Dynamic Mapping in Energy Constrained Heterogeneous Computing Systems

Jong-Kook Kim[1], H. J. Siegel[2,3], Anthony A. Maciejewski[2], and Rudolf Eigenmann[1]

[1]Purdue University
Electrical and Computer Engineering School
West Lafayette, IN 47907-1285, USA
jongkook@ieee.org
eigenman@ecn.purdue.edu

Colorado State University
[2]Electrical and Computer Engineering Dept.
[3]Computer Science Dept.
Fort Collins, CO 80523-1373, USA
{hj, aam}@colostate.edu

Abstract

An ad hoc *grid is a wireless heterogeneous computing environment without a fixed infrastructure. The wireless devices have different capabilities, have limited battery capacity, support dynamic voltage scaling, and are expected to be used for eight hours at a time and then recharged. To maximize the performance of the system, it is essential to assign resources to tasks (match) and order the execution of tasks on each resource (schedule) in a manner that exploits the heterogeneity of the resources and tasks while considering the energy constraints of the devices. In the single-hop* ad hoc *grid heterogeneous environment considered in this study, tasks arrive unpredictably, are independent (i.e., no precedent constraints for tasks), and have priorities and deadlines. The problem is to map (match and schedule) tasks onto devices such that the number of highest priority tasks completed by their deadlines during eight hours is maximized while efficiently utilizing the overall system energy. A model for dynamically mapping tasks onto wireless devices is introduced. Seven dynamic mapping heuristics for this environment are designed and compared to each other and to a mathematical bound.*

1. Introduction

An *ad hoc* grid is a heterogeneous computing (HC) environment consisting of mobile battery-powered computing devices that communicate using wireless connections. *Ad hoc* grid environments enable users to communicate and share computational load and results with other users in the system to coordinate efforts to accomplish a mission. Examples of applications of *ad hoc* grids include: wildfire fighting, disaster management, and military situations [25]. HC is the coordinated use of various resources with different capabilities to satisfy the requirements of varying task mixtures. When the resources are wireless and mobile, the limited battery capacity becomes a constraint and power or energy management becomes a critical issue. The heterogeneity of the resources and tasks in an HC system is exploited to maximize the performance or the cost-effectiveness of the system (e.g., [6, 11, 15, 24]). An important research problem is how to assign resources to the tasks (match) and to order the tasks for execution on the resources (schedule) to maximize some performance criterion of an HC system. This procedure is called mapping. The power management aspect further complicates the problem.

There are two different types of mapping: static and dynamic. Static mapping is performed when the applications are mapped in an off-line planning phase, e.g., planning the schedule for a set of production jobs. Dynamic mapping is performed when the applications are mapped in an on-line fashion, e.g., when tasks arrive at unpredictable intervals and are mapped as they arrive (the workload is not known *a priori*). In both cases, the mapping problem has been shown, in general, to be NP-complete (e.g., [9, 12, 18]). Thus, the development of heuristic techniques to find near-optimal solutions for the problem is an active area of research (e.g., [4, 5, 6, 7, 11, 13, 14, 20, 23, 27, 33, 37]).

In this research, the *dynamic mapping* of tasks onto machines is studied. Simulation is used for the evaluation and comparison of the dynamic heuristics developed in this research. As described in [23], dynamic mapping heuristics can be grouped into two categories, immediate and batch mode. Each time a mapping is performed (mapping event), immediate mode heuristics only consider the new task for mapping, whereas batch mode considers a subset of tasks for mapping, thus having more information about the task mixture before mapping the tasks. In this paper, we attempted both approaches.

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

The power management is accomplished by using dynamic voltage scaling (DVS) [32]. DVS is based on exploiting the relationship between the CPU supply voltage of a device and the power usage (e.g., Crusoe [8] and ARM7D [2]). The relationship between power and energy is that energy consumed is power multiplied by the amount of time that power is used. The relationship of power to voltage is a strictly increasing convex function, represented by a polynomial of at least second degree [17]. Most processors that support DVS use discrete levels. The DVS technique allows the reduction of a CPU's energy usage (through CPU voltage (clock frequency) reduction) at the expense of increasing the task execution time.

In the environment for this research, the devices are wireless and can communicate with each other (in a single-hop *ad hoc* network). The batteries for these devices are assumed to be recharged after a certain amount of time (e.g., recharged after an eight hour shift or an eight hour work day). Using a device, a user can execute a program (task) to obtain results, receive data, and send data. For the efficient use of the overall system energy available, it may be best for certain tasks to be executed on a remote, rather than the local, device. An RMS makes this decision of locating a "suitable" device. A device performing a computation may receive input data from other devices or external sources. The resulting output data will be sent back to the requester device. Tasks can have different priority levels (i.e., high, medium, or low) and a deadline. The primary goal of this research is to complete as many high priority tasks by their deadlines as possible during a given interval of time (i.e., eight hours). The secondary performance goal is to maximize the sum of the weighted priorities of medium and low priority tasks completed by their deadlines during that interval of time, building on the FISC measure in [19].

This research introduces a model for dynamically mapping tasks onto wireless devices while managing power using the DVS method. Seven dynamic resource allocation heuristics for this environment are designed and compared. Mathematical bounds on the performance of the heuristics are derived.

The next section discusses the heterogeneous *ad hoc* environment followed by a summary of the literature related to this work. In Section 4, the heuristics studied in this research are presented. Section 5 describes the simulation setup. The results are examined in Section 6, and the last section gives a brief summary of this research.

