Design of Fast and Efficient Energy-Aware Gradient-Based Scheduling Algorithms for Heterogeneous Embedded Multiprocessor Systems

Lee Kee Goh, Bharadwaj Veeravalli, *Senior Member*, *IEEE*, and
Sivakumar Viswanathan, *Student Member*, *IEEE*

Abstract—In this paper, we present two heuristic energy-aware scheduling algorithms—1) Energy Gradient-based Multiprocessor Scheduling (EGMS) algorithm and 2) Energy Gradient-based Multiprocessor Scheduling with Intratask Voltage scaling (EGMSIV) algorithm—for scheduling task precedence graphs in an embedded multiprocessor system having processing elements with dynamic voltage scaling capabilities. Unlike most energy-aware scheduling algorithms that consider task ordering and voltage scaling separately from task mapping, our algorithms consider them in an integrated way. EGMS uses the concept of energy gradient to select tasks to be mapped onto new processors and voltage levels. EGMSIV extends EGMS by introducing intratask voltage scaling using a Linear Programming (LP) formulation to further reduce the energy consumption. Through rigorous simulations, we compare the performance of our proposed algorithms with a few approaches presented in the literature. The results demonstrate that our algorithms are capable of obtaining energy-efficient schedules using less optimization time. On the average, our algorithms produce schedules which consume 10 percent less energy with more than 47 percent reduction in optimization time when compared to a few approaches presented in the literature. In particular, our algorithms perform better in generating energy-efficient schedules for larger task graphs. Our results show a reduction of up to 57 percent in energy consumption for larger task graphs compared to other approaches.

Index Terms—Energy-aware scheduling, power management, dynamic voltage scaling, energy minimization, heterogeneous multiprocessor scheduling, embedded systems.

1 Introduction

The demand for performance and functionality of embedded systems is increasing over the years. Many applications that require high-speed processing such as multimedia streaming, medical imaging, and high-definition televisions are commonly implemented on embedded systems. These embedded systems are often implemented on platforms comprising of multiple heterogeneous processing units such as general-purpose central processing units (CPU), digital signal processors (DSP), and application-specific integrated circuits (ASIC). Devices such as mobile phones and personal digital assistants (PDA) are also implemented on such multiprocessor systems. These devices handle diverse data types and operate in dynamic application and communication environments.

Energy management is an essential requirement in these devices in order to extend their battery life. Therefore, the scheduler in such devices needs to optimize the performance with respect to energy consumption. The modern day devices utilize processing elements that support dynamic voltage scaling (DVS) [4], [5], [9], [10], [11], [12], [13], [14], [16], [22] to reduce the energy consumption. This technique lowers the supply voltage and operational frequency during runtime at the expense of a longer execution time. By carefully scheduling the tasks to execute at different voltage levels, an optimized schedule with minimum energy consumption can be obtained without compromising the performance.

1.1 Scope of Our Work

In this paper, we attempt to solve the following problem. Given a heterogeneous multiprocessor architecture and a set of tasks, how do we assign the tasks to the heterogeneous multiprocessor system such that the total energy consumption is minimized and all the tasks meet their deadline requirements? The scope of our work does not include the selection of suitable types of processors to be included in a multiprocessor architecture during the design phase. Therefore, factors such as area constraints and monetary costs are not our concern here. Instead, we assume that we are given the heterogeneous multiprocessor architecture and our job is to assign and schedule the given set of tasks to the processors in order to minimize energy consumption and meet real-time constraints. Specifically, our work focuses on scheduling dependent tasks with precedence relationships as represented by a task precedence graph. A task precedence graph is a directed, acyclic graph where nodes represent tasks and edges between the nodes represent the communication

- *L.K. Goh and S. Viswanathan are with the Communication Systems Department, Institute for Infocomm Research, 1 Fusionopolis Way, #21-01 Connexis, Singapore 138632.*
  E-mail: {lkgoh, siva}@i2r.a-star.edu.sg.
- *B. Veeravalli is with the Computer Networks and Distributed Systems (CNDS) Laboratory, Department of Electrical and Computer Engineering, National University of Singapore, 4 Engineering Drive 3, Singapore 117576. E-mail: elebv@nus.edu.sg.*

Manuscript received 24 Jan. 2007; revised 25 Oct. 2007; accepted 5 Mar. 2008; published online 7 Apr. 2008.
Recommended for acceptance by A. Gonzalez.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0022-0107.
Digital Object Identifier no. 10.1109/TPDS.2008.55.

