Available online at www.sciencedirect.com

SCIENCE DIRECT®

Parallel Computing 31 (2005) 671–690

PARALLEL COMPUTING

www.elsevier.com/locate/parco

Mapping subtasks with multiple versions on an ad hoc grid

S. Shivle [a], P. Sugavanam [a], H.J. Siegel [a,b,*], A.A. Maciejewski [a], T. Banka [a], K. Chindam [a], S. Dussinger [c], A. Kutruff [a], P. Penumarthy [a], P. Pichumani [a], P. Satyasekaran [a], D. Sendek [a], J. Smith [d], J. Sousa [c], J. Sridharan [a], J. Velazco [e]

[a] *Colorado State University, Department of Electrical and Computer Engineering, Fort Collins CO 80523-1373, USA*

[b] *Colorado State University, Department of Computer Science, Fort Collins, CO 80523, USA*

[c] *HP Technologies, Fort Collins, CO 80528-9544, USA*

[d] *IBM Corporation, Boulder, CO 80301, USA*

[e] *Stelex, San Juan, Puerto Rico*

Received 31 October 2004; received in revised form 15 March 2005; accepted 15 April 2005
Available online 16 June 2005

Abstract

An ad hoc grid is a heterogeneous computing system composed of mobile devices. Each computing resource is constrained in battery energy. The problem being studied is to assign statically computing resources to the subtasks of an application that has an execution time constraint, when the resources are oversubscribed. All subtasks must be executed; to accommodate this in an oversubscribed environment, each subtask has two versions: the primary or

* Corresponding author. Address: Colorado State University, Department of Computer Science, Fort Collins, CO 80523, USA.

E-mail addresses: ssameer@engr.colostate.edu (S. Shivle), prasanna@engr.colostate.edu (P. Sugavanam), hj@colostate.edu (H.J. Siegel), aam@engr.colostate.edu (A.A. Maciejewski), tarunb@engr.colostate.edu (T. Banka), kiran@engr.colostate.edu (K. Chindam), sjd@fc.hp.com (S. Dussinger), kutruffa@lamar.colostate.edu (A. Kutruff), prashyp@engr.colostate.edu (P. Penumarthy), prkash@engr.colostate.edu (P. Pichumani), moses@engr.colostate.edu (P. Satyasekaran), sendekdm@lamar.colostate.edu (D. Sendek), bigfun@us.ibm.com (J. Smith), jso@fc.hp.com (J. Sousa), jaya@engr.colostate.edu (J. Sridharan), jose.velazco@abbott.com (J. Velazco).

0167-8191/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.parco.2005.04.003

full version, and the secondary or degraded version. The secondary version utilizes only 10% of the resources that the primary version requires, and produces only 10% of the data output for the subsequent children subtasks. Thus, the degraded version (secondary version) represents a reduced capability of lesser overall value, while consuming fewer resources. The goal is to assign resources so that the application meets an execution time constraint and the battery energy constraint while minimizing the number of degraded versions used. Five resource allocation heuristics to derive near-optimal solutions to this problem are presented, evaluated, and compared.

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

1. Introduction and problem statement

An ad hoc grid is a heterogeneous computing (HC) and communication system, all of whose components are mobile [18]. Each computing resource is constrained in battery energy. 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. In all of these cases, a grid-like environment is necessary to support reliably the coordinated effort of a group of individuals working under extreme conditions.

An important research problem is how to assign (match) resources to the subtasks and order (schedule) the execution of these subtasks that are matched to maximize some performance criterion of an 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., [9,11,14]). Thus, the development of heuristic techniques to find near-optimal solutions for the mapping problem is an active area of research (e.g., [1,2,5–8,10,17,19,28]).

For this research, a single, large application task composed of S communicating subtasks with data dependencies among them is to be mapped to machines in an ad hoc grid. There is a maximum execution time allowed for the application. All subtasks must be executed; to accommodate this in an oversubscribed environment, it is assumed that each subtask of the application task has two versions that could be executed. There is a primary or preferred version, called the full version, and a secondary or lesser preferred version, called the degraded version, utilizing only 10% of the resources that the full version requires, and producing only 10% of the data output for the subsequent children subtasks. Thus, the degraded version (secondary version) represents a reduced capability of lesser overall value, while consuming fewer resources. For example, a degraded version of a subtask may sample only 10% of the input data points that a full version samples. The resource utilization for the degraded version is arbitrarily set to be 10% for this study.

Each machine in the grid has some initial battery energy available at the beginning of the mission. Every time a subtask is executed on a machine it uses up some of the battery energy on that machine and hence the available battery energy on each machine becomes a constraint while mapping the application task to the grid. An additional constraint is the execution time, during which the entire application task must finish executing.

This study follows the work done in [23], which focused on energy management in ad hoc grids. However, unlike this study, the application task in [23] did not have versions for its subtasks and the environment was such that there were enough resources available for the application task to complete within the energy and time constraints using full versions for all subtasks. The problem studied in [23] focused on statically assigning resources in an ad hoc grid to an application composed of communicating subtasks in such a way as to minimize the average percentage of energy consumed by the application to execute in the ad hoc grid. The mapping heuristics were allowed to take up to 1 h to derive an allocation. This precomputed allocation was then used when the application was actually deployed in a mission.

However, what if some of the machines fail or are not available when the application is about to be deployed in the actual mission? In that case, due to lack of resources, it will not be possible to execute the application task and successfully complete the mission within the energy and time constraints using the precomputed static allocation of full versions of subtasks to machines. Some of the subtasks of the application task will then be forced to receive degraded service. This was the idea behind having versions for subtasks in the application task. Furthermore, this new allocation, using machines in the ad hoc grid that are functioning at the time of deployment, must be derived quickly. In this study, we set the maximum time for a heuristic to derive the new allocation to be 60 s (in contrast to the 60 min in [23]).

