k)\right). \quad (55)$$

The energy consumed to send a data item g of size $|g|$ from machine j to machine k is given by

$$CMT(j,k) \times C(j) \times |g|.$$

For the environment considered in this study it is assumed that devices are close enough to each other so that single-hop communication is possible. In addition, for the simulations described in Section 5, it is assumed that there are $|M| = 8$ distinct communication channels. Thus, each machine can transmit data to any other machine, but only one destination at a time, and can do so while computing. A machine can simultaneously handle one outgoing data transmission and one incoming data reception. Similar to the study in [WaS97], we assume that:

(a) a subtask can send out data only after it has completed execution; and
(b) a subtask may not begin execution until it receives all of its input data items.

The ad hoc grid that is considered for this project is a simplified version of an actual one. The list of simplifying assumptions that have been made are as follows:

(a) the energy consumed by a subtask to receive a data item is ignored;
(b) any initial data (i.e., data not generated during execution of the application task) is preloaded before the actual execution of the application task begins;
(c) a machine consumes no energy if it is idle (i.e., not computing or not transmitting).

The performance metric for this study is based on the energy consumption across all the machines in the ad hoc grid. The total battery energy consumed by a machine j after the entire application task has been completed is given by $EC(j)$. The performance metric, B_{pavg} used to evaluate the mapping is defined as the percentage of energy consumed by each machine to complete the entire application task, averaged across all machines, and is given by

$$B_{\text{pavg}} = \frac{\sum_{j=0}^{|M|-1} (EC(j)/B(j))}{|M|}.$$

The goal is to map all the subtasks to machines in such a way as to minimize B_{pavg} while meeting certain constraints. The motivation for minimizing B_{pavg} is to allow each machine to retain energy for performing operations in addition to the application task. The use of B_{pavg} as a metric is one way to capture this attribute.

The first constraint is that all the subtasks in the application task have to be executed. The second constraint is the energy constraint. Each machine in the grid has some initial battery energy. Every time a subtask is executed or data are transmitted by a machine some of the battery energy on that machine is consumed. Hence, the available battery energy on each machine becomes a constraint while mapping the application task to

the grid. In addition to the energy constraint, for this study an additional execution time constraint τ has been imposed, during which the entire application task must finish executing. The makespan is defined as the overall execution time of the application task on the machine suite in the ad hoc grid. The final makespan of a mapping must be less than or equal to a time constraint τ. Finally, the wall clock time for each mapper itself to execute is required to be less than or equal to 60 min on a typical unloaded 1 GHz desktop machine. This constraint was to prevent some heuristics from taking an "unreasonable" amount of time; the value of 60 min was arbitrary.

Six static mapping schemes are studied in this paper: Levelized Weight Tuning (LWT), Bottoms Up, Min–Min, A*, Simplified Lagrangian (SL), and Genetic Algorithm (GA). For this study, 10 different ETC matrices and 10 different DAGs were generated to create 100 different scenarios, where each scenario is a combination of one of the application task graphs and one of the ETC matrices. The performance of each heuristic is studied across these 100 different scenarios.

3. Related work

A significant amount of research has been performed in the areas of power constrained resource management in uniprocessors (e.g., [HoK99,YaD95]) and also in heterogeneous multiprocessors (e.g., [MiR03,YuP02]). In all these studies, however, power management is achieved through voltage scaling, which allows the reduction of the power usage by a CPU (which requires a reduction in clock frequency) at the expense of increasing the execution time of a task. Thus, these papers focused on using voltage scaling, while our work assumes processors that operate at only one voltage level and focuses on minimizing B_{pavg} by using an appropriate allocation of resources.

The literature was examined to select a set of heuristics appropriate for the HC environment considered here. New heuristics based on these approaches were designed for minimizing B_{pavg} for ad hoc grids. Three of the six heuristics presented in this paper, namely Min–Min, GA, and A*, have been used previously to map tasks onto heterogeneous machines (e.g., [BrS01b]). However, unlike [BrS01b], where the goal was to minimize the total time required to complete an application task, the goal of our study is to minimize the average percentage of energy consumed by the machines. The Min–Min heuristic approach has proven to be a good heuristic for dynamic and static mapping problems in earlier studies (e.g., [BrS01b,MaA99]). The Bottoms Up heuristic used in this study is a variation of the Min–Min heuristic. Bottoms Up assigns tasks to machines in a manner similar to the Min–Min heuristic, but considers tasks in a different order.

GAs are a technique used for searching large solution spaces and have been used for mapping tasks to machines in an HC environment (e.g., [BrS01b,SrP94,WaS97]). The GA used in this study is based on [WaS97] and has been modified for this problem environment. A* is a search technique that is highly effective in searching a tree or graph and has been used for many task allocation problems (e.g., [BrS01b,ChL91,KaA98]). The SL heuristic presented in this paper is a modified version of the one used in [LuZ00]. Lagrangian relaxation techniques have been used in [LuZ00] for job scheduling in an industrial environment.

4. Heuristics

For all the heuristics except Bottoms Up, only the subtasks whose predecessors had been fully mapped could be considered during a given mapping iteration (referred to as mappable subtasks). Also, for the final mapping of all six heuristics, the energy constraint is that $B(j)$ is not exceeded for any machine, and the time constraint is that the execution time of the application does not exceed τ. This section describes the six heuristics and a LB on the objective function, B_{pavg}.