2. Energy Constrained Environment

The *ad hoc* grid environment is controlled by an RMS. The RMS performs matching, scheduling, and power management to maximize the goal stated earlier. In this environment, the wireless devices have limited battery capacity (energy). The users are allowed one battery for the operation of a given device for an interval of time. The batteries are recharged after eight hours. All devices employ DVS for power management but use different discrete voltage levels.

We make the simplifying assumption that the RMS is located on a dedicated machine that has unlimited power and that the devices are within transmission range of the RMS and each other. The users send task requests to the RMS. Once a task request is received, the RMS locates a "suitable" device and sends a task execution command (Figure 1). If input data is required, the data is communicated directly to the executing device from the source. A source could be other wireless devices or outside sources (e.g., from a weather station). The result of the task execution (e.g., a wind direction estimate) is sent back to the task requester device, if the task was not executed on that device. The tasks discussed here have a priority level (e.g., high, medium, or low and a deadline. If the task cannot complete by its deadline, it has no value.

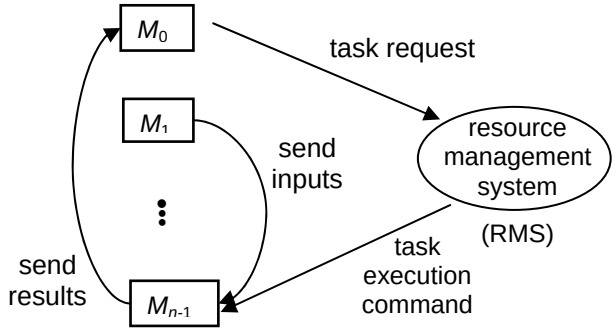

Figure 1: The *ad hoc* grid heterogeneous computing environment considered in this study.

The communication of inputs and results is assumed to be done directly from device to device (i.e., single-hop *ad hoc* network) using the IEEE 802.11b standard, which is a popular wireless standard. In this research, only one task receives or sends data at any instant in time (i.e., the communication link is saved for one task for the duration of its communication). This communication scheme is desirable when a certain quality of service must be met for the tasks. For example, if task *a* arrives at time *t* and the deadline is at $t + d$, the communication of inputs to this task, the execution of the task, and the communication of the results must be done in time *d*. If other communications are allowed while task *a* is still communicating, then the communication time of task *a* is no longer guaranteed. As more and more communications are allowed, the finish time of the communication for task *a* will be delayed.

In this environment, it is assumed that the types of devices that may connect to the system are known. In

addition, there is pre-determined set of tasks that a user can request. However, it is not known *a priori* exactly which tasks will be requested and when they will be requested. In an example military scenario, there are pre-determined types of wireless devices allowed to connect to the military system. In this environment, there is a set of tasks that may be requested for execution (e.g., target determination and weather prediction). A requested task is executed on a "suitable" device and the information is sent back to the task requester. Because the devices and the tasks are known, the task execution times on those devices are known to the RMS (this assumption is typically made when studying mapping heuristics for heterogeneous computing environments (e.g., [21, 34, 35])) and are determined by running the tasks on the devices. It is assumed that all devices are equipped with all programs required and only input data is needed to execute a task and send back results. Thus, the time to communicate the task request to the RMS and to send a task execution command to the device is assumed to be negligible.

3. Related Work

There has been much research on power constrained (power-aware) resource management in uni-processors (e.g., [3, 16, 26, 36]). The research in [3] presents a static scheduling solution of periodic tasks on a processor assuming the worst-case scenario, a dynamic reclaiming algorithm for tasks that complete before their worst-case scenario, and an adaptive speed adjustment mechanism to anticipate the probable early completion of future task executions. A power minimizing approach for variable-voltage systems is developed in [16], where tasks are periodic and independent. The method described in [26] assumes a dynamic preemptive environment where periodic independent tasks arrive and leave a system. In [36], a formal analysis of the minimum energy scheduling problem is provided for a single processor and a model that assumes a task with an arrival time and deadline. The difference between these studies and our research is that our energy constrained *ad hoc* grid environment considers multiple heterogeneous devices and non-periodic independent tasks with priorities and deadlines that need input and/or output communicated. The fact that our environment has heterogeneous multiple devices adds new issues to the resource management problem.

Some research projects have explored a multi-processor environment with static resource management (e.g., [10, 28, 38, 39]). In [10], a genetic algorithm is used to synthesize distributed heterogeneous embedded systems. Using a static schedule derived from a list scheduling scheme, the study in [28] does static and dynamic power management. The work in [38] describes a linear programming method that statically schedules periodic tasks on heterogeneous processing elements. The research in [39] assumes homogeneous processors and frame-based tasks. In static mapping, information of all tasks is known and the heuristic execution time is not a constraint. The difference is that our research explores a dynamic environment where the arrival time of a task is not known prior to its arrival and the task mapping time must be fast. In addition, the devices are heterogeneous in our studies.

The research in [22] statically schedules periodic tasks onto homogeneous processing elements first using the tool in [10] and then slots are created in this static schedule to accommodate aperiodic tasks with hard deadlines. They assume that the minimum interval between two hard aperiodic tasks is larger than the lowest common multiple period of all periodic tasks. Then an online scheduler modifies the system to minimize the response times for aperiodic tasks with soft deadlines. The static schedule is unchanged and the soft aperiodic tasks are run when there is unused time. In our research, all of the devices are heterogeneous and all tasks are aperiodic with hard deadlines. Because all tasks are aperiodic, slots are not created among task periods, i.e., the RMS approaches are quite different. Furthermore our research considers the case where not all tasks with hard deadlines can complete and does not assume a minimum interval between the arrivals of two tasks.