The mission for this study is assumed to be less than a day-long operation. It is assumed that any of the subtasks of the application task must receive all external inputs before beginning execution and generate all external results after completing execution. These are in addition to the global data items that one subtask sends to another. An example of this for a hypothetical wildfire-fighting mission is shown in Fig. 1. Also, any of the subtasks of the application task can use either its full or degraded versions irrespective of what version their parent subtask(s) used. For the wildfire-fighting example, even if the degraded version of a subtask is used for determining the temperature of the environment, the degraded output can be used by a full version subtask to analyze the effect of wind speed on the temperature.

The studied HC environment is designed such that once a subset of machines fails it is not possible to map all the subtasks using their full version within the application execution time constraint and battery energy constraints. The goal of this study is to maximize the number of full version (versus degraded version) subtasks that can be executed while still completing the application task within the energy constraint and the application execution time constraint τ. The more full versions used, the higher the quality of the calculations.

Five different heuristic approaches and a lower bound have been designed and compared via simulations to solve this ad hoc grid resource allocation problem. We address two methods of solving this problem. One of the heuristics initially uses the previous static mapping for subtasks mapped to all the machines from [23]

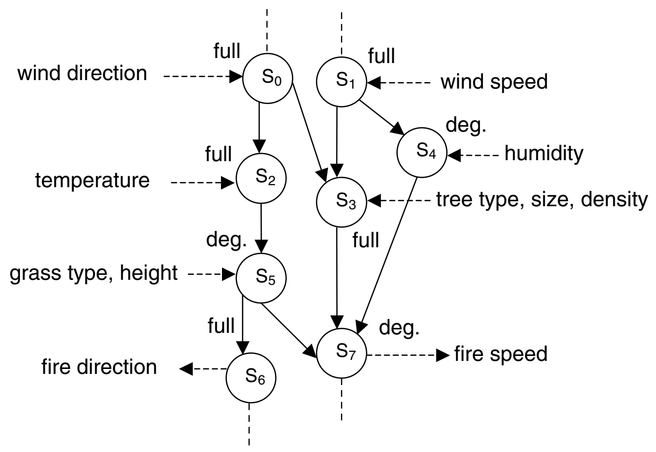

Fig. 1. External inputs and external outputs for a hypothetical wildfire-fighting example.

(assuming that all the machines in the grid are working) and then derives a new mapping for only the subtasks mapped to the failed machines. The other four heuristics derive a new mapping for the entire application task to the set of machines that are still available.

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

2. Simulation setup

In this study, the application task is composed of $S = 1024$ communicating subtasks. This large number of subtasks is chosen to present a significant mapping challenge for each heuristic. The data dependencies among the subtasks are represented by a directed acyclic graph (DAG). The pseudocode to generate the DAG is given in the appendix of this paper. For this study, ten 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 ten DAGs generated are 12 and 2, 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 sampled from a Gamma distribution [21], with a mean value of 2.8 Mbits and a variance of 1.4 Mbits.

Initially, for the baseline grid configuration (i.e., when all machines are present), it is assumed that there are a total of eight machines in the ad hoc grid and these are divided equally into two classes: “fast machines” (e.g., laptops) and “slow machines” (e.g., PDAs). It is further assumed that when the mapping is actually deployed (e.g., the wildfire fighters arrive at the fire scene), it is discovered that some machines have

Table 1
Simulation configurations for post failure mapping problem

Configuration	# Fast machines	# Slow machines
Case A	2	2
Case B	2	1
Case C	1	2

failed and only a subset of them are available for use. At this point, a fast mapping heuristic must decide how the subtasks are to be allocated among the functioning machines.

For this study, three different ad hoc grid failure cases are considered as shown in Table 1. Case A represents the grid configuration where two fast and two slow machines are present; Case B has two fast and one slow machine; and Case C has one fast and two slow machines.

The estimated expected execution time for each subtask on each machine is assumed to be known a priori as a result of experimentation with the known application. The estimated time to compute (ETC) values are used by the mapping heuristics. The estimated execution time of subtask i on machine j is ETC(i,j). The ETC values for all subtasks, taking heterogeneity into consideration, are generated using the Gamma distribution method described in [3]. For this research, a task mean and coefficient of variation (COV) are used to generate the ETC matrices. The mean subtask execution time is chosen to be 100 s and a COV of 0.9 is used to generate an ETC matrix with high task and high machine heterogeneity. For this study, ten different ETC matrices are generated and used with each of the ten DAGs to create 100 different data set scenarios.

When the ETC matrix is initially generated, the values for both slow and fast machines are generated in the same way, as given above. Arbitrarily, half of the initial eight machines are designated as "slow" and the other half as "fast." The ETC values must be adjusted so that the machines designated as "slow" are indeed slower than those designated as "fast."

To obtain the two classes of machines, all the ETC values for the slow machines are adjusted by a multiplicative factor (MF) [23]. For each subtask i the ratio diff_i of the ETC value of the fastest slow machine to the ETC value of the slowest fast machine is calculated as

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

Then the value of MF is given by

$$\mathrm{MF} = 2/(\min \mathrm{diff}_i \text{ for } i \in [0,\ 1023]).$$

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

The ETC values obtained using the above procedure correspond to the estimated time to compute values for full version subtasks. The degraded version of a subtask utilizes only 10% of the resources that the full version needs; hence, its estimated execution time was 10% of that given in the ETC matrix.

The ad hoc grid model that is considered for this project is a simplified version of an actual one. In this research, the following simplifying assumptions are made to make the ad hoc grid model more manageable:

(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 and not transmitting);
(d) The energy consumed to run the mapping heuristic is ignored;
(e) The machines are not multitasking, i.e., the subtasks are executed sequentially within a machine; and
(f) The energy to transmit any external outputs (e.g., the predicted direction of the fire in a wildfire scenario) is negligible.

Each machine j has four energy parameters associated with it:

(a) Maximum battery energy: $B(j)$;
(b) Rate at which it consumes energy for subtask execution, per ETC time unit: $E(j)$;
(c) Rate at which it consumes energy for subtask data *transmission*, per communication time unit: $C(j)$; and
(d) The machine's communication bandwidth: $BW(j)$.

Parameters (b) and (c) use a simplified model of real energy consumption.

The values of $B(j)$, $C(j)$, $E(j)$, and $BW(j)$ for both fast and slow machines are shown in Table 2. 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 for personal digital assistant (PDA) computers, such as the DELL Axim X5 that uses an Intel PXA255 processor operating at 400 MHz.

The energy consumed for executing the full version of a single subtask i on machine j is $\text{ETC}(i,j) \times E(j)$ and the energy consumed for executing the degraded version

Table 2
The values of $B(j)$, $C(j)$, $E(j)$, and $\text{BW}(j)$ for fast and slow machines

	Fast machines	Slow machines
$B(j)$ (energy units)	580	58
$C(j)$ (energy units/s)	0.2	0.002
$E(j)$ (energy units/s)	0.1	0.001
$\text{BW}(j)$ (Mbits/s)	8	4

of a single subtask i on machine j is $0.1 \times \mathrm{ETC}(i,j) \times E(j)$. The time required to transfer one bit of a data item between machine j and machine k is the inter-machine communication time called $\mathrm{CMT}(j,k)$ and is given by:

$$\mathrm{CMT}(j,k) = 1/\min(\mathrm{BW}(j), \mathrm{BW}(k)).$$

The energy consumed to send a data item g of size $|g|$ from machine j to machine k by a full version subtask is $\mathrm{CMT}(j,k) \times C(j) \times |g|$. A degraded version subtask produces only 10% of the data output of a full version subtask for its subsequent children subtasks. Hence, the energy consumed to send a data item g of size $|g|$ from machine j to machine k by a degraded version subtask is $0.1 \times \mathrm{CMT}(j,k) \times C(j) \times |g|$. The value 10% was chosen arbitrarily to scale the energy consumption of the degraded versions without loss of generality. When machines fail the system becomes oversubscribed. That is, not all of the subtasks can be completed within the time and energy constraints without employing some degraded version subtasks. The amount of degradation influences the count of subtasks that must use their degraded versions, but the general approach is unaffected.

Each machine can transfer data to only one destination at a time, and can do so while it is computing. A machine can simultaneously handle one outgoing data transmission and one incoming data reception. Similar to the study in [27], 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 value of the time constraint τ was chosen in [23] so that it ensured that all the machines were utilized, while assuming the baseline grid configuration (i.e., when all machines were present and working) and that all the subtasks used their full versions. Experimentation with a simple greedy mapping heuristic was used to determine the value of τ as 34,075 s.

The underlying assumptions are that the battery energy and maximum time allowed to execute the application are hard constraints, and that it will be necessary for the heuristics to use degraded versions of some subtasks to meet the constraints. Five static mapping schemes are studied in this paper: Recursive Bisection, Post Failure Bottoms Up, Post Failure Genetic Algorithm, Post Failure Min–Min, and Post Failure MCT. The performance of each heuristic is studied for each of the three ad hoc grid configurations (in Table 1) and across the 100 different scenarios. The time for each mapper itself to execute is required to be less than or equal to 60 s on a typical unloaded (i.e., running no other application) 1 GHz desktop machine. This time was selected to reflect an acceptable delay when machines are found to have failed as the mission is about to be deployed.

3. Related work

A significant amount of research has been performed in the areas of power constrained resource management in uniprocessors [12,25] and also in heterogeneous multiprocessors [20,26]. In all of these studies, 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. However, in our study we assume a power constrained heterogeneous computing environment where the CPUs that are heterogeneous in nature operate only at one voltage level as is true for most processors currently available. Also, the application task is composed of communicating subtasks with multiple versions and the execution time of each subtask on each CPU is known in advance. The goal is then to complete executing the application task within the specified time constraint and energy constraint while mapping as many full version (versus degraded) subtasks as possible.

The literature was examined to select a set of heuristics appropriate for the HC environment considered here. The heuristics generated here include some new heuristics that are based on earlier approaches, but designed to meet the goal of this study. The Min–Min heuristic approach has proven to be a good heuristic for dynamic and static mapping problems in earlier studies (e.g., [8,14,17]). The phase 1 of the Post Failure Min–Min and the Post Failure Bottoms Up heuristics used in this study are related to the Min–Min heuristic. In phase 1, Post Failure Bottoms Up assigns subtasks to machines in a manner similar to the Min–Min heuristic, but considers subtasks for scheduling in a different manner. The phase 1 of the Recursive Bisection heuristic, which just divides a DAG into levels, has been a part of other heuristics, such as the Earliest Task First heuristic studied in [13] for scheduling tasks on homogeneous processors, and the Fast Load Balancing heuristic [22] and Levelized Mean Time heuristic [27] for heterogeneous systems. Part of phase 1 of the Recursive Bisection heuristic also is related to Min–Min. The Post Failure MCT heuristic is related to the MCT heuristic from earlier studies (e.g., [8,17]).

Genetic Algorithms are a technique used for searching large solution spaces and have been used for mapping subtasks to machines in an HC environment (e.g., [8,24,27]). The phase 1 of the Post Failure Genetic Algorithm used in this study is based on [27] and has been modified for this problem environment.

4. Heuristics

For all the heuristics, except Post Failure Bottoms Up, only the subtasks whose predecessors are mapped could be considered during a given mapping iteration (referred to as mappable subtasks). After mapping a subtask, all heuristics are required to update the time and energy used for both subtask execution and intermachine communication. The makespan is defined as the total execution time of the entire application task on the machine suite in the ad hoc grid. Hence, the final makespan of any mapping must be less than or equal to τ.

The Post Failure Bottoms Up, Post Failure Genetic Algorithm, Post Failure Min–Min, and Post Failure MCT heuristics all have two phases. In phase 1 for all these heuristics except Post Failure Genetic Algorithm, subtasks are assigned to machines using their full versions, ignoring energy and makespan constraints. For phase 1 of the Post Failure Genetic Algorithm heuristic, it is assumed that all the machines are present in the grid and a static mapping is performed. In phase 2, this initial mapping is then modified based on the knowledge that a specific subset of machines is no longer available in the ad hoc grid. Some of the subtasks are converted to their degraded versions, so that the new mapping does not violate either the energy or time constraint. This section describes the five heuristics that were studied.

4.1. Recursive Bisection

The Recursive Bisection (RB) heuristic sorts all subtasks using their full versions in descending order of their average ETC times across all machines, to form list L. This ordering of subtasks in list L is used to determine the version of the subtask to be mapped. In a manner similar to that used in [15] and as shown in Fig. 2, subtasks are assigned to levels depending on the data precedence constraints.

The lowest numbered level consists of subtasks with no predecessors and the highest numbered level consists of subtasks with no successors. Each of the other subtasks is at one level number above the highest producer of its global data items. Subtasks are considered for mapping from the lowest level number to the highest level number. Similar to [13,22], for each level, a list of mappable subtasks (subtasks whose predecessors are mapped) is formed. The machine that gives the minimum completion time (ignoring other unmapped subtasks) for each of the unmapped subtask is determined. From these subtask/machine pairs, the pair that gives the smallest completion time is selected and the subtask is mapped onto that machine. Resource utilization is then updated. This procedure is repeated until all subtasks in that level are mapped.

Initially, all the subtasks are mapped to their minimum completion time machines using their full versions. After all subtasks have been mapped, the entire mapping is evaluated to ensure that energy and makespan constraints have been met. If either the energy or makespan constraint is violated, then using the bisection procedure, described below, some of the subtasks are converted to their degraded versions.

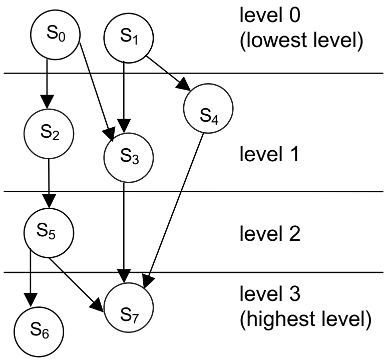

Fig. 2. Levelizing of subtasks S_0, S_1, S_2, S_3, S_4, S_5, S_6 and S_7 for a given sample DAG.

Bisection procedure

The bisection procedure performs a binary search on the list L for the minimum number of degraded version subtasks. The procedure begins by attempting to convert the subtasks in the first half of the sorted list L to their degraded version to meet the energy and makespan constraints. Each successive iteration attempts to increase (or decrease) the number of degraded version subtasks by increasing by half (or decreasing by half) the number of subtasks converted in the previous iteration. Subtasks are converted to their full version if the energy and makespan constraints were met by the previous iteration of the search. Similarly, subtasks are converted to their degraded version if either constraint was not met. Subtasks are converted to their degraded versions from the beginning of the sorted list L, and converted to their full versions from the bottom of the sorted list L. The procedure terminates when an iteration converts only one subtask to either its full or degraded version.

4.2. Post Failure Bottoms Up

In phase 1, Post Failure Bottoms Up (PFBU) assigns subtasks to levels in a manner similar to the RB heuristic. However, unlike RB, the PFBU heuristic begins by mapping subtasks from the highest level number. Thus, for the PFBU heuristic, the set of mappable subtasks at any given time consists of all subtasks that have no successors and all subtasks whose successors have previously been mapped.

Let the time for execution and output communication of subtask i on machine j, normalized with respect to the maximum time required for execution and output communication by subtask i across all machines be $\mathrm{NT}(i,j)$. 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 $\mathrm{NE}(i,j)$. The fitness value γ_{ij} for each mappable subtask on each machine is calculated as

$$\gamma_{ij} = \mathrm{NT}(i,j) + \mathrm{NE}(i,j).$$

For each mappable subtask considered, the machine that gives the minimum fitness value is determined (ignoring other unmapped subtasks). From these subtask/machine pairs, the pair that gives the minimum fitness value is selected and the subtask is mapped onto that machine. Resource utilization and the set of mappable subtasks are then updated. This process of machine assignment (matching) is repeated for all subtasks in the application task. Once all unmapped subtasks are assigned to machines, the entire mapping is evaluated. Subtasks are scheduled for execution in the reverse order they were matched. The entire mapping is then evaluated. If the mapping does not meet either the energy or the makespan constraint, then phase 2 of the heuristic is executed.

In phase 2, the makespan constraint is met as follows. All the subtasks in the application task using their full versions and using the mappings obtained from phase 1 are sorted in the descending order of their execution times. Using this ordering, one by one each subtask is converted to its degraded version, until the makespan constraint is met. Every time a subtask is converted to its degraded version, the entire mapping is evaluated. After the makespan constraint is met, if the battery constraint is exceeded on any of the machines, then the following procedure is carried out. For each machine that exceeds its maximum battery energy, a list of all the subtasks mapped to that machine in descending order of the energy consumed is generated. Using this ordering, one by one the subtasks are converted to their degraded versions, until the energy constraint on that machine is met. This is done for all the machines that exceed their maximum battery energy. In this way the energy constraint is met on all machines.

4.3. Post Failure Genetic Algorithm

In phase 1 of the Post Failure Genetic Algorithm (PFGA), all the subtasks are mapped to machines using the Genetic Algorithm (GA) in [23], assuming that all the eight machines are available. In [23], the goal was to minimize the average percentage of energy consumed by the application (B_{pavg}) to execute in the ad hoc grid and the maximum execution time for the entire application was the constraint. Each chromosome represents one solution to the problem and a set of chromosomes is called a population. The GA used in [23] operates on a population of 100 chromosomes. Each chromosome is made of a scheduling string and a matching string, and has a fitness value (B_{pavg}) associated with it. 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. Similar to the approach in [27], these chromosomes then undergo selection, crossover, mutation, and evaluation. Please see [23] for the GA details.

All the subtasks that are mapped to machines that are labeled as failed are considered to be unmapped subtasks. These unmapped subtasks are selected for mapping in the order they appear in the scheduling string of the final mapping selected by the GA [23]. The set of mappable subtasks consists of all unmapped subtasks whose predecessors are mapped.

The unmapped subtasks are assigned machines using a fitness function. Let the time for execution of subtask i on machine j and the time for input communication, normalized with respect to the maximum time required for execution and input communication by subtask i across all machines, be $\text{NT}'(i,j)$. Let the energy consumed for execution of subtask i on machine j and energy for input communication, normalized with respect to the maximum energy consumed for execution and input communication of subtask i across all machines, be $\text{NE}'(i,j)$. The fitness value f_{ij} for the next mappable subtask in the scheduling string is then calculated on each machine as

$$f_{ij} = \text{NT}'(i,j) + \text{NE}'(i,j).$$

This unmapped subtask is then assigned to the machine that gives the minimum fitness value. Once all of the unmapped subtasks are assigned to machines, the entire mapping is evaluated. If the makespan and energy constraints are violated then they are met using the same procedure as that of phase 2 of PFBU.

4.4. Post Failure Min–Min

In phase 1, the Post Failure Min–Min (PFMM) heuristic (based on the concept in [14]) forms a set of mappable subtasks consisting of all unmapped subtasks whose predecessors are mapped. PFMM utilizes a fitness function to evaluate all mappable subtasks. The fitness function is the same as the f_{ij} function used for mapping unmapped subtasks by PFGA.

For each mappable subtask considered, the machine that gives the minimum fitness value is determined (ignoring other unmapped subtasks). From these subtask/machine pairs, the pair that gives the minimum fitness value is selected and the subtask is mapped onto that machine. Resource utilization and the set of mappable subtasks are then updated. This process continues until all subtasks are mapped. Once all the unmapped subtasks are assigned to machines, the entire mapping is evaluated. If the mapping does not meet either the energy or the makespan constraint then phase 2 of the heuristic is executed.

In phase 2, a time portion metric $\mathrm{CP}_{\mathrm{avg}}$ and an energy portion metric $\mathrm{EP}_{\mathrm{avg}}$ is calculated. Let $N(j)$ be the number of subtasks mapped to a machine j. Then,

$$\mathrm{CP}_{\mathrm{avg}} = \tau/N(j) \quad \text{and} \quad \mathrm{EP}_{\mathrm{avg}} = B(j)/N(j).$$

Subtasks are considered for conversion to their degraded version in the same order they were mapped in phase 1. If subtask i is the nth subtask mapped to machine j, then it is checked if the energy consumed by subtask i on machine j exceeds $\mathrm{EP}_{\mathrm{avg}}$ and also if its completion time exceeds $n \times \mathrm{CP}_{\mathrm{avg}}$. If either of the two quantities is exceeded, the subtask is converted to its degraded version. Every time a subtask is converted to its degraded version, the entire mapping is evaluated. If the mapping does not meet either the energy or makespan constraint, the next mapped subtask (in the order of mapping used in phase 1) is considered for conversion to its degraded version using the above procedure. In this way, the energy and time constraints are met.

Experiments also were conducted using phase 1 of PFMM with phase 2 of PFBU. This gave better results than using the above phase 2 for PFMM.

4.5. Post Failure MCT

For phase 1, the Post Failure Minimum Completion Time (PFMCT) heuristic based on [4] uses the minimum completion time machine to map a subtask. A mappable subtask is one whose predecessors are mapped. For each mappable subtask, in an arbitrary order, the machine that increases the makespan by the least amount if the subtask is mapped to it is determined and the subtask is mapped to it. The mapped subtask is removed from the mappable set and this process continues until all subtasks are mapped. Once all the subtasks are mapped to machines, the entire mapping is evaluated. If either the energy or the makespan constraints are violated then they are met using the same procedure as that of phase 2 of PFBU.

4.6. Lower bound

The method for calculating a lower bound on the number of degraded version subtasks assumes a system consisting of homogeneous minimum execution time (MET) fast machines and homogeneous MET slow machines. The execution time of each subtask on the homogeneous MET fast machines is the same and is equal to the minimum time that the subtask would take to execute across all the original set of fast machines. In a similar manner, the execution time for each subtask on the homogeneous MET slow machines is the same and is equal to the minimum time that the subtask would take to execute across all the original set of slow machines.

The energy consumed by each subtask to execute is related to time. The homogeneous MET system uses the minimum time to execute each subtask if it is mapped onto either the fast or slow machine. The lower bound found using the homogeneous system of MET fast and MET slow machines will hence be lower than or equal to the lower bound on the number of degraded versions using the set of machines available.

This lower bound method ignores the data precedence constraints and inter-machine communications. The energy consumption rate of a fast machine is 100 times more than the energy consumption rate of a slow machine and the average ETC value across slow machines was approximately only seven times the average ETC value across fast machines. Therefore, in general, if a subtask running on the fast machine were to be moved to the slow machine, it will consume less energy on the slow machine than on the fast machine. Let R_i be the fraction of energy that the slow machine will consume to execute subtask i as compared to the energy consumed by a fast machine. That is, if s_i and f_i are the energy consumed by the MET slow machine and MET fast machine to execute subtask i, respectively, then $R_i = s_i/f_i$, $i = 0,\ldots,1023$. The maximum of all R_i ($R_{\max}$) will give the worst case energy consumption by the MET slow machine as compared to the MET fast machine if any subtask was moved from the MET fast machine to the MET slow machine. That is, the amount of energy consumed by MET slow machine will be at most $R_{\max}$ times that consumed by the MET fast machine for executing any subtask. So, we have

$$R_{\max} = \max(s_i/f_i), \quad i = 0,\ldots,1023.$$

Recall, the energy initially usable on a slow machine is 58 units, and on a fast machine is 580 units. The effective total energy available (ETE) normalized based on the MET slow machine is given by the following equation:

$$\text{ETE} = (R_{\max} \times 580 \times \text{avail. \#fast machines}) + (58 \times \text{avail. \#slow machines}).$$

This constitutes the maximum pool of energy available to execute all subtasks after scaling the MET fast machine energy to MET slow machine energy. Additionally, the slow machines could use only 34.075 units of energy if the time constraint is considered (i.e., $\tau \times$ rate of energy consumed by MET slow machine = $34{,}075 \times 0.001$). The fast machines could consume all of the energy available within the time constraint. Hence, the energy on the slow machines could be limited as mentioned above and this results in the ETE equation being rewritten as

$$\begin{aligned}\mathrm{ETE} = {} & (R_{\max} \times 580 \times \text{avail. \#fast machines}) \\ & + (34.075 \times \text{avail. \#slow machines}).\end{aligned}$$

To find the minimum number of subtasks required to use their degraded versions, the set of all subtasks T are sorted in descending order based on their energy consumptions on the MET slow machine using full versions. If the subtasks are converted to their degraded version in order until the total energy consumed $\leqslant$ ETE, then the minimum number of subtasks will be converted to degraded versions. This is the same as maximizing the number of subtasks using full versions in executing all the subtasks within ETE.

Let T_1 be the set of subtasks using their full versions as found using the previously defined method. Then the lower bound on the number of degraded versions will be $|T| - |T_1|$. Define energy (i) to be the energy consumption of task i on the MET slow machine. To formally show that no more than T_1 subtasks can use their full versions consider the following proof.

Theorem. *There can be no more than T_1 subtasks running full versions that combined consume energy less than or equal to* ETE *where the energy constraint*: $\sum_{\forall t, t \in T_1} \mathrm{energy}(t) + 0.1 \times \sum_{\forall d, d \notin T_1} \mathrm{energy}(d) \leqslant \mathrm{ETE}$ *is met.*

Proof. Assume there is a set T_2 such that $T_2 \subseteq T$ and $|T_2| > |T_1|$. Obviously, to maximize $|T_2|$ requires the $|T_2|$ subtasks that consume the least amount of energy on the MET slow machine. Therefore, by definition $T_1 \subset T_2$, and the energy consumption for T_2 can be expressed in terms of the energy consumption of T_1 as follows:

$$\sum_{\forall t, t \in T_2} \mathrm{energy}(t) = \sum_{\forall a, a \in T_1} \mathrm{energy}(a) + \sum_{\forall b, b \in (T_2 - T_1)} \mathrm{energy}(b). \tag{1}$$

Expressing the energy constraint for T_2 using Eq. (1) gives:

$$\sum_{\forall t, t \in T_1} \mathrm{energy}(t) + \sum_{\forall a, a \in (T_2 - T_1)} \mathrm{energy}(a) + 0.1 \times \sum_{\forall d, d \notin T_2} \mathrm{energy}(d) \leqslant \mathrm{ETE} \tag{2}$$

By the definition of T_1, moving any subtask to T_1 would cause T_1 to violate the energy constraint, therefore moving any task from $(T_2 - T_1)$ to T_1 would violate the energy constraint. But Eq. (2) is equivalent to converting all $(T_2 - T_1)$ subtasks to their full version and by definition violates the energy constraint. Therefore the set $(T_2 - T_1)$ must be the empty set, implying that $|T_2| = |T_1|$ which contradicts the original assumption. □

5. Results

The simulation results for the post failure mapping problem are shown in Figs. 3 and 4. All heuristics were run for 10 different task graphs (DAGs), using 10 different ETCs (i.e., for a total of 100 different combinations) across three different cases of available machines in the ad hoc grid (Table 1). The average values and 95% confi-

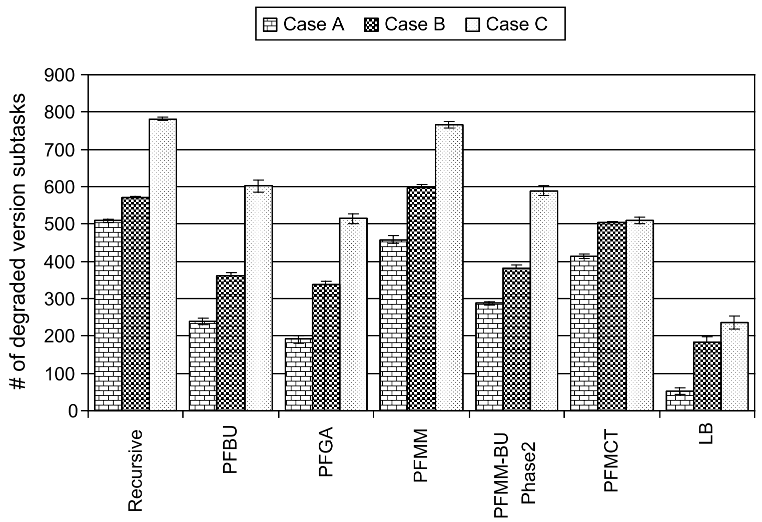

Fig. 3. The simulation results for number of subtasks that used full version subtasks for the different heuristics across the three different ad hoc grid scenarios.

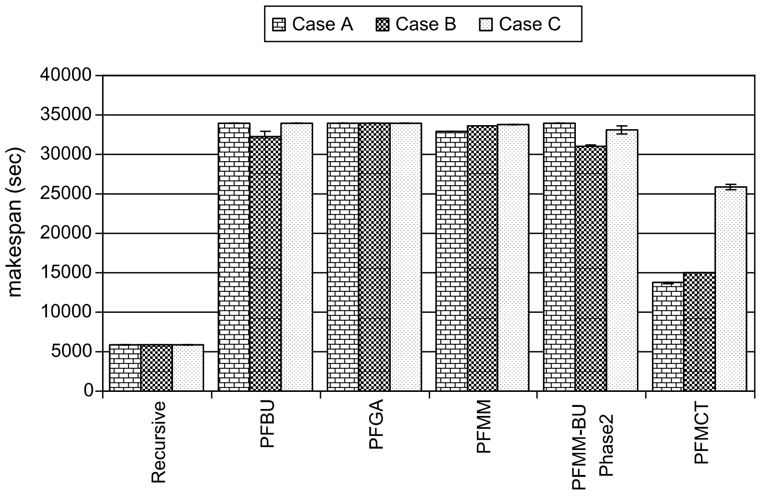

Fig. 4. The simulation results for makespan in seconds for the different heuristics across the three different ad hoc grid scenarios.

dence intervals [16] of the number of degraded versions used are plotted for all heuristics. The running times of the heuristics averaged over 100 trials, mapping 1024 subtasks per trial, are shown in Table 3.

As seen in Fig. 3, PFGA and PFBU were the best two heuristics for Case A and Case B, PFMCT and PFGA were the best two heuristics for Case C, and PFMM and RB performed poorly for all three cases. PFGA, PFBU, and PFMCT differed only by their initial subtask to machine assignments, they all used an identical

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

Heuristic	Average execution times (s)
Recursive Bisection	0.1
PFBU	0.38
PFGA	0.49
PFMM	0.84
PFMM-BU phase 2	0.36
PFMCT	0.56

procedure for phase 2 to convert subtasks to their degraded versions to meet the time and the energy constraints. PFBU used a fitness function to assign all the subtasks to machines, while PFGA used the initial static mapping and in addition a fitness function for all the unmapped subtasks. PFMCT assigned subtasks to machines that gave the minimum completion time. It was observed that PFMCT assigned more subtasks to fast machines as compared to PFGA and PFBU that had a more balanced load distribution, and hence PFMCT had a smaller final makespan as shown in Fig. 4. This also resulted in a larger number of degraded version subtasks for PFMCT in Cases A and B. For Case C that had only one fast machine, PFMCT was forced to assign many more subtasks to the two slow machines in addition to the one fast machine. Hence, unlike in Case A and B, PFMCT performed comparably to PFGA.

The Recursive Bisection heuristic also used completion time based Min–Min to map subtasks to machines and hence mapped a majority of the subtasks to fast machines. Recursive Bisection converted subtasks to their degraded versions in decreasing order of average execution time across all machines irrespective of whether the subtask was actually mapped to a fast machine or to a slow machine. Another problem with the RB heuristic is that while matching subtasks to machines in phase 1, only the execution times of subtasks is considered ignoring the communications. Almost every time, a subtask mapped to a fast machine was converted to its degraded version. This resulted in Recursive Bisection having a very small final makespan (as seen in Fig. 4) but a very large number of degraded version subtasks.

Phase 2 of PFMM considered subtasks for conversion in the order they were mapped depending upon a time factor and an energy factor that in turn depended upon the number of subtasks assigned to a particular machine. This procedure was found to perform poorly, as can be seen from Fig. 3. The phase 2 of PFMM attempted to force equal portions of time and energy for the subtasks assigned to the same machine and did not consider subtasks mapped to other machines. On the other hand, the phase 2 of PFBU considers all the subtasks mapped across all machines for conversion to the degraded version. Hence when the phase 2 of PFBU was used for PFMM there was a considerable improvement in the results across all the three scenarios as opposed to the results obtained using the previous phase 2 of PFMM.

6. Summary

Five heuristics were designed, developed, and simulated using the HC environment presented. Application tasks composed of communicating subtasks with data dependencies and multiple versions were mapped using the heuristics described in this research. The PFGA gave the best or comparable average performance in all cases. For the situations in cases A and B, which included two fast machines, the PFBU performed nearly as well while not requiring the computation of a complete static mapping a priori. For the case where there was only one fast machine available, PFMCT gave the performance comparable to PFGA. Thus, two approaches are recommended: a. use the PFGA with a priori static complete mapping; or b. use PFBU in cases where there are multiple fast machines available, and PFMCT where there is just one fast machine available.

The results of this work may be used in the development of ad hoc grids in support of many applications of importance, such as disaster management. All of the methods studied here can be extended to encompass a larger analysis. For example, the communication time for receiving data and energy consumption during idle time may be considered in the future work. Another extension of this model would be to associate priorities with subtasks, so that subtasks whose accuracies are more critical for a given mission will be less likely to be degraded. Thus, the model and heuristics presented here can form the basis for continued research in support of the use of ad hoc grids to perform applications of importance to society.

Acknowledgements

A preliminary version of portions of this material was presented at the 3rd International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks (HeteroPar '04). The authors thank Shoukat Ali, Jong-Kook Kim, Vladimir Shestak, Ashish Mehta, and Mohana Oltikar for their valuable comments. This research was supported in part by the Colorado State University George T. Abell Endowment and in part by IBM Corporation's Academic Learning Assistance Program.

Appendix A. 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 incom-
   ing 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 pseudocode.
```

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, Elsevier, Amsterdam, 2005, pp. 91–128.

[2] 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 and Distributed Computing Practices, Special issue on Algorithms, Systems and Tools for High Performance Computing, in press.

[3] S. Ali, H.J. Siegel, M. Maheswaran, D. Hensgen, S. Ali, Representing task and machine heterogeneities for heterogeneous computing systems, Tamkang Journal of Science and Engineering 3 (3) (2000) 195–207, Special 50th Anniversary Issue (invited).
[4] R. Armstrong, D. Hensgen, T. Kidd, The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions, in: 7th IEEE Heterogeneous Computing Workshop (HCW 1998), March 1998, pp. 79–87.
[5] 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), 15th International Parallel and Distributed Processing Symposium (IPDPS 2001), paper HCW 15, April 2001.
[6] I. Banicescu, V. Velusamy, Performance of scheduling scientific applications with adaptive weighted factoring, in: 10th IEEE Heterogeneous Computing Workshop (HCW 2001), 15th International Parallel and Distributed Processing Symposium (IPDPS 2001), paper HCW 06, April 2001.
[7] 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).
[8] 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, Bin Yao, A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, J. Parallel Distr. Comput. 61 (6) (2001) 810–837.
[9] E.G. Coffman Jr. (Ed.), Computer and Job-Shop Scheduling Theory, John Wiley & Sons, New York, 1976.
[10] M.M. Eshaghian (Ed.), Heterogeneous Computing, Artech House, Norwood, MA, 1996.
[11] D. Fernandez-Baca, Allocating modules to processors in a distributed system, IEEE Trans. Softw. Eng. SE-15 (11) (1989) 1427–1436.
[12] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, M.B. Srivastava, Power optimization of variable-voltage core-based systems, IEEE Trans. Comput. Aid. Des. Integr. Circ. Syst. 18 (12) (1999) 1702–1714.
[13] J.-J. Hwang, Y.-C. Chow, F.D. Anger, Y.C. Lee, Scheduling precedence graph in systems with interprocessor communication times, SIAM J. Comput. 18 (2) (1989) 244–257.
[14] O.H. Ibarra, C.E. Kim, Heuristic algorithms for scheduling independent tasks on non-identical processors, J. ACM 24 (2) (1977) 280–289.
[15] M.A. Iverson, F. Ozguner, G.J. Follen, Parallelizing existing applications in a distributed heterogeneous environment, in: 1995 Heterogeneous Computing Workshop (HCW '95), April 1995, pp. 93–100.
[16] R. Jain, The Art of Computer Systems Performance Analysis Techniques for Experimental Design, Measurement, Simulation, and Modeling, Wiley, New York, 1991.
[17] 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.
[18] D. Marinescu, G. Marinescu, Y. Ji, L. Boloni, H.J. Siegel, Ad hoc grids: communication and computing in a power constrained environment, in: Workshop on Energy-Efficient Wireless Communications and Networks 2003 (EWCN 2003), April 2003.
[19] Z. Michalewicz, D.B. Fogel, How to Solve It: Modern Heuristics, Springer-Verlag, New York, NY, 2000.
[20] R. Mishra, N. Rastogi, Z. Dakai, D. Mosse, R. Melhem, Energy aware scheduling for distributed real-time systems, in: International Parallel and Distributed Processing Symposium 2003 (IPDPS 2003), April 2003, pp. 22–26.
[21] A. Papoulis, Probability, Random Variables, and Stochastic Processes, McGraw-Hill, New York, NY, 1984.
[22] A. Radulescu, A.J.C. van Gemund, Fast and effective task scheduling in heterogeneous systems, in: 9th Heterogeneous Computing Workshop (HCW 2000), May 2000, pp. 229–238.
[23] S. Shivle, R. Castain, H.J. Siegel, A.A. Maciejewski, T. Banka, K. Chindam, S. Dussinger, P. Pichumani, P. Satyasekaran, W. Saylor, D. Sendek, J. Sousa, J. Sridharan, P. Sugavanam, J. Velazco,

Static mapping of subtasks in a heterogeneous ad hoc grid environment, in: 13th IEEE Heterogeneous Computing Workshop (HCW 2004), April 2004.
[24] M. Srinivas, L.M. Patnaik, Genetic algorithms: a survey, IEEE Comput. 27 (6) (1994) 17–26.
[25] F. Yao, A. Demers, S. Shenker, A scheduling model for reduced CPU energy, IEEE Ann. Foundations Comput. Sci. (1995) 374–382.
[26] Y. Yu, V.K. Prasanna, Power-aware resource allocation for independent tasks in heterogeneous real-time systems, in: 9th International Conference on Parallel and Distributed Systems, December 2002, pp. 341–348.
[27] 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 Distr. Comput. 47 (1) (1997) 8–22.
[28] M.-Y. Wu, W. Shu, H. Zhang, Segmented min–min: a static mapping algorithm for meta-tasks on heterogeneous computing systems, in: 9th IEEE Heterogeneous Computing Workshop (HCW 2000), May 2000, pp. 375–385.