4.1. Levelized Weight Tuning

In a manner similar to that used in [IvO95] and as shown in Fig. 1, the LWT heuristic assigns subtasks to different levels depending on the data precedence constraints. The lowest level consists of subtasks with no predecessors and the highest level consists of subtasks with no successors. Each remaining subtasks is at one level below the lowest producer of its global data items. Starting from the lowest level, each subtask on its respective level is assigned a priority based on the total size (sum) of its output global data items, the larger the sum the higher the priority.

The LWT heuristic can be summarized by the following procedure:

1. All the subtasks are first assigned levels depending on the precedence constraints. Subtasks on each level are assigned a priority as described above.
2. Starting from the lowest to the highest level (see Fig. 1), subtasks are considered for mapping by levels. Within each level, subtasks are considered by priority, from high priority to low priority.
3. For every level L, a ratio $\lambda(L)$ is calculated as follows:

   $\lambda(L)$=(current level number+1)/(total number of levels).

4. Every time a subtask S_j within a level is considered for mapping on a machine:
   Find a machine M_1 that will increase the current B_{pavg} of the system by the least amount. Also, find a machine M_2 that will increase the current makespan of the system by the least amount. A ratio η, which is the ratio of current makespan to τ is calculated.
5. If $\eta > (\lambda(L) \times F)$, where F is a weighting factor that is experimentally determined,
   then the subtask is mapped to machine M_2,
   else
   the subtask is mapped to machine M_1.
6. Update the time and energy availability of the machine on which the subtask is mapped. Also, update the energy availability across all machines that send global data items to the mapped subtask.

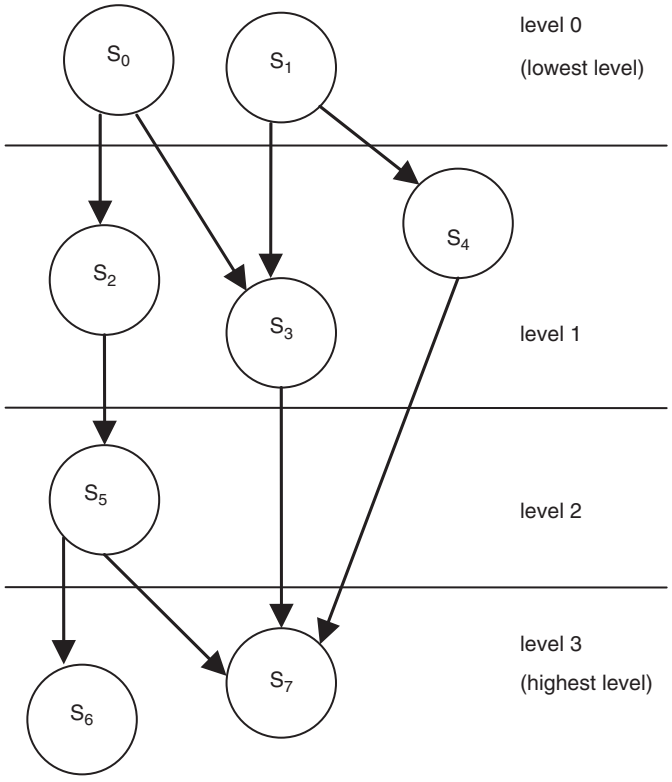

Fig. 1. Levelizing of subtasks S_0, S_1, S_2, S_3, S_4, S_5, S_6, and S_7 for a given sample DAG.

7. Repeat steps 2–6 until all the subtasks are mapped and calculate the final value of B_{pavg}.

The value of the weighting factor F was varied from 1 to 2 in steps of 0.1 for each complete mapping for each scenario. From among all these different mappings for each scenario, the mapping that gave the smallest value of B_{pavg} and also met the energy and time constraints is chosen as the final mapping. The average value of F for this study was found to be 1.6.

4.2. Bottoms Up

The Bottoms Up (BU) heuristic assigns subtasks to levels in a manner similar to the LWT heuristic. However, unlike LWT, the BU heuristic begins by mapping subtasks from the highest level. Thus, for the BU heuristic, the set of mappable subtasks at any given time consists of all subtasks that either have no successors or subtasks whose successors have previously been mapped.

Let the time for execution and communication of subtask i on machine j, normalized with respect to the maximum time required for execution and communication by subtask i across all machines, be $NT(i, j)$. Let the energy consumed for execution and output communication of subtask i on machine j, normalized with respect to the maximum energy consumed for execution and output communication of subtask i across all machines, be $NE(i, j)$. Then, using β as a weighting parameter, the fitness value γ_{ij} is calculated as follows:

$$\gamma_{ij} = (\beta \times NT(i, j)) + ((1 - \beta) \times NE(i, j)). \quad (27)$$

The BU heuristic can be summarized by the following procedure.

1. All the subtasks are first assigned levels depending on the precedence constraints as explained above.
2. Starting from the highest level to the lowest level, for each level a list of mappable subtasks is generated.
3. For each mappable subtask i at the current level, find the machine j across all machines that gives the subtask its minimum fitness value γ_{ij}, ignoring other subtasks on that level.
4. From among all the subtask/machine pairs found in the above step, find the pair that gives the minimum fitness value.
5. The subtask found in the above step is then assigned to its paired machine.
6. Repeat steps 2–5 for each level (from highest to lowest level) until all subtasks are assigned machines.
7. After all subtasks are assigned machines, they are scheduled in the reverse order they were matched.
8. The entire mapping is then evaluated and the final value of B_{pavg} is calculated.

The value of the weighting factor β was varied from 0 to 1 in steps of 0.1 for each complete mapping for each scenario. From among all these different mappings for each scenario, the mapping that gave the smallest value of B_{pavg} and also met the energy and time constraints is chosen as the final mapping. The average value of β for this study was 0.5.

4.3. Min–Min

Based on the Min–Min concept in [IbK77], this heuristic utilizes a fitness function to evaluate all mappable subtasks. The fitness function is chosen such that it reflects the change in B_{pavg} and also the change in the makespan of the system if a subtask is mapped onto a machine. Let $PB_{\text{pavg}}(i, j)$ be the partial B_{pavg} of the system if subtask i was mapped to machine j. Let $PCT(i, j)$ be the partial completion time of machine j, normalized with respect to τ, if subtask i was mapped to machine j. Then using α as a weighting parameter, the fitness value $f(i, j)$ of any subtask i on machine j is calculated as follows:

$$f(i, j) = \alpha \times PB_{\text{pavg}}(i, j) + ((1 - \alpha) \times PCT(i, j)). \quad (67)$$

The Min–Min heuristic can be summarized by the following procedure:

1. A list of mappable subtasks is created. Initially, this list consists of subtasks with no predecessors.
2. For each subtask i in the above list, across all machines, find the machine j that gives the subtask its minimum fitness value $f(i, j)$, ignoring other subtasks in the list. This is the first "Min."

3. From among all the subtask/machine pairs found in step 2, find the pair that gives the minimum fitness value. This is the second "Min."
4. The subtask found in the above step is then removed from the list of mappable subtasks and is mapped to its paired machine.
5. Update the time and energy availability of the machine on which the subtask is mapped. Also, update the energy availability across all machines that send global data items to the mapped subtask.
6. The set of mappable subtasks is updated to include any other new subtasks all of whose predecessors have been mapped.
7. Repeat steps 2–6 until all the subtasks are mapped and calculate the value of B_{pavg}.

The value of α was varied from 0 to 1 in steps of 0.1 for each complete mapping for each scenario. From among all these different mappings for each scenario, the mapping that gave the smallest value of B_{pavg} and also met the energy and time constraints is chosen as the final mapping. The average value of α for this study was 0.9.

4.4. A*

The A* technique used in this study is in concept based on that used in [BrS01b,ChL91]. A* is a tree-search algorithm, beginning at a root node that is a null solution. As the tree grows, nodes represent partial mappings (a subset of subtasks is assigned to machines). The partial mapping (solution) of a child node has one subtask mapped more than the parent node. For each node n, a cost function $c(n)$ is calculated.

Let $g(n)$ be the maximum of the machine completion times for the subtasks mapped through node n (this calculation includes time for communications). Let $mmct(n)$ be the maximum of the minimum machine completion times over all unassigned subtasks U at node n and is defined as follows:

$$mmct(n) = \max_{i \in U} \left(\min_{0 \leqslant j \leqslant |M|-1} (\text{time machine } j \text{ available} + ETC(i, j)) \right).$$

The LB estimate of the completion time $h(n)$, of all the unassigned subtasks U at node n (this calculation does not include time for communications) is defined as follows:

$$h(n) = \max(0, (mmct(n) - g(n))).$$

The function $f(n)$ that is an estimate of the time required to complete all the subtasks, normalized with respect to τ, is then given by

$$f(n) = (g(n) + h(n))/\tau.$$

The function $p(n)$ is the LB of the estimated B_{pavg} for all the subtasks through node n. Let $g'(n)$ be the B_{pavg} for all the subtasks mapped through node n (this calculation includes the energy for communications). Let $h'(n)$ be the LB estimate of the B_{pavg} for the set of unassigned subtasks U at node n (this calculation does not include energy consumed for communications) and it is defined as follows:

$$h'(n) = \sum_{i=0}^{|U|-1} \left(\min_{0 \leqslant j \leqslant |M|-1} (\text{new } B_{\text{pavg}} \text{ (if } i \text{ is mapped to } j) - g'(n)) \right).$$

Thus, $h'(n)$ is calculated assuming that every unassigned subtask is assigned to a machine that increases the B_{pavg} of the system by the least amount. The function $p(n)$ is then given as follows:

$$p(n) = g'(n) + h'(n).$$

The cost function for node n is then given by

$$c(n) = \sqrt{(\mu \times f(n)^2) + p(n)^2}.$$

The weighting factor of μ was empirically determined by evaluating values between 0 and 1 in steps of 0.1, and then refining in steps of 0.01 for a sample scenario. Using the results obtained from this sample scenario, it was decided to use the weighting factor of $\mu = 0.07$ as it gave complete valid mappings (within the time and energy constraints) for all scenarios.

The A* heuristic can be summarized by the following procedure.

1. A valid total ordering of subtasks that satisfies the precedence constraints for the entire application task is first generated. All subtasks are considered for mapping in this order.
2. The root node generates eight nodes (partial mappings) by allocating the first mappable subtask to each of the eight machines.
3. After a parent node generates child nodes, it becomes inactive (i.e., it is not eligible for further expansion). The new nodes created are considered to be active nodes and are stored in a node list. The size of the node list is always kept at 100 by retaining only the best 100 active nodes (based on $c(n)$) at any one time. Similar to [BrS01b], this is done to keep the execution time of the heuristic tractable.
4. For the next mappable subtask, the node with the minimum $c(n)$ in the node list is then expanded to generate eight more new child nodes (corresponding to mapping that subtask to each of the eight machines).
5. Repeat steps 2–4 for every mappable subtask until finally a node is expanded to give eight complete mappings. From these eight complete mappings, the mapping that gives the best value of B_{pavg} and also meets the energy and time constraints is then selected as the final mapping.

Experiments with node lists of sizes larger than 100 were also conducted. However, it was found that there was no significant improvement in the value of B_{pavg}, but the heuristic execution time increased considerably.

4.5. Simplified Lagrangian

Lagrangian-based approaches have been applied to solve a wide range of complex production scheduling problems [LuZ00]. The technique used here is a modified version of [LuZ00] that is suitable for the problem environment in this study. At any time k, if the energy remaining in machine j is denoted $ER(j,k)$ and the makespan is denoted $makespan(k)$, then the Lagrangian equation, $L(\delta,k)$ is given by

$$L(\delta,k) = \delta\left(\sum_{j=0}^{|M|-1} ER(j)/B(J) \Big/ |M|\right) + (1-\delta)\,(1-(makespan(k)/\tau))\,.$$

The value of δ was empirically determined by evaluating values between 0 and 1 in steps of 0.1 for a sample scenario. Using the results obtained from this sample scenario, it was decided to use the weighting factor of $\delta = 0.8$ as it gave complete valid mappings (within the time and energy constraints) for all scenarios.

The SL heuristic can be summarized by the following procedure.

1. Every time a subtask is considered for mapping, the next available machine (i.e., the machine with the minimum machine availability time) is selected. If more than one machine has the same minimum machine availability time, then one of these machines is selected randomly.
2. For the selected machine, the list of mappable subtasks is generated. The list of mappable subtasks consists of all the subtasks whose predecessors have been mapped and can begin execution on the selected machine without violating time or energy constraints.
3. Find the potential contribution of each mappable subtask in the above list to the system Lagrangian (i.e., $L(\delta,k)$), ignoring other subtasks in the list.
4. From among the mappable subtasks found in the above step find the subtask that gives the largest value of the system Lagrangian, $L(\delta,k)$.
5. The subtask found in the above step is then removed from the list of mappable subtasks and is mapped to its selected machine.
6. Update the time and energy availability of the machine on which the subtask is mapped. Also, update the energy availability across all machines that send global data items to the mapped subtask.
7. Repeat steps 1–6 until all the subtasks are mapped and calculate the value of B_{pavg}.

The SL heuristic allowed a mappable subtask to be scheduled to execute at a time prior to the target machine's availability time (time when all subtasks already assigned to the machine will be completed) if a sufficiently large "hole" in the existing schedule could be found that complied with precedence constraints. As a result, the SL-generated mappings exhibited a very small makespan as compared to all the other heuristics.

4.6. Genetic Algorithm

This method is adapted for this problem domain from the GA approach used in [WaS97]. The GA operates on a population of 100 chromosomes. Each chromosome represents one solution to the problem and a set of chromosomes is called a population. Each chromosome is composed of a scheduling string and a matching string. The scheduling string is a total ordering of the subtasks in the DAG that obeys the precedence constraints, while the matching string gives the subtask-to-machine assignments. To form a scheduling string, the DAG is topologically sorted to form a basis scheduling string. Then, for each chromosome in the initial population, this basis string is mutated (similar to the mutation procedure described below) a random number of times to generate 96 other valid scheduling strings. The corresponding 97 matching strings are generated by randomly assigning subtasks to machines. The population also includes three chromosomes (seeds) that are the Min–Min, LWT, and Bottoms Up solutions. Similar to the approach in [WaS97], these chromosomes then undergo selection, crossover, mutation, and evaluation.

Each chromosome has a fitness value (B_{pavg}) associated with it. The rank-based roulette wheel scheme is used for selection [SrP94]. This scheme probabilistically duplicates some chromosomes and deletes others, where better mappings have a higher probability of being duplicated in the next generation. *Elitism*, the property of guaranteeing the best solution remains in the population, is also implemented [Rud94]. The population size stays fixed at 100.

In the crossover step, a pair of parent chromosomes is selected from the chromosome population. For scheduling string crossover, a random cut-off point that cuts the scheduling strings into top and bottom parts is generated for each pair selected. Then, the subtasks in each bottom part are reordered. The new ordering of the subtasks in one bottom part is the relative position of these subtasks in the other original scheduling string in the pair, thus guaranteeing that the newly generated scheduling strings are valid scheduling strings. For matching string crossover, again a random cut-off point that cuts the matching strings into top and bottom parts is generated. Then the machine assignments of the subtasks in the bottom parts are exchanged. After the crossover operation for both the scheduling and the matching strings, the newly generated chromosomes are evaluated and if the new chromosomes generated do not violate energy or time constraints, then they replace the parent chromosomes in the population; otherwise the new chromosomes are dropped and no child chromosomes are created.

In the mutation step, a parent chromosome is selected for mutation from the chromosome population. In case of scheduling string mutation, for each chosen parent scheduling string, a subtask (called the victim subtask) is selected randomly. This victim subtask is then moved randomly to another position in the scheduling string in such a way that it does not violate any precedence constraints to obtain a new valid scheduling string. In case of matching string mutation, for each chosen parent matching string, two subtask/machine pairs are selected ran-

domly and their machine assignments are swapped. Similar to crossover, after the mutation operation for both the scheduling and matching strings, the new chromosome generated is evaluated and if the new chromosome generated does not violate energy or time constraints, then it replaces the parent chromosome in the population; otherwise the new chromosome is dropped and no child chromosome is created.

For both crossover and mutation operations, the chromosome population is traversed serially in the order generated by the rank-based roulette wheel scheme. Every chromosome is considered for crossover with a probability of 40% and for mutation with a probability of 20%. These probabilities for crossover and mutation were selected by experimentation. Selection, crossover, mutation, and evaluation steps constitute a single GA iteration. The GA stops after a total of 400 iterations. Until the stopping criterion is met, the loop repeats, beginning with the selection step. At the end of 400 iterations, the chromosome that gave the best B_{pavg} is selected as the final mapping. For this study, at any point in time only chromosomes that do not violate the energy or time constraint are allowed to be in the population and the population size is always kept constant at 100 chromosomes.

4.7. Lower Bound (LB)

The method developed for estimating a LB on B_{pavg} for this study ignores data precedence constraints, inter-machine communications, the battery power constraint, and τ. For each subtask (in any order) in the application task, the minimum percentage energy it will consume over all the machines is found. These minimum percentage energy values for all the subtasks are summed up and then finally averaged over all machines. This gives us a LB on B_{pavg}. Thus, the LB can be given as

$$\frac{1}{|M|} \times \sum_{i=0}^{|S|-1} \left(\min_j \left(\frac{ETC(i,j) \times E(j)}{B(j)} \right) \right).$$

5. Simulation setup

In this study, the application task is composed of 1024 communicating subtasks. This large number of subtasks is chosen to present a significant mapping challenge for each heuristic. The pseudocode to generate the DAG is given in the appendix of this paper. For this study, 10 different DAGs are developed. The maximum fan-in (i.e., the number of input global data items received by a subtask) and fan-out (i.e., the number of output global data items sent out from a subtask) for all the 10 DAGs generated are 12 and two, respectively. Also, for each DAG there are seven subtasks with no predecessors, seven subtasks with no successors, and the remaining 1010 subtasks have predecessors and successors. The sizes of the global data items to be transferred from one subtask to another are sampling determined by a Gamma distribution, with a mean value of 2.8 megabits and a variance of 1.4 megabits.

Table 1
The values of $B(j)$, $C(j)$, $E(j)$, and $BW(j)$ for fast and slow machines

	Fast machines	Slow machines
$B(j)$	580 energy units	58 energy units
$C(j)$	0.2 energy units/s	0.002 energy units/s
$E(j)$	0.1 energy units/s	0.001 energy units/s
$BW(j)$	8 megabits/s	4 megabits/s

The ETC values for all the subtasks calculated for the simulations, taking heterogeneity into consideration, were generated using the Gamma distribution method described in [AlS00]. For this research, a task mean and coefficient of variation (COV) were used to generate the ETC matrices. The mean subtask execution time was chosen to be 100 s and a COV of 0.9 was used to generate an ETC matrix with high task and high machine heterogeneity. For this study, 10 different ETC matrices were generated and used with each of the 10 DAGs to create 100 different scenarios.

The ad hoc grid considered for this study has a total of eight machines, which were divided equally into two classes, such that machines 0–3 are the four "fast machines" and machines 4–7 are the four "slow machines." To obtain the two classes of machines, all the ETC values for the slow machines are adjusted by a multiplicative factor (MF). For each subtask i the ratio $diff_i$ of the ETC value of the fastest slow machine (machines 4–7) to the ETC value of the slowest fast machine (machines 0–3) is calculated as

$$diff_i = \left(\frac{\min ETC(i,j) \text{ for } j \text{ across slow machines}}{\max ETC(i,j) \text{ for } j \text{ across fast machines}} \right).$$

Then the value of MF is given by

$$MF = 2/(\min diff_i \text{ for } i = 0, \ldots, 1023).$$

All the ETC values for the slow machines were multiplied by MF to obtain their new adjusted values. After creating the two classes of machines, the new mean estimated execution time for a single subtask was 131 s. For this study, across all the subtasks in an ETC matrix, the average ETC value across slow machines is approximately seven times the average ETC value across fast machines.

The values of $B(j)$, $C(j)$, $E(j)$, and $BW(j)$ for both fast and slow machines are shown in Table 1. These values represent an approximate industry average based on microprocessors and battery capacity selected on currently commercially available machines. Fast machines are typified by the DELL Precision M60 notebook computer using an Intel MP4M processor operating at 1.7 GHz. The statistics for the slow machines are typical personal digital assistant (PDA) computers, such as the DELL Axim X5 that uses an Intel PXA255 processor operating at 400 MHz.

The HC system simulated for this study was assumed to be less than a day long operation. It was assumed that any of the subtasks of the application task could receive external inputs and can generate results in addition to the global data items that one subtask sent to another.

The value of the time constraint τ was chosen so that it prevented any heuristic from mapping subtasks only to slow machines, which consume less energy to execute a subtask. Experimentation with a simple greedy mapping heuristic was used to determine the value of τ as 34,075 s.

6. Results

The simulation results are shown in Figs. 2–4. The average parameter values of all the heuristics are summarized in Table 2. All heuristics were run for 10 different application task graphs (DAGs), using 10 different ETCs (i.e., for a total of 100 different scenarios). The average values over the 100 scenarios for B_{pavg} (Fig. 2) and makespan (Fig. 3) are shown along with 95% confidence intervals [Jai91]. The execution times of the heuristics averaged over 100 scenarios, mapping 1024 subtasks per scenario, are shown in Table 3.

As seen from Fig. 2, the three faster heuristics (Min–Min, LWT, and Bottoms Up), performed comparably in terms of overlapping confidence intervals. The LWT had the best average performance, though only marginally better than Bottoms Up. LWT considers subtasks for mapping by levels. Within each level, LWT tends to map subtasks to either their best B_{pavg} machine or best completion time machine depending upon a threshold factor that is level dependent. As compared to other heuristics (except the GA), for most of the scenarios, the LWT heuristic managed to map more subtasks to their best B_{pavg}

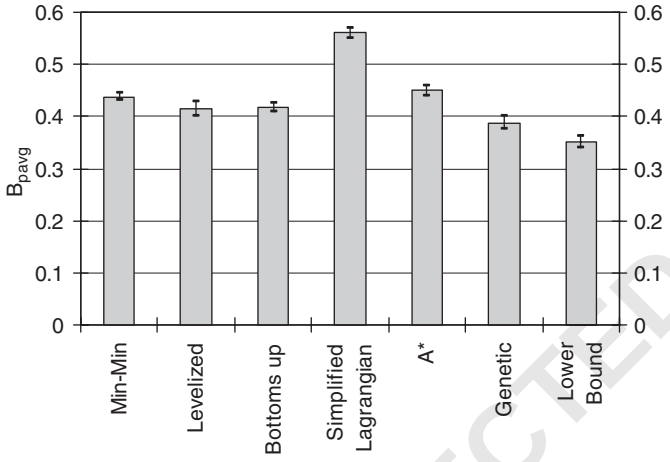

Fig. 2. The simulation results for B_{pavg}.

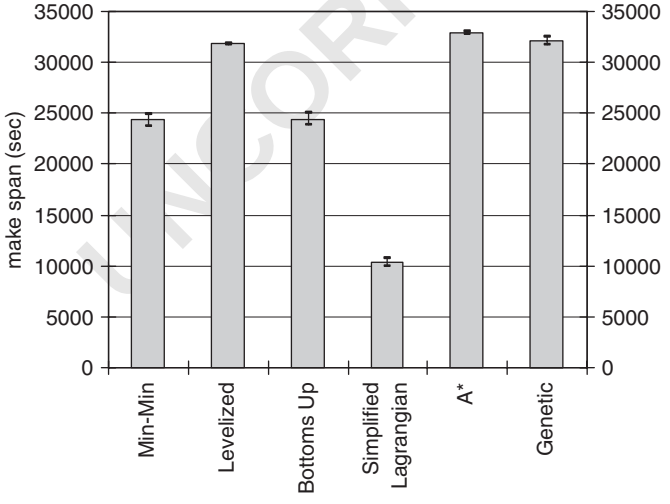

Fig. 3. The simulation results for makespan.

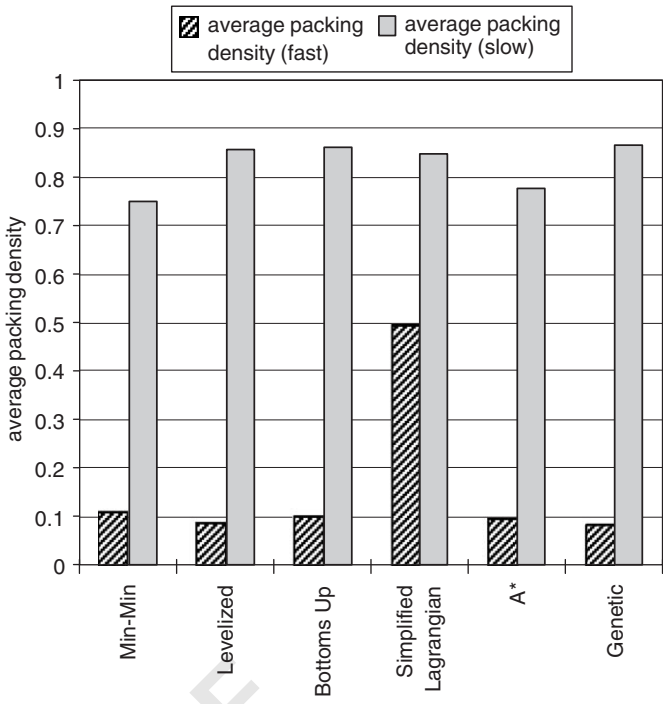

Fig. 4. The simulation results for average packing density across fast machines and slow machines.

Table 2
The parameter values of the mapping heuristics averaged over 100 scenarios

Heuristic	Average parameter values
Min–Min	$\alpha = 0.9$
Levelized Weight Tuning	$F = 1.6$
Bottoms Up	$\beta = 0.5$
A*	$\mu = 0.07$ (constant)
Simplified Lagrangian	$\delta = 0.8$ (constant)
Genetic Algorithm	Crossover probability = 0.4 Mutation probability = 0.2

Table 3
The execution times of the mapping heuristics (for mapping 1024 subtasks) averaged over 100 scenarios (using a typical 1 GHz unloaded machine)

Heuristic	Average execution times (seconds)
Min–Min	19
Levelized Weight Tuning	12
Bottoms Up	9
A*	645
Simplified Lagrangian	1200
Genetic Algorithm	3420

machines, which are mostly slow machines in this study. Hence, this heuristic tends to give a small value of B_{pavg} and a relatively high value of makespan, as seen from Fig. 3.

The next two best fast heuristics, Bottoms Up and Min–Min, are basically two-phase greedy heuristics that optimize a fitness function. The major difference between the two other than the fitness function is that Min–Min used the top to bottom approach beginning from the root node to the leaf node of the subtask graph, whereas Bottoms Up used the bottom to top approach. To see the impact of the fitness function, experiments were conducted using exactly the same procedure as Min–Min but using the Bottoms Up fitness function. It was found that the new results obtained were comparable to the old results obtained using the old Min–Min fitness function. Thus, the slight variation in the average values of B_{pavg} for Min–Min and Bottoms Up, was mainly because of the way the application task graph was traversed rather than the fitness function used. The Min–Min fitness function involved the calculation of partial makespan and hence it was not possible to implement the Bottoms Up procedure using the Min–Min fitness function.

Overall among all the heuristics, the GA performed the best and in fact performed only 14% greater than the unattainable LB. It was expected that the GA would give the best performance among all the heuristics because the GA was seeded using solutions obtained from the Min–Min, LWT, and Bottoms Up heuristics and also because it used the concept of elitism that ensured that the B_{pavg} of the new solution obtained was either better or at least the same as the seed.

The SL had the highest average B_{pavg} because it tried to optimize the makespan along with the main objective function of B_{pavg}. It tried to fill in the gaps in the machine subtask queues, when the machine was not computing and waiting for global data items, by allowing a mappable subtask to be scheduled for a time prior to the target machine's availability time (time when all subtasks already assigned to the machine will be completed) if it was possible to do so without violating precedence constraints. As described below, this resulted in a higher average usage of fast machines, which in turn leads to a higher B_{pavg}. As seen in Fig. 3, the makespan generated by the SL is significantly less than that of the other heuristics.

Another parameter, called packing density, was used to study the behavior of the heuristics for the given problem. Packing density is defined as the ratio of the total time spent by a machine for subtask execution only (ignoring the time required for communication) to the total makespan. As seen from Fig. 4, the SL had a higher average packing density over the fast machines. Thus, for all the heuristics except the SL, the fast machines had many time gaps when the machines were not doing any computation but were waiting for global data items.

7. Summary

Six static heuristics were designed, developed, and simulated using the HC environment presented. Application tasks composed of communicating subtasks with data dependencies were mapped using the heuristics described in this research.

The best B_{pavg} value was obtained by using the GA, approaching the theoretical LB derived for this study by a margin of 14%. The LWT and Bottoms Up heuristics performed comparably and were the second best. The GA used LWT and Bottoms Up as seeds and on average did approximately 4% better than these two. However, the time required for the GA itself to execute (i.e., heuristic execution time) is extremely high as compared to either the LWT or Bottoms Up heuristic.

Clearly, the results shown are for the specific parameters used for the simulation study. In practice, different missions may result in parameter values that differ from the ones used here. This includes the number of subtasks in each application, the total number of machines, and the mixture of fast and slow machines. While the relative performance of the heuristics presented here may change, the model can still be applied and the concepts underlying the heuristics are still valid.

In conclusion, the results of this research may be used in the development of ad hoc grids in support of many applications of importance, such as disaster management. In particular, many of the heuristics developed for the environment considered performed well with respect to a loose LB. A specific heuristic may be selected based on characteristics of the application domain such as heuristic execution time.

Acknowledgments

A preliminary version of portions of this material was presented at the 13th IEEE Heterogeneous Computing Workshop (HCW 2004). The authors thank Shoukat Ali, Jong-Kook Kim, and Jay Smith for their valuable comments.

Appendix

Pseudocode for generating the DAGs

```
/* input:
    Na subtask nodes with no predecessors and only successors, with id #s ranging from 1
      to Na
    Nb subtask nodes with both predecessors and successors, with id #s ranging from Na + 1
      to Na + Nb
    Nc subtask nodes with no successors and only predecessors, with id #s ranging from
      Na + Nb + 1 to Na + Nb + Nc
    maxFanOut, the maximum number of edges out of a node
    minFanOut, the minimum number of edges out of a node
*/
```

```
/* output:
   a DAG where all edges point from a smaller id node to a larger id node
*/
DAG generator pseudocode
1.  for every node with successors, i,
        /* the maximum number of outgoing edges of node i must be equal to the
           maximum fanout or the number of nodes with id larger than node i */
2.      maxedges = min(maxFanOut, number of nodes with id larger than i)
3.      generate a random number, j, between (minFanOut, maxedges)
4.      randomly select j nodes with id larger than i and generate an edge from i to each of
        the j nodes, updating the data structures accordingly
5.  endfor

/* check for nodes from (Na + 1) to (Na + Nb + Nc) that do not have an incoming edge*/
6.  for each node, i,
7.    if there is no incoming edge
        /* find the first node with id less than i that can be used to make an edge to the
        node i */
8.      for k = 1 to (i − 1) do
9.          if k does not have max outgoing edges
10.             generate an edge between the node k and the node i, and break out of this
                for loop
11.         else if k has an outgoing edge pointing to a node that has more than 1 incoming
            edge
12.         move the outgoing edge to point to node i, and break out of this for loop
13.       endif /* matches the if in Line (9) */
14.     endfor /* matches the for in Line (8) */
15.   endif /* matches the if in Line (7) */
16. endfor /* matches the for in Line (6) */
End of DAG generator pseudo code.
```


References

[AlK02] S. Ali, J.-K. Kim, Y. Yu, S.B. Gundala, S. Gertphol, H. J. Siegel, A.A. Maciejewski, V. Prasanna, Utilization-based techniques for statically mapping heterogeneous applications onto the HiPer-D heterogeneous computing system, Parallel Distributed Comput. Practices 5 (4) (2002) (Special issue on Algorithms, Systems and Tools for High Performance Computing).