The research in [31] is similar to our work in that it tries to send tasks to another device to be computed. It uses a distributed economic-based subcontracting protocol to determine which device to use. The goal of the devices in [31] is to find a suitable device that can execute tasks to save energy. A cost is associated with devices that are willing to execute a task for other devices. The device that wants to move one of its tasks to another device bargains with those willing devices. The underlying model of our work differs from this in that the environment in our research assumes that all devices are capable of DVS and tasks have deadlines and priorities.

The research in [29] and [30] studies static RMSs for minimizing energy consumption for a heterogeneous *ad hoc* grid. The differences are that in our research, the heuristics operate dynamically, each device supports DVS, tasks have priorities, and it is assumed that not all tasks are completed before their hard deadlines.

4. Heuristic Description

4.1. Mapping Event

A dynamic mapping approach is designed to assign resources to new tasks faster than the anticipated average arrival rate of the tasks. Therefore, the heuristics that are developed have a limit on the maximum time each computation of a new mapping can take. A mapping event occurs when a new task arrives. When a task arrives while

a mapping event is in progress, the current mapping event is not disturbed. As soon as the current mapping event is completed, the next mapping event starts and includes any tasks that have arrived. At any mapping event, the first task in each machine's wait queue is not considered for remapping in any of the heuristics. The reason is that while a mapping event occurs the current executing task can finish, leaving the device idle. Therefore, to help ensure that a device will not be idle for the duration of a mapping event, the first task in the queue is not considered for mapping. While it is still possible that a device may become idle, it is highly unlikely for the assumptions in this research (the mean execution time of a task is 60 or 600 seconds while the mean execution time of a mapping event is less than 0.5 seconds). The mappable tasks are defined as the tasks that are waiting to be executed in the machine queue (except the first task) and the new task(s).

4.2. Opportunistic Load Balancing and Fast Greedy

The immediate mode opportunistic load balancing (OLB) heuristic is a common method for scheduling tasks. The heuristic assigns a task to the device that is expected to be the first to be ready to execute a task. This is a simplistic method that ignores the relationship between the execution of a task and the capability of the devices in the *ad hoc* grid. At a mapping event, among the devices that can map the new task without violating its deadline and have enough energy to complete the task, the heuristic selects the device that will be ready (i.e., executes all the tasks already in its queue) first to map the new task. The immediate mode fast greedy (FG) heuristic assigns tasks onto the device that completes them consuming the minimum amount of energy. This is a scheme that ignores other tasks already in the system. At a mapping event, the device that can complete the task by its deadline and executes the task using the minimum amount of energy is selected for mapping.

The following are same for both approaches. All communications are scheduled as early as possible. If there are previous communications scheduled, then current communications are inserted in the gaps between the communications already scheduled if possible, or else they are put at the end of the communication scheduling queue. If all devices cannot complete the task by its deadline, the task is deleted from the system. The energy consumed status is updated at every mapping event. The scheme of running the high priority tasks using the fastest speed level and the slowest speed level for the rest of the tasks was the best.

4.3. Switching Algorithm

The immediate mode Switching Algorithm heuristic builds on the switching algorithm in [23]. The basic idea behind this heuristic is to first try to map tasks onto their "best" machine according to some metric. But, when the load on the system becomes unbalanced, the strategy is changed to balance the load. When the load becomes balanced then the mapping method is changed back to the original method. For this heuristic, a load balance ratio is used to determine whether the system is load balanced.

In this study, two different load balance ratios are calculated. One is for the high priority tasks and other is for the medium and low priority tasks. The high load balance ratio is the ratio of the earliest device availability time over all the devices in the suite to the latest device availability time. For this ratio, the device availability times are determined using the last high priority task in each queue. If there are no high priority tasks in a device queue, then the device available time is the completion time of the task that is running if it is the only task on the device. If there are other tasks on the device, then the device available time is the completion time of the first waiting task. The low load balance ratio is same as the high load balance ratio except that it is calculated with all tasks. For both load balance ratios, a common high threshold and low threshold are established by experimentation (high threshold > low threshold).

Initially, the system maps new tasks onto their minimum energy consumption device using the slowest speed level. If the task that arrived is a high priority task and there are no devices that can complete the high priority task by its deadline, then the speed level of the devices is increased starting from device 0 using the method described below to test if there are devices that can complete the high priority task with a level increase. When increasing a device's speed level, the total number of speed levels of a device is taken into consideration. For example, assume there is a device 1 that has sixteen speed levels and another device 2 that has four speed levels. If device 1 increased its speed levels at least four times, only then device 2 is considered for speed level increase. Only the speed level for the device selected for mapping is increased. Once the speed level of a device is increased to a faster level, the level will not be decreased to a slower level. This is to guarantee that tasks mapped earlier complete by their deadline. At any mapping event, the speed level is changed at most two times. This is to avoid increasing the speed level to accommodate the current task while not leaving enough energy for future use.

The Switching Algorithm heuristic can be summarized by the following procedure. The total energy consumed is equal to the total CPU energy used plus the energy used for communication.

(1) Determine the priority level of the new task.
(2) Calculate the high (or low) load balance ratio.
(3) If the high (or low) load balance ratio > high threshold, then current method is to use the minimum *energy consumption* device to map the new task.

If the high (or low) load balance ratio < low threshold, then current method is to use the minimum *completion* time device to map the new task.

If low threshold ≤ high (or low) load balance ratio ≤ high threshold, then current method is the one used at the previous mapping event to map the new task.

(4) Initialize "iteration" to the number of speed level changes on the device where the speed level was changed the most.

(5) If the task is a medium or low priority task, assuming that it will be mapped at the end of a device queue, determine all devices that can complete the task by its deadline

if the task cannot be completed on any device, it is deleted from the system

else, select a device using the current method, map the task to this device, and all communications are scheduled using the method in Subsection 4.2.

(6) If the task is a high priority task, assuming that it will be mapped after the last high priority task in a device queue, determine all devices that can complete the task by its deadline.

do until a device is selected for mapping or iteration is increased twice.

if the task cannot be completed on any device, increase the speed level

for each device, change the increase one speed level if the (maximum number of speed levels among all devices)/(total number of levels on the device) ≤ iteration and test if the device can complete the task.

iteration = iteration + 1

else, select a device using the current method, map the task to this device

if the task cannot be completed on any device, return all device's speed level to the level before this task arrived and drop the task.

else, return all unselected devices' speed level to the level before this task arrived.

(7) If there is enough energy on a device to execute tasks at the highest speed level and transmit data for the rest of the remaining time (until the end of the eight hour period), then the speed level for that device is increased to the highest speed level. Check all devices.

(8) Update device availability and energy consumed status.

4.4. Min-Min

The batch mode Min-Min heuristic builds on the Min-Min (greedy) concept in [18]. The Min-Min type heuristic performed very well in previous studies of different environments (e.g., [7, 23]). The Min-Min finds the "best" device for all tasks that are considered and then among these task/device pairs it selects the "best" pair to map first. To determine which device or which task/device pair is the best, a fitness value is used. The fitness value of a task on a given device for this study is (a) the energy consumed for high priority tasks, and (b) the energy consumed multiplied by the weighted priority divided by the execution time of the task for medium and low priority tasks. The energy consumed is equal to the energy used by the CPU plus the energy used for communication. This method also starts the simulation by using the slowest speed level of devices to map tasks.

The Min-Min procedure starts at a mapping event and it is assumed that none of the mappable tasks are mapped, i.e., they are not in any device queue.

(1) All high priority tasks are considered first then the other tasks are considered.

(2) All high priority tasks in the mappable task list are checked to see if they can be completed by their deadline.

(3) If there are some tasks that cannot be completed on any device then the speed level is increased or the task is dropped according to the method used in Subsection 4.3.

(4) For each high priority task in the mappable task list, find the device that gives the task its minimum fitness value (the first "Min") among the devices that can complete the task by its deadline using the current speed level and ignoring other tasks in the mappable task list.

(5) Among all the task/device pairs found from above, find the pair that gives the minimum fitness value (the second "Min"), map the task to the device and remove the task from the mappable task list.

Input or results communication is scheduled using the method in Subsection 4.2.

(6) Update the device availability and energy consumed status.

(7) Do steps (2) to (6) until all high priority tasks are mapped, and then do the same for medium and low priority tasks except the speed level is not increased.

(8) If there is enough energy on a device to execute tasks at the highest speed level and transmit data for the rest of the remaining time (until the end of the eight hour period), then the speed level for that device is increased to the highest speed level. Check all devices.

4.5. Sufferage

The Sufferage heuristic builds on the sufferage concept in [23]. The Sufferage heuristic applies the same fitness value calculation used in the Min-Min heuristic (Subsection 4.4) but when deciding which task to map, the task that "suffers" most if not mapped to its "first choice machine" is selected.

The Sufferage procedure starts at a mapping event. When the mapping event begins, it is assumed that none of the mappable tasks are mapped, i.e., they are not in any device queue.

(1) All high priority tasks are considered first then the other tasks are considered.
(2) All high priority tasks in the mappable task list are checked if they can be completed by their deadline.
(3) If there are some tasks that cannot be completed on any device then the speed level is increased or the task is dropped according to the method used in Subsection 4.3.
(4) For each task in the mappable task list, find the device that gives the task its minimum fitness value among the devices that can complete the task by its deadline using the current speed level, ignoring other tasks in the mappable task list.
(5) If there is contention among any of the high priority tasks, select the task that will suffer the most (the task with the largest difference of fitness value between the best and the second best devices) to map onto the device selected.
    else, map all the high priority tasks.
    all communications are scheduled using the method in Subsection 4.2.
(6) Remove the above task(s) from the mappable task list.
(7) Update the device availability and energy consumed status.
(8) Repeat steps (2) to (7) until all high priority tasks are mapped, and do the same for the medium or low priority tasks except the speed level is not increased.
(9) If there is enough energy on a device to execute tasks at the highest speed level and transmit data for the rest of the remaining time (until the end of the eight hour period), then the speed level for that device is increased to the highest speed level. Check all devices.

4.6. Originator and Random

The immediate mode originator heuristic executes the task on the device that originated the task. This heuristic is run to compare to the performance of heuristics that utilizes other devices in the system. The immediate mode random heuristic maps the new task on a randomly selected device when the new task arrives. This heuristic is run as a pseudo lower bound on the performance and to compare to the performance of other heuristics. The following is for both heuristics. The method in Subsection 4.2 is used for communication scheduling. If the selected device cannot complete the task by its deadline or there is not enough energy to complete the task, the task is deleted from the system. The energy consumed status is updated at every mapping event. The scheme of always running the high priority tasks using the fastest speed level and the slowest speed level for the rest of the tasks was the best.