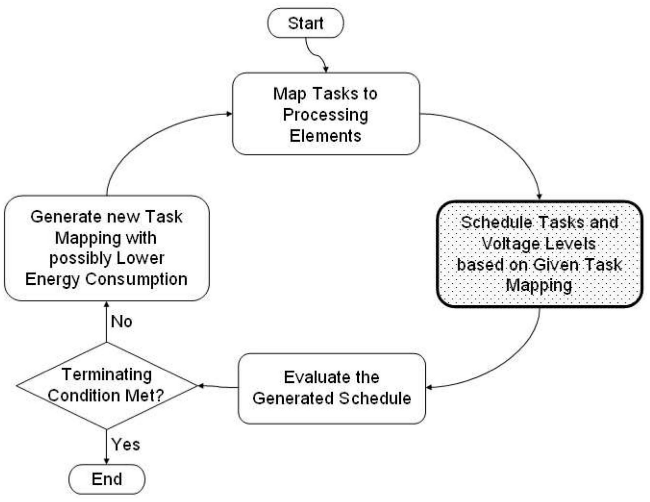

Start
Map Tasks to Processing Elements
Schedule Tasks and Voltage Levels based on Given Task Mapping
Evaluate the Generated Schedule
Terminating Condition Met?
No
Yes
Generate new Task Mapping with possibly Lower Energy Consumption
End

Fig. 1. Typical flow for solving energy-aware scheduling for dependent tasks.

between the tasks. The directions on the edges represent the order in which the tasks must be executed, while the weights on the edges represent the time required to communicate a result from one task to another if they are placed on different processors.

1.2 Related Work

The problem of scheduling dependent tasks with precedence constraints on a heterogeneous multiprocessor system with the objective of minimizing the total energy consumption is NP-complete [25]. The most common way to solve this problem is to divide it into two subproblems. In the first subproblem, we assume that the mapping of tasks to processing elements is known and try to schedule/order the tasks and assign them to the various voltage levels so as to minimize the total energy consumption. We shall define this as the task scheduling and voltage scaling (TSVS) subproblem. The second subproblem is the task mapping (TM) subproblem. Here, we try to iteratively improve the TM based on the feasibility and energy consumption of the schedule that is obtained by solving the TSVS subproblem. Fig. 1 shows the typical flow in solving this energy-aware scheduling problem. The TSVS subproblem is highlighted by the shaded rectangle.

There are many papers [3], [6], [7], [19], [20] that focus solely on the TSVS subproblem. Gruian and Kuchcinski [19] use a list scheduling heuristic with a priority function based on the average energy consumption. Whenever an infeasible schedule is found, the priorities of the tasks are dynamically increased and the tasks are rescheduled. Luo and Jha [20] try to minimize the energy consumption by evenly distributing the slack among the tasks. In [7], Gorjiara et al. propose a fast heuristic by randomly slowing down some of the high-power tasks. Tasks with higher power consumption have higher probabilities of being slowed down. More recently, the authors propose another stochastic-based scheduling algorithm [3], [6] that is faster and more energy efficient. In this approach, they randomly slow down or speed up the tasks based on their energy gradient and execution delays. Tasks with higher energy gradients and lower execution delays are assigned higher probabilities of being slowed down.

There are also some papers [5], [14], [16] that assume that the task ordering is known and focus only on voltage scaling. In [16], Schmitz and Al-Hashimi propose a heuristic that is based on energy gradient and takes into account the power variations among the tasks. Zhang et al. [14] formulate the voltage scaling problem as an integer linear programming (ILP) problem for a fixed task ordering and without considering communication time and energy. Andrei et al. [5] use a mixed integer linear programming (MILP) method to solve the combined problem of voltage scaling and adaptive body biasing assuming a known task ordering. However, the long runtime of the optimal formulation makes it impractical to be used within a TM and scheduling algorithm.

Leung et al. [8] formulate the whole problem of TM, task ordering, and voltage scaling as a mixed integer nonlinear programming (MINLP) problem with continuous voltage levels. However, since their runtime is very long, they propose a divide-and-conquer approach to speed up the optimization process at the expense of losing the optimality of their solution. A part of our earlier work presented in [18] demonstrated the feasibility of using the energy-gradient approach. In [15], Schmitz et al. propose a strategy that also considers TM, task ordering, and voltage scaling. In their strategy, they use a list scheduling heuristic where the priorities of the tasks are generated using a genetic algorithm (GA) and voltage scaling of the tasks is done using [16]. This is then nested inside another GA that is used to determine the optimal mapping of the tasks to the processing elements. While this approach is efficient in generating low-energy schedules, the optimization time is very high due to the nested nature of the GA algorithms and the high complexity of the voltage scaling heuristic that is being used.