[AlS00] S. Ali, H.J. Siegel, M. Maheswaran, D. Hensgen, S. Ali, Representing task and machine heterogeneities for heterogeneous computing systems, Tamkang J. Sci. Engg. 3 (3) (2000) 195–207 (invited) (Special 50th Anniversary Issue), http://www.engr.colostate.edu/~hj/journals/70.pdf

[BaS01] H. Barada, S.M. Sait, N. Baig, Task matching and scheduling in heterogeneous systems using simulated evolution, 10th IEEE Heterogeneous Computing Workshop (HCW 2001), 15th International Parallel and Distributed Processing Symposium (IPDPS 2001), Paper HCW 15, April 2001.

[BaV01] I. Banicescu, V. Velusamy, Performance of scheduling scientific applications with adaptive weighted factoring, 10th IEEE Heterogeneous Computing Workshop (HCW 2001), 15th International Parallel and Distributed Processing Symposium (IPDPS 2001), Paper HCW 06, April 2001.

[BrS01b] 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 Distributed Comput. 61 (6) (2001) 810–837.

[BrS01a] T.D. Braun, H.J. Siegel, A.A. Maciejewski, Heterogeneous computing: goals, methods, and open problems, 2001 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2001), June 2001, pp. 1–12 (invited keynote paper).

[ChL91] K. Chow, B. Liu, On mapping signal processing algorithms to a heterogeneous multiprocessor system, International Conference on Acoustics, Speech, and Signal Processing (ICASSP '91), vol. 3, 1991, pp. 1585–1588.

[Cof76] E.G. Coffman Jr. (Ed.), Computer and Job-Shop Scheduling Theory, Wiley, New York, NY, 1976.

[Esh96] M.M. Eshaghian (Ed.), Heterogeneous Computing, Norwood, MA, Artech House, 1996.

[Fer89] D. Fernandez-Baca, Allocating modules to processors in a distributed system, IEEE Trans. Software Eng. SE-15 (11) (1989) 427–1436.

[GhY93] A. Ghafoor, J. Yang, Distributed heterogeneous supercomputing management system, IEEE Comput. 26 (6) (1993) 78–86.

[HoK99] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, M.B. Srivastava, Power optimization of variable-voltage core-based systems, IEEE Trans. Computer-Aided Design Integrated Circuits Systems 18 (12) (1999) 1702–1714.

[IbK77] O.H. Ibarra, C.E. Kim, Heuristic algorithms for scheduling independent tasks on non-identical processors, J. ACM 24 (2) (1977) 280–289.

[IvO95] M.A. Iverson, F. Ozguner, G.J. Follen, 'Parallelizing existing applications in a distributed heterogeneous environment,' 1995 Heterogeneous Computing Workshop (HCW '95), 1995, pp. 93–100.

[Jai91] R. Jain, The Art of Computer Systems Performance Analysis Techniques for Experimental Design, Measurement, Simulation, and Modeling, Wiley, New York, 1991.

[KaA98] M. Kafil, I. Ahmad, Optimal task assignment in heterogeneous distributed computing systems, IEEE Concurrency 6 (3) (1998) 42–51.

[KhP93] A.A. Khokhar, V.K. Prasanna, M.E. Shaaban, C.L. Wang, Heterogeneous computing: challenges and opportunities, IEEE Comput. 26 (6) (1993) 18–27.

[LuZ00] P. Luh, X. Zhao, Y. Wang, L. Thakur, Lagrangian relaxation neural networks for job shop scheduling, IEEE Trans. Robotics Automat. 16 (1) (2000) 78–88.

[MaA99] 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 Distributed Comput. 59 (2) (1999) 107–121.

[MaB99] 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, NY, 1999, pp. 679–690.

[MaM03] D. Marinescu, G. Marinescu, Y. Ji, L. Boloni, H.J. Siegel, Ad hoc grids: communication and computing in a power constrained environment, Workshop on Energy-Efficient Wireless Communications and Networks 2003 (EWCN 2003), 22nd International Performance, Computing, and Communications Conference (IPCCC), April 2003.

[MiF00] Z. Michalewicz, D.B. Fogel, How to Solve It: Modern Heuristics, Springer, New York, NY, 2000.

[MiR03] R. Mishra, N. Rastogi, Z. Dakai, D. Mosse, R. Melhem, Energy aware scheduling for distributed real-time systems, International Parallel and Distributed Processing Symposium 2003 (IPDPS 2003), April 2003, pp. 22–26.

[Rud94] G. Rudolph, Convergence analysis of canonical genetic algorithms, IEEE Trans. Neural Networks 5 (1) (1994) 96–101.

[SiD97] H.J. Siegel, H.G. Dietz, J.K. Antonio, Software support for heterogeneous computing, in: A.B. Tucker, Jr. (Ed.), The Computer Science and Engineering Handbook, CRC Press, Boca Raton, FL, 1997, pp. 1886–1909.

[SiY96] H. Singh, A. Youssef, Mapping and scheduling heterogeneous task graphs using genetic algorithms, in: 5th IEEE Heterogeneous Computing Workshop (HCW 1996), 1996, pp. 86–97.

[SrP94] M. Srinivas, L.M. Patnaik, Genetic algorithms: a survey, IEEE Comput. 27 (6) (1994) 17–26.

[YaD95] F. Yao, A. Demers, S. Shenker, A scheduling model for reduced CPU energy, IEEE Annual Foundations Comput. Sci. (1995) 374–382.

[YuP02] Y. Yu, V.K. Prasanna, Power-aware resource allocation for independent tasks in heterogeneous real-time systems, 9th International Conference on Parallel and Distributed Systems, December 2002, pp. 341–348.

[WaS97] 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 Distributed Comput. 47 (1) (1997) 8–22.

[WuS00] M.-Y. Wu, W. Shu, 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.