4.7. Upper Bound

Two upper bound methods are presented in this section. Each time the environment is simulated, the overall upper bound (UB) is determined by selecting the tighter bound of the two methods.

The first upper bound (UB1) uses the arrival time of tasks, priority of tasks, the deadline of the tasks, and the time interval between the arrivals of tasks. The bound ignores the communication and the energy consumed. The tasks that have arrived before or at the mapping event are called selectable tasks. At any mapping event, only the selectable tasks are considered for the calculation of the upper bound. Let ET(i, j) be the execution time of task i on device j and let Q_i be equal to the priority weighting of task i divided by the minimum ET(i, j) over all machines.

The scheme starts by initializing all tasks' remaining ET values, rET(i, j), to the minimum ET(i, j) over all devices. The UB1 follows the procedure described below.

(1) At a mapping event, determine the total aggregate computation time (TACT) until the next task arrives. That is, TACT = time interval between arrival times of the new task and the next task multiplied by the number of machines.
(2) Selectable tasks with rET(i, j) > 0 are put in a task list.
(3) Sort high priority tasks in the task list using minimum ET values and then the medium and low priority tasks with Q_i
(4) If there are high priority tasks in the task list, select the high priority task a that has the minimum ET value
    else, select the medium/low priority task a with the highest Q_a from the task list.
(5) If TACT ≤ rET(a, j)
      if the selected task is high priority,
        subtract TACT from rET(a, j)
      if the selected task is medium or low priority
        add (Q_a × TACT) to the secondary metric
        subtract TACT from rET(a, j)
        done (i.e., TACT = 0)
    if TACT > rET(i, j)
      if the selected task is high priority
      subtract rET(a, j) from TACT (this becomes the new TACT), rET(a, j) = 0
      if the selected task is medium or low priority
        add (Q_a × TACT) to the secondary metric
        subtract rET(a, j) from TACT (this becomes the new TACT), rET(a, j) = 0
(6) Repeat steps (4) and (5) until TACT is equal to 0 or there are no selectable tasks with rET(a, j) > 0.
(7) Repeat steps (1) to (6) until the end of the simulation.

The second upper bound (UB2) uses the energy consumed information of tasks. The total energy available is the sum of all devices' maximum energy available. The energy consumed is equal to the energy used by the CPU plus the energy used for communication. The UB2 starts by determining the minimum energy consumed over all devices for each task. Then the high priority tasks are

ordered in the task list using minimum energy consumed and then the medium and low priority tasks are ordered using the minimum energy consumed divided by the weighted priority Using this order, the number of tasks completed is computed by adding the energy consumed by the tasks until the sum exceeds the total energy available.

While two methods were attempted, UB1 was always tighter than UB2 despite the fact that, in general, UB1 is an unreachable loose bound for this environment.

5. Simulation Setup

Ten types of wireless computing devices and 50 task types are used in the simulated system. Because the devices and the tasks are known, the execution times (ET) of tasks on these different devices are known. In each simulation of a system, eight devices are picked with equal probability. The arrival of tasks is simulated by mean inter-task arrival times using a (memoryless) Poisson distribution. The system is simulated for 480 minutes (i.e., eight hour work time), with eight bursty periods of ten minutes that do not overlap with each other. The bursty periods have faster arrival rates (mean is twice as fast as the rate of the normal period).

A ten by 50 ET matrix of the 50 types of tasks on ten types of devices taking heterogeneity into consideration is generated using the gamma distribution method described in [1], with COV of 0.9 for task heterogeneity and COV of 0.6 for device heterogeneity. When a task is determined to arrive, one of the 50 task types is selected with equal probability. A trial is defined as one such simulation of the HC system (one ten by 50 ET matrix). For each mean inter-task arrival time, 50 trials are run for all heuristics.

Each task is assigned a priority level of high, medium, or low, with equal likelihood. The priority levels of medium and low are given a weighting of four and one. This weighting is to calculate the performance of the value of medium and low priority tasks completed by their deadlines (secondary goal) if the number of high priority tasks completed by their deadlines (primary goal) is the same for some heuristics.

For each device, the maximum battery capacity, the maximum CPU energy consumption rate, and the number of discrete levels for DVS are given. The discrete levels for DVS correspond to the speed at which the CPU is run and defined as speed levels. The environment assumes the IEEE 802.11b standard for wireless communication. It is assumed that the data communication and the task computation/execution can be done simultaneously. Based on two types of wireless devices (a laptop and a handheld), the energy consumption rates are determined. These two devices can be selected with equal probability. The maximum CPU energy consumption rates are determined using a uniform distribution with a range of 0.1 to 0.3 for laptops or 0.01 to 0.03 for handheld devices. The reason for the two ranges is that the CPU energy consumption rate of a laptop is about ten times higher than that of a handheld device (based on sample devices from the Dell website). Based on sample communication adapters (e.g., Linksys) for the two types of devices, the transmission energy consumption rate is 0.6 (about three times the CPU energy consumption rate of a laptop) or 0.2 (about one third of transmission energy consumption rate of a laptop) for the first range or the second range, respectively. The reception energy consumption rate and the idle energy consumption rate are assumed to be 65% and 25% of the transmission power consumption rate respectively. For the simulation study, the maximum battery capacity (energy) of device j, $BC(j)$, is set to the maximum CPU energy consumption rate plus the transmission energy consumption rate, multiplied by the maximum operation time. The maximum operation time is determined using a uniform distribution with a range of 1 to 2 hrs. This means that if the CPU is used at the maximum speed level and the device is always transmitting then the battery capacity is only enough to operate the device for 1 to 2 hrs.

To simplify DVS, this research assumes that each voltage level of a processor corresponds to a clock speed level for the processor. Each device can have 2, 4, 8, or 16 discrete speed levels with equal probability. After the number of levels is decided, the relative speed of each level is determined. The lowest speed level of a device is assumed to be one third of the maximum speed level (e.g., if the maximum speed level is 1.2GHz, then the lowest speed level will be 400MHz). We make the simplifying assumption that task execution time varies linearly with the discrete speed level. It is assumed that the voltage switching is done dynamically and that the overhead associated with the switching is negligible ($20\mu \sim 150\mu$ seconds). The power consumption as a function of speed (voltage) levels is assumed to be a quadratic function. For the example with four speed levels, assume that the maximum energy consumption rate is 0.16. Using a simple equation of maximum energy consumption rate = $\alpha \times$ (relative speed of a speed level to the maximum speed level)^2, α is 0.16. The relative speed of the slowest speed level is 1/3 of the maximum speed level, next will be 5/9 and 7/9 of the maximum speed level (linear). Using these fractions, the energy consumption rates for each speed level are calculated. In this example, the energy consumption rates would be $0.16 \times 1/9$, $0.16 \times 25/81$, $0.16 \times 49/81$, and 0.16 from the slowest speed level to the fastest (maximum) speed level respectively. When the device is idle, the energy consumption rate is assumed to be 1/12 of the maximum energy consumption rate.

The eight devices are assumed to transmit and receive at the speed of 1Mbps. When tasks need to communicate input or output, it is assumed that only one communication is allowed at a time. If multiple tasks need

input data at this moment in time, only one task at a time may receive its input data. For simulation purposes, the size of the input data was calculated using 1K bits as the mean and a COV of 0.7 with the maximum size of 1M bits. The size of the result (output) was calculated using 10K bits as the mean and a COV of 0.7 with the maximum size being 10M bits. A task may receive input from other devices and from one outside source (e.g., a weather station for forecast reports). The maximum total number of inputs a task may need would be nine. The average number of input sources was 2.5.

In a real system, the deadline of a task may be set by the user that requested the task or the system operator. This research assumes that when the task arrives, the deadline of the task is given. For our simulation studies, the deadline of task *i* was equal to its the arrival time plus the overall mean execution time of all tasks plus two times the median execution time of task *i* on all devices plus the communication time (input and result) plus communication wait time (= the mean number of input receptions multiplied by seven multiplied by the mean input communication time plus seven multiplied by the mean result communication time).

6. Results

The simulation results for the different mean execution times and mean inter-task arrival times are shown. Figures 2(a) and 2(b) show the performance of the heuristics as a function of the mean task arrival rates when the mean task execution is 60 seconds. The results show that the Switching Algorithm, Min-Min, and Sufferage heuristics are the best and they performed comparably. The Switching Algorithm, Min-Min, and Sufferage heuristics explicitly consider the high priority tasks first. Other heuristics run the high priority tasks using the fastest speed level, giving the high priority tasks a low probability of getting dropped. The average running times, in seconds per mapping event, of random, originator, OLB, Fast Greedy, Switching Algorithm, Min-Min, and Sufferage are 0.00001, 0.00001, 0.00004, 0.00005, 0.0015, 0.28, and 0.34, respectively. As the mean task arrival rates increase, the number of tasks in the system increases and the percentage of high priority tasks completed decreases. The average number of tasks per trial was 3373, 4185, and 5688 for mean inter-task arrival time of 10, 8, and 6 seconds.

Figures 2(c) and 2(d) show the results when the mean task execution time is increased to 600 seconds. As expected, the performance degraded from the results shown in Figures 2(a) and 2(b). Tasks are more likely to be dropped because of the longer mean execution time. The results show that the Min-Min and Sufferage heuristics are the best and they performed comparably. As the mean task arrival rate increases, the number of tasks in the system also increases and the percentage of high priority tasks completed decreases. As it gets more difficult to complete high priority tasks (as there are more tasks in the system), the batch mode heuristics Min-Min and Sufferage perform better than the rest of the heuristics. While remapping, the batch mode heuristics consider all mappable tasks in the system and the order in which the tasks are mapped can be different from the previous mapping event. Therefore the tasks can be assigned to another machine that is better suited or they can be rescheduled. The Switching Algorithm only considers the new task that arrived and once the task is mapped, it is not moved to another device nor rescheduled. It also can only increase the speed level for one device per mapping event.

7. Summary

An *ad hoc* grid heterogeneous computing environment was modeled and simulated. Seven dynamic heuristics were designed, developed, and tested using the HC environment. The environment includes randomly arriving tasks with priorities and a deadline, and devices with limited battery capacity that use DVS for power management. In this scenario, a resource manager needs to exploit the heterogeneity of the tasks and resources while managing the energy. The primary goal of this study was to complete as many high priority tasks as possible under the constraint of available system energy. A mathematical upper bound was derived.

Min-Min and Sufferage were the best heuristic and they performed comparably. However they required significantly more time than the other heuristics. In cases where the mean task execution times are short the Switching Algorithm may be preferable because it is very fast and can perform comparably to the two best heuristics.

Acknowledgements: The authors thank Sameer Shivle, Prasanna Sugavanam, and T. N. Vijaykumar for their valuable comments.

References

[1] 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).

[2] ARM processor, accessed Mar. 2004, http://www.arm.com.

[3] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez, "Power-aware scheduling for periodic real-time tasks," *IEEE Trans. on Computers*, Vol. 53, No. 5, May 2004, pp. 584-600.

[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), *15th Int'l 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), *15th Int'l 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 Int'l Conf. on Parallel and Distributed Processing Techniques and Applications* (*PDPTA'01*), 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," *J. of Parallel and Distributed Computing*, Vol. 61, No. 6, June 2001, pp. 810-837.

[8] Crusoe Processor, accessed Mar. 2004, http://www.transmeta.com.

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

[10] R. P. Dick and N. K. Jha, "MOCSYN: Multiobjective core-based single-chip system synthesis," *Design Automation & Test in Europe Conference*, Mar. 1999, pp. 263-270.

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

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

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

[14] R. F. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J. D. Lima, F. Mirabile, L. Moore, B. Rust, and H. J. Siegel, "Scheduling resources in multiuser, heterogeneous, computing environments with SmartNet," *7th IEEE Heterogeneous Computing Workshop* (*HCW 1998*), Mar. 1998, pp. 184–199.

[15] R. F. Freund and H. J. Siegel, "Heterogeneous processing," *IEEE Computer*, Vol. 26, No. 6, June 1993, pp. 13–17.

[16] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. B. Srivastava, "Power optimization of variable-voltage core-based systems," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 18, No. 12, Dec. 1999, pp. 1702-1714.

[17] I. Hong, G. Qu, M. Potkonjak, and M. Srivastava, "Synthesis techniques for low-power hard real-time systems on variable voltage processors," *19th IEEE Real-Time Systems Symposium* (*RTSS '98*), Dec. 1998, pp. 95-105.

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

[19] J.-K. Kim, D. A. Hensgen, T. Kidd, H. J. Siegel, D. St. John, C. Irvine, T. Levin, N. W. Porter, V. K. Prasanna, and R. F. Freund, "A flexible multi-dimensional QoS performance measure framework for distributed heterogeneous systems," *Cluster Computing*, accepted in 2004, to appear.

[20] J.-K. Kim, S. Shivle, H. J. Siegel, A. A. Maciejewski, T. D. Braun, M. Schneider, S. Tideman, R. Chitta, R. B. Dilmaghani, R. Joshi, A. Kaul, A. Sharma, S. Sripada, P. Vangari, and S. S. Yellampalli, "Dynamic mapping in a heterogeneous environment with tasks having priorities and multiple deadlines," *12th Heterogeneous Computing Workshop* (*HCW 2003*), *17th Int'l Parallel and Distributed Processing Symposium* (*IPDPS 2003*)), April 2003.

[21] C. Leangsuksun, J. Potter, and S. Scott, "Dynamic task mapping algorithms for a distributed heterogeneous computing environment," *4th IEEE Heterogeneous Computing Workshop* (*HCW '95*), Apr. 1995, pp. 30–34.

[22] J. Luo and N. K. Jha, "Power-conscious joint scheduling of periodic task graphs and aperiodic tasks in distributed real-time embedded systems," *Computer-Aided Design*, Nov. 2000, pp. 357-364.

[23] 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," *J. of Parallel and Distributed Computing*, Vol. 59, No. 2, Nov. 1999, pp. 107-121.

[24] M. Maheswaran, T. D. Braun, and H. J. Siegel, "Heterogeneous distributed computing," *Encyclopedia of Electrical and Electronics Engineering*, *Vol. 8*, J. G. Webster, ed., pp. 679-690, John Wiley, New York, NY, 1999.

[25] 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), *22nd Int'l Performance, Computing, and Communications Conf. (IPCCC)*, Apr. 2003.

[26] P. Mejia-Alvarez, E. Levner, and D. Mosse, "Power-optimized scheduling server for real-time tasks," *IEEE Real-Time and Embedded Technology and Applications Symposium*, Sep. 2002, pp. 239-250.

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

[28] R. Mishra, N. Rastogi, Z. Dakai, D. Mosse, and R. Melhem, "Energy aware scheduling for distributed real-time systems," *Int'l Parallel and Distributed Processing Symposium 2003* (*IPDPS 2003*), Apr. 2003.

[29] 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), *18th Int'l Parallel and Distributed Processing Symposium* (*IPDPS 2004*), Apr. 2004.

[30] S. Shivle, 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, "Mapping of subtasks with multiple versions in a heterogeneous *ad hoc* grid environment," *3rd Int'l Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks* (*HeteroPar '04*), July 2004.

[31] L. Shang, R. P. Dick, and N. K. Jha, "DESP: A distributed economics-based subcontracting protocol for computation distribution in power-aware mobile ad hoc networks," *IEEE Trans. on Mobile Computing*, Vol. 3, No. 1, pp. 33-45.

[32] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced CPU energy," *USENIX Symposium on Operating Systems Design and Implementation*, Nov. 1994, pp. 13-23.

[33] 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.

[34] D. Xu, K. Nahrstedt, and D. Wichadakul, "QoS and contentionaware multi-resource reservation," *Cluster Computing*, Vol. 4, No. 2, Apr.2001, pp. 95-107.

[35] J. Yang, I. Ahmad, and A. Ghafoor, "Estimation of execution times on heterogeneous supercomputer architectures," *Int'l Conf. on Parallel Processing*, Aug. 1993, pp. I-219-I-226.

[36] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," *IEEE Annual Foundations of Computer Science*, 1995, pp. 374-382.

[37] V. Yarmolenko, J. Duato, D. K. Panda, and P. Sadayappan, "Characterization and enhancement of dynamic mapping heuristics for heterogeneous systems," *Int'l Workshop on Parallel Processing*, Aug. 2000, pp. 437-444.

[38] Y. Yu and V. K. Prasanna, "Power-aware resource allocation for independent tasks in heterogeneous real-time systems," *9th Int'l Conf. on Parallel and Distributed Systems*, Dec. 2002, pp.341-348.

[39] D. Zhu, R. Melhem, and B. R. Childers, "Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems," *IEEE Trans. on Parallel and Distributed Systems*, Vol. 14, No. 7, July 2003, pp. 686-700.

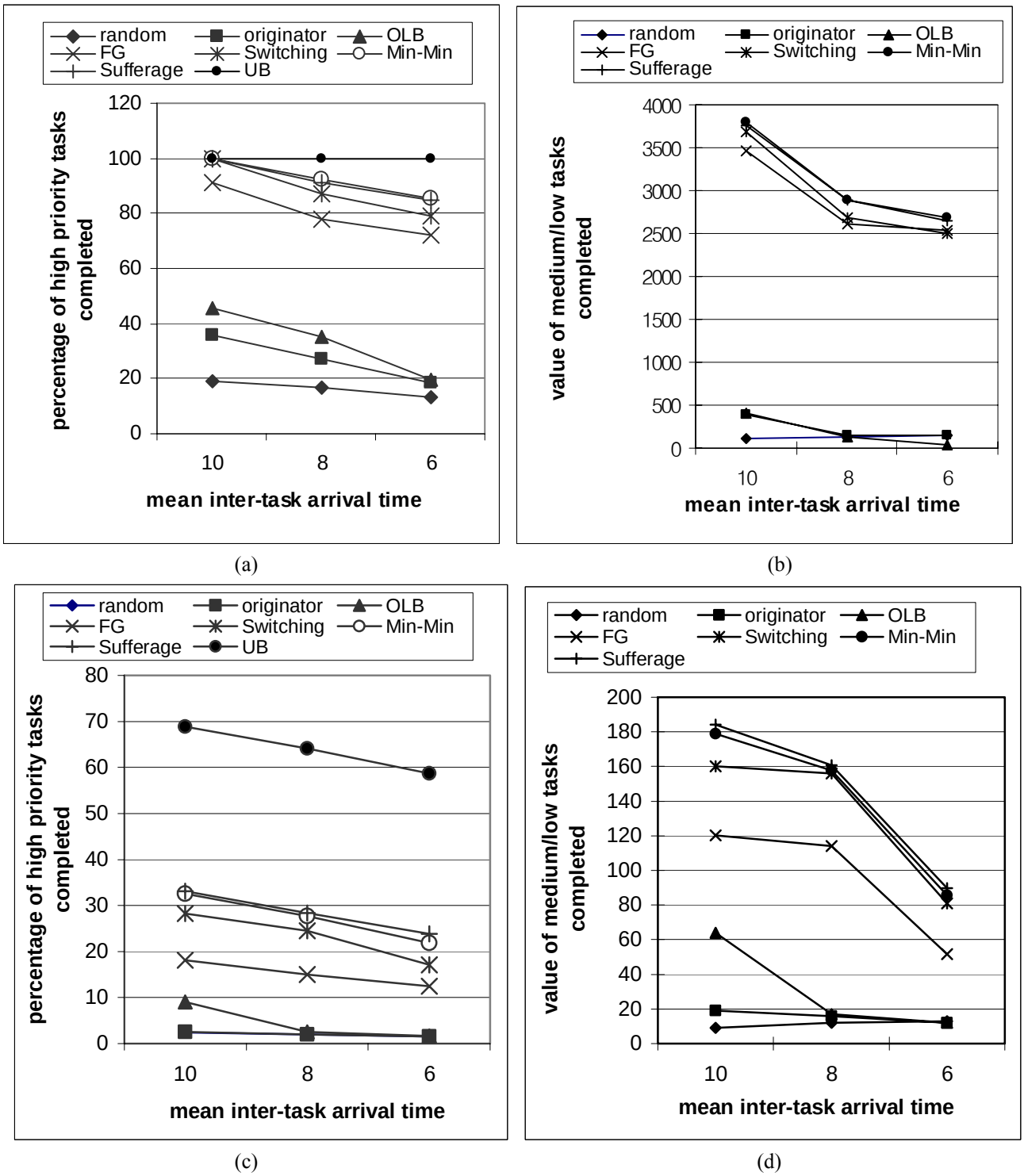

Figure 2: The simulation results using mean inter-task arrival times of 10, 8, and 6 seconds. Graphs (a) and (b) are for amean task execution time of 60 seconds with graphs (c) and (d) for a mean task execution time of 600 seconds. For each mean inter-task arrival time, the value plotted is the average of 50 trials that were run for all heuristics. Note that because energy usage is a constraint and not the optimization criterion, the heuristics will strive to use up all the energy to complete more tasks.