1.3 Our Contributions

In this paper, we propose two fast and efficient energy-aware heterogeneous embedded multiprocessor scheduling algorithms for tasks with precedence relationships as represented by a task precedence graph. As shown in Fig. 1, the optimization for TM and TSVS are usually done separately and independently of each other. TSVS assumes a known TM and derives a feasible energy-optimized schedule for that mapping. TM depends only on the results of TSVS and does not consider the process of obtaining the optimized schedule. Our algorithms use an integrated approach in which we optimize TM and TSVS at the same time in order to derive an energy-efficient schedule that satisfies the deadline requirements of the tasks. We start from a schedule generated based on an initial TM and where all the tasks are scheduled at the highest voltage levels. In each optimization step of our algorithms, we try to reduce the energy consumption of the schedule by considering which task should be mapped to another processor, which processor it should be mapped to, and the voltage level at which it should be executed. We use the concept of energy gradient as a priority function to guide the optimization process. Our first algorithm schedules a task precedence graph such that when a task is scheduled onto a processor at a particular voltage level, the entire task must be executed at that voltage level. Our second algorithm extends the first algorithm by using a Linear Programming (LP) formulation for intratask voltage scaling. This allows each task to be scheduled at more than one voltage level on the same processor and, therefore, reduces the energy consumption further. While our algorithms address TM and TSVS in an integrated way, we also describe the modifications needed to use them for TM or TSVS individually. We compare the performance of our algorithms with a few approaches in the literature using both hypothetical and real-life task graphs and show that our algorithms are able to generate schedules that consume less energy while using less optimization time. In particular, our results show that our algorithms are very effective in generating low-energy schedules for larger task graphs. In addition, we also demonstrate that although our algorithms are designed for heterogeneous embedded multiprocessor platforms, they can be used for homogeneous multiprocessors without any loss in performance.

1.4 Organization of Paper

The remainder of this paper is organized as follows: The various models for power, system, and tasks will be introduced in Section 2. We will give detailed descriptions of our proposed algorithms in Section 3 and compare them to existing algorithms in our simulations in Section 4. Lastly, we present the conclusions and discuss some possible extensions in Section 5.

2 Problem Formulation

Our objective is to find a static schedule for the tasks in the task precedence graph on the heterogeneous processors at particular voltage levels such that the total energy consumption is minimized while the task precedence constraints are observed and all the tasks meet their deadline requirements. We shall describe the power, system, and task models in this section.

2.1 Power Model

The dominant source of power dissipation in a digital CMOS circuit is the dynamic power dissipation [24], which is given by (1), where P denotes the dynamic power consumption, C_{ef} the effective load capacitance, V_{dd} the supply voltage, and f the processor frequency. Due to the quadratic relationship between P and V_{dd}, reducing V_{dd} lowers the power consumption effectively. However, reducing V_{dd} results in an increase in the circuit delay. This circuit delay is given by (2), where T_D denotes the circuit delay, k a proportionality constant, V_T the threshold voltage, and α the velocity saturation index. V_T and α are properties of the CMOS circuit and are constant for a particular circuit. Most literatures [9], [11], [13], [15], [16], [22] use the value $\alpha = 2$. The time taken to execute the task is given by (3), where t denotes the execution time of the task and n_c the number of execution cycles required to execute the task. Since $E = P \cdot t$, the total energy dissipation is therefore given by (4):

$$P = C_{ef} \cdot V_{dd}^2 \cdot f, \tag{1}$$

$$T_D = k\frac{V_{dd}}{(V_{dd} - V_T)^{\alpha}}, \tag{2}$$

$$t = \frac{n_c}{f}, \tag{3}$$

$$E = C_{ef} \cdot V_{dd}^2 \cdot n_c. \tag{4}$$

From the above equations, we see that when there is a reduction in the supply voltage, the energy savings increase quadratically. DVS exploits this feature to reduce the energy consumption of the processor at the expense of longer execution times for the tasks.

2.2 System Model

Our system consists of a set of N_p heterogeneous processors, $\{PE_1, PE_2, \ldots, PE_{N_p}\}$, connected to a single bus. Each processor is equipped with DVS functionality. The available discrete voltage levels of PE_j are given by $V(j,k)$, $k = 1, 2, \cdots, N(j)$, where $N(j)$ denotes the total number of discrete voltage levels of PE_j. Without loss of generality, we let $N(1) = N(2) = \ldots = N(N_p) = N_v$ in this paper for simplicity. The power consumption and processor frequency of PE_j at voltage level $V(j,k)$ are given by $P(j,k)$, and $f(j,k)$, respectively. The power consumption of the bus is denoted by P_b. We assume that negligible power is consumed by the processors and the bus when they are idle.

2.3 Task Model

We consider a set of N_t dependent tasks $\{T_1, T_2, \ldots, T_{N_t}\}$ that are related by some precedence constraints as given in the task precedence graph. The amount of time required to execute a task might vary on different processors and also voltage levels. Suppose T_i is executed on PE_j at the voltage level $V(j,k)$, the worst-case execