Mapping of Subtasks with Multiple Versions in a Heterogeneous Ad Hoc Grid Environment

Sameer Shivle[1], H. J. Siegel[1,2], Anthony A. Maciejewski[1], Tarun Banka[1], Kiran Chindam[1], Steve Dussinger[1,3], Andrew Kutruff[1], Prashanth Penumarthy[1], Prakash Pichumani[1], Praveen Satyasekaran[1], David Sendek[1], J. Sousa[1,3], Jayashree Sridharan[1], Prasanna Sugavanam[1], and Jose Velazco[4]

Colorado State University
[1]Electrical and Computer Engineering Dept.
[2]Computer Science Dept.
Fort Collins, CO 80523, U.S.A
{ssameer, hj, aam, tarunb, kiran, prashyp, prkash, moses, jaya, prasanna}@engr.colostate.edu
{rcastain, sendekdm}@lamar.colostate.edu

[3]HP Technologies
Fort Collins, CO 80528-9544
{sjd, jso}@fc.hp.com

[4]Abbott Laboratories
Barceloneta, Puerto Rico 00617
jose.velazco@abbott.com

***Abstract*—An ad hoc grid is a heterogeneous computing system composed of mobile devices. The problem studied here is to statically assign resources to the subtasks of an application, which has an execution time constraint, when the resources are oversubscribed. Each subtask has a preferred version, and a secondary version that uses fewer resources. The goal is to assign resources so that the application meets its execution time constraint while minimizing the number of secondary versions used. Five resource allocation heuristics to derive near-optimal solutions to this problem are presented and evaluated.**

***Index Terms*— 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 without a fixed infrastructure (i.e., all of its components are mobile). *Ad hoc* grids allow a group of individuals to accomplish a mission that involves extensive 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 peacekeeping operations [16]. 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. If, for any reason, some of the machines in the *ad hoc* grid fail, then due to limitations on the resource availability, some of the subtasks of the application task will be forced to receive degraded service.

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., [8, 10, 12]). Thus, the development of heuristic techniques to find near-optimal solutions for the mapping problem is an active area of research (e.g., [1, 4, 5, 6, 7, 9, 11, 15, 17, 21]).

For this research, a single, large application task is considered to be composed of S communicating subtasks with data dependencies among them. Each subtask has two versions that could be executed. There is a primary version, called full version and a secondary version, called degraded version, utilizing only 10% of the resources that the primary version needs, and producing only 10% of the data output for the subsequent children subtasks. Thus, the degraded version (secondary version) represents a reduced capability, designed to provide some lesser overall value while consuming fewer resources. This application task is to be executed in an *ad hoc* grid as part of the mission being conducted.

Initially it is assumed that the simulated HC environment consists of M machines in the *ad hoc* grid. It is also assumed that a static mapping that maps the subtasks to these M machines using their full versions in the *ad hoc* grid already exists [18]. However, if for some reason a subset of machines fails, then the existing static mapping cannot be used and some of the subtasks will be forced to use their degraded versions

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

0-7695-2210-6/04 $20.00 © 2004

due to a lack of available resources. We address two ways of solving this problem. First, use the previous static mapping for subtasks mapped to the all the machines (assuming that all the machines in the grid are working) and derive a new mapping for only the subtasks mapped to the failed machines. Second, discard the entire previous static mapping and derive a new mapping for the entire application task to the set of machines that are still available.

The studied HC environment has been designed such that once a subset of machines fails it is not possible to map all the subtasks using their full version within the limited battery energy available. Hence, the available battery energy becomes a constraint. The goal of this study is to maximize the number of full version subtasks that can be executed while still completing the application task within the energy constraint in addition to an application execution time constraint τ. The underlying assumptions are that the battery power and maximum time allowed to execute the application are hard constraints, and that it will be necessary to use degraded versions of some subtasks to meet the constraints. In this study, the performance of five mapping heuristics in three different, *ad hoc* grid configurations is evaluated and compared. The wall clock time for each mapper itself to execute is required to be less than or equal to 180 seconds on a typical unloaded (i.e., running no other application) 1 GHz desktop machine.

The next section describes the simulation setup used for this research. Section 3 provides a list of 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 1,024 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 twelve and two, respectively. Also, for each DAG there are seven subtasks with no predecessors, seven subtasks with no successors, and the remaining 1,010 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, with a mean value of 2.8 megabits and a variance of 1.4 megabits.

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 simulated *ad hoc* grid and these are divided equally into two classes: "fast machines" and "slow machines."

The three different *ad hoc* grid configurations, each with less than eight machines are 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.

Table 1: Simulation configurations.

configuration	# fast machines	# slow machines
Case A	2	2
Case B	2	1
Case C	1	2

The estimated expected execution time for each subtask on each machine is assumed to be known *a priori*. 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, were generated using the Gamma distribution method described in [2]. 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 seconds and a COV of 0.9 was used to generate an ETC matrix with high task and high machine heterogeneity. For this study, ten different ETC matrices were generated and used with each of the ten DAGs to create 100 different scenarios.

To obtain the two classes of machines, all the ETC values for the slow machines are adjusted by a multiplicative factor (MF) [18]. 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

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

Then the value of MF is given by $MF = 2 / (\min diff_i$ for $i \in [0, 1023])$. All the ETC values for the slow machines are now multiplied by the MF to get the new adjusted values. After creating the two classes of machines, the new mean estimated execution time for a single subtask is 131 seconds. For this study, across all

the subtasks in an ETC matrix, the average fastest machine is approximately ten times faster than the average slowest machine. 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 communication, 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 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: $CMT(j,k) = 1 \,/\, \min\left(BW(j), BW(k)\right)$. The energy consumed to send a data item g of size $|g|$ from machine j to machine k is $CMT(j, k) \times C(j) \times |g|$. 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 [20], 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; and
c. a machine consumes no energy if it is idle (i.e., not computing or not transmitting).

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 1.7GHz. 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.

Table 2: 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 value of the time constraint τ is chosen so that it ensures that all machines are utilized. A simple greedy mapping heuristic was used to determine the value of τ as 34,075 seconds. The performance of each heuristic is studied for each of the three *ad hoc* grid configurations and across the 100 different scenarios.

3. Related Work

The literature was examined to select a set of heuristics appropriate for the HC environment considered here. The ETC matrices and the DAGs were generated using the same method as in [18]. However, in [18] the subtasks did not have versions and also the performance metric was to minimize the battery energy consumed, unlike in this study, where the battery energy is a constraint.

Min-Min, Genetic Algorithm, and MCT, have been used previously to map tasks onto heterogeneous machines (e.g., [7]). The performance goal in [7] was to minimize the total time required to complete an application task, whereas the goal of our study is to minimize the number of subtasks using degraded versions mapped while completing the entire application task within the available resources and within a time constraint. The Min-Min heuristic has proven to be a good heuristic for dynamic and static mapping problems in earlier studies (e.g., [7, 15]). 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 for scheduling in a different manner. Genetic Algorithms are a technique used for searching large solution spaces and have been used for mapping subtasks to machines in a HC environment (e.g., [7, 19, 20]). The Genetic Algorithm used in this study is a slightly modified version of the one used in [20].

4. Heuristics

For all the heuristics, except Bottoms Up, only the subtasks whose predecessors had been fully mapped (referred to as mappable subtasks) could be considered during a given mapping iteration. After mapping a subtask, all heuristics were required to update the time and energy used for both subtask execution and inter-machine communication. The makespan is defined as the overall execution time of the application task on the machine suite in the *ad hoc* grid. So the final makespan of all mappings has to be less than or equal to τ.

The Bottoms Up, Genetic Algorithm, Min-Min, and MCT heuristics all have two phases. In the first phase, subtasks are assigned to machines using their full versions, ignoring energy and makespan constraints. In the second phase, some of the subtasks are converted to their degraded versions, such that the new mapping does not violate either the energy or time constraint. Also, the Genetic Algorithm heuristic initially assumes that all the machines are present in the grid and performs a static mapping. 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. This section describes the five heuristics that were studied.

4.1. Recursive Bisection

Using their full versions, all subtasks are sorted in descending order of their average ETC times across all machines to form List 1. This ordering of subtasks in List 1 is used to determine the version of the subtask to be mapped.

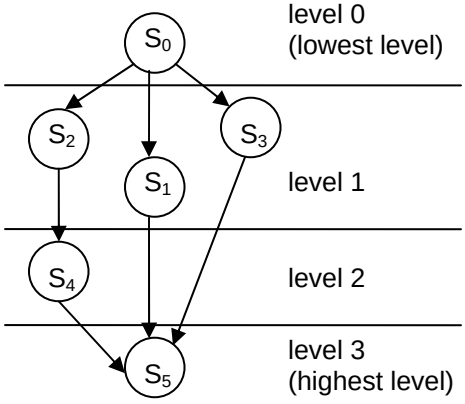

Figure 1: Levelizing of subtasks S_0, S_1, S_2, S_3, S_4, and S_5 for a given sample DAG.

In a manner similar to that used in [13] and as shown in Figure 1, the Recursive Bisection (RB) heuristic assigns subtasks to 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 of the other subtasks is at one level above the highest producer of its global data items. Subtasks are mapped to their minimum completion time machine in order from the lowest-level subtask to the highest-level subtask. Within a level, subtasks are chosen randomly for mapping.

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 schedule, described below, some of the subtasks are converted to their degraded versions.

Bisection Schedule

If the bisection schedule is needed, the first 512 subtasks (half of the total number of subtasks) in List 1 are selected to be degraded version subtasks. All subtasks are then mapped using the mapping procedure described earlier. At the end of the mapping, depending on the makespan and the battery energy on each of the machines, it is decided whether to increase or decrease the number of degraded version subtasks. If either the energy or time constraint is violated, then the number of degraded version subtasks is increased by half of the current number. If the energy or time constraint is not violated, then the number of degraded version subtasks is decreased by half of the current number. This process of bisection is repeated until the number of degraded version subtasks cannot be reduced any more within the energy and makespan constraints.

4.2. Bottoms Up

In the first phase, Bottoms Up (BU) assigns subtasks to levels in a manner similar to the RB heuristic. However, unlike RB, 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 have no successors and all subtasks whose successors have previously been mapped. Subtasks within each level are randomly selected for mapping.

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 $\underline{NT}(i, j)$. 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 $NE(i, j)$. The fitness value γ_{ij} for each mappable subtask on each machine is calculated as $\gamma_{ij} = NT(i, j) + NE(i, j)$.

For each mappable subtask considered, the machine that gives the minimum fitness value is determined and the subtask is assigned to that machine. This process of machine assignment (matching) is repeated for all subtasks in the application task. The 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 the second phase of the heuristic is executed.

In the second phase, the makespan constraint is met as follows. All the subtasks in the application task using their full versions and using the mappings obtained from the first phase 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.

4.3. Genetic Algorithm

Initially all the subtasks are mapped to machines using the Genetic Algorithm (GA) in [18], assuming that all the eight machines are available (please see [18] for the GA details due to the length constraints). All the subtasks that were mapped to machines that are labeled as failed are considered to be unmapped subtasks and a list of these unmapped subtasks is formed. These unmapped subtasks are selected for mapping in the order they come in the scheduling string of the GA [18]. The unmapped subtasks are assigned machines using a fitness function. For each unmapped subtask i, the maximum ETC value (maxETC) and the maximum energy value (maxEN) needed to execute this subtask on any machine, across the available set of machines is found. The fitness value f_{ij} is then calculated as:

$$f_{ij} = [ETC(i, j) / \text{maxETC}] + [(ETC(i, j) \times E(j)) / \text{maxEN}].$$

The 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 the second phase of Bottoms Up.

4.4. Min-Min

For the first phase, the Min-Min heuristic (based on the concept in [12]) utilizes a fitness function to evaluate all mappable subtasks. The fitness function, similar to that used for mapping unmapped subtasks by GA is calculated as:

$$f_{ij} = [ETC(i, j) / \text{maxETC}] + [(ETC(i, j) \times E(j)) / \text{maxEN}].$$

For each mappable subtask considered, the machine that gives the minimum fitness value is determined. 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 is 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 the second phase of the heuristic is executed.

In the second phase, a time metric CT_{avg} and an energy metric EC_{avg} is calculated. If $N(j)$ is the number of subtasks mapped to a machine j then, $CT_{avg} = \tau / N(j)$ and $EC_{avg} = B(j) / N(j)$. Subtasks are considered for conversion to their degraded version in the same order they were mapped in the first phase. If subtask i is the n^{th} subtask mapped to machine j, then it is checked if the energy consumed by subtask i on machine j exceeds EC_{avg} and also if its completion time exceeds $n \times CT_{avg}$. 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 the first phase) is considered for conversion to its degraded version using the above procedure. In this way, the energy and time constraints are met.

4.5. MCT

For the first phase, the MCT heuristic based on [3] uses the minimum completion time machine to map a subtask. For each mappable subtask considered, 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. 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 the second phase of Bottoms Up and Genetic Algorithm.

5. Results

The simulation results are shown in Figures 2 and 3. 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 grid scenarios. Their average values and 95% confidence intervals [14] are plotted. The running times of the heuristics averaged over 100 trials, mapping 1024 subtasks per trial, are shown in Table 3.

As seen in Figure 2, GA and BU were the best two heuristics for Case A and Case B, MCT and GA were the best two heuristics for Case C, and Min-Min and Recursive performed poorly for all three cases. GA, BU, and MCT differed only by their initial subtask to machine assignments, they all used an identical procedure for phase 2 to convert subtasks to their degraded versions to meet the time and the energy constraints. BU used a fitness function to assign all the subtasks to machines, while GA used the initial static mapping and in addition a fitness function for all the unmapped subtasks.

Table 3: The execution times of the heuristics averaged over 100 scenarios.

heuristic	average execution times (seconds)
Recursive	0.1
Bottoms Up	0.38
Genetic Algorithm	0.49
Min-Min	0.84
MCT	0.56

MCT assigned subtasks to machines that gave the minimum completion time. It was seen that MCT assigned more subtasks to fast machines as compared to GA and BU, which had a better load distribution, and hence had a smaller final makespan as seen in Figure 3. This also resulted in a larger number of degraded version subtasks for MCT in Cases A and B. For Case C, which had only one fast machine, MCT 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, MCT performed comparably to GA.

Similar to MCT, the Recursive Bisection heuristic also used minimum completion time to map subtasks to machines and hence mapped a majority of the

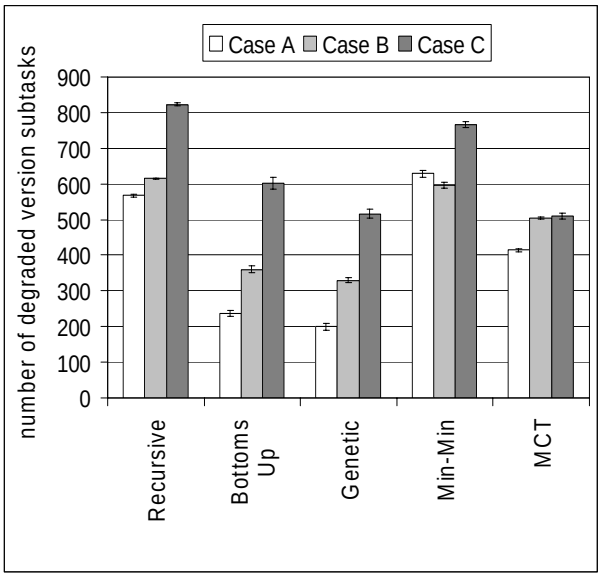

Figure 2: The simulation results for number of subtasks that used degraded versions for the five different heuristics across the three different *ad hoc* grid scenarios.

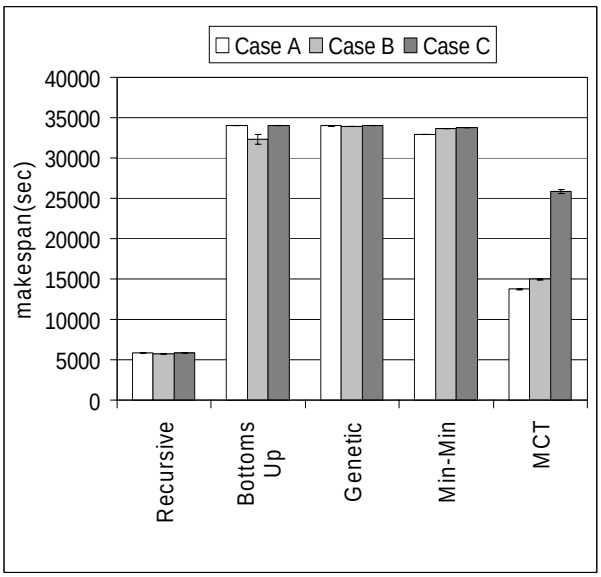

Figure 3: The simulation results for makespan in seconds for the five different heuristics across the three different *ad hoc* grid scenarios.

degraded version subtasks in decreasing order of average execution time across all machines irrespective of whether the subtask was actually mapped to a fast machine or a slow machine. Almost every time, a subtask mapped to a fast machine was converted to its secondary version. This resulted in Recursive Bisection having a very small final makespan (as seen in Figure 3) but a very large number of degraded versions.

Min-Min considered subtasks for conversion in the order they were mapped depending upon a time factor and an energy factor, which in turn depended upon the number of subtasks assigned to a particular machine. This procedure was not found to work well, as can be seen from Figure 2.

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. For all the cases considered, the GA, on average produced the best mappings. The results of this work can be used in the development of *ad hoc* grids.

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 pseudocode.

Acknowledgment: The authors thank Jay Smith for his valuable comments.

References

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

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

[3] R. Armstrong, D. Hensgen, and T. Kidd, "The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions," *7th IEEE Heterogeneous Computing Workshop* (*HCW 1998*), Mar. 1998, pp. 79-87.

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

[5] I. Banicescu and V. Velusamy, "Performance of scheduling scientific applications with adaptive weighted factoring," 10th IEEE Heterogeneous Computing Workshop (HCW 2001), in *Proceedings of the 15th International Parallel and Distributed Processing Symposium* (*IPDPS 2001*), paper HCW 06, Apr. 2001.

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

[7] T. D. Braun, H. J. Siegel, N. Beck, L. Boloni, R. F. Freund, D. Hensgen, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and Bin Yao, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *Journal of Parallel and Distributed Computing*, Vol. 61, No. 6, June 2001, pp. 810-837.

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

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

[10] D. Fernandez-Baca, "Allocating modules to processors in a distributed system," *IEEE Transaction on Software Engineering*, Vol. SE-15, No. 11, Nov. 1989, pp. 1427–1436.

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

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

[13] M. A. Iverson, F. Ozguner, and G. J. Follen, "Parallelizing existing applications in a distributed heterogeneous environment," in *Proceedings of 1995 Heterogeneous Computing Workshop (HCW '95),* April 1995, pp. 93-100.

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

[15] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," *Journal of Parallel and Distributed Computing*, Vol. 59, No. 2, Nov. 1999, pp. 107-121.

[16] D. Marinescu, G. Marinescu, Y. Ji, L. Boloni, and 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), in *Proceedings of the 22nd International Performance, Computing, and Communications Conference (IPCCC)*, Apr. 2003.

[17] Z. Michalewicz and D. B. Fogel, *How to Solve It: Modern Heuristics*, New York, NY, Springer-Verlag, 2000.

[18] S. Shivle, R. Castain, H. J. Siegel, A. A. Maciejewski, T. Banka, K. Chindam, S. Dussinger, P. Pichumani, P. Satyasekaran, W. Saylor, D. Sendek, J. Sousa, J. Sridharan, P. Sugavanam, and J. Velazco, "Static Mapping of Subtasks in a Heterogeneous Ad Hoc Grid Environment," 13th IEEE Heterogeneous Computing Workshop (HCW 2004), in *Proceedings of the 18th International Parallel and Distributed Processing Symposium* (*IPDPS 2004*), Apr. 2004.

[19] M. Srinivas and L. M. Patnaik, "Genetic algorithms: A survey," *IEEE Computer*, Vol. 27, No. 6, June 1994, pp. 17-26.

[20] L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski, "Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach," *Journal of Parallel and Distributed Computing*, Vol. 47, No. 1, Nov. 25, 1997, pp. 8-22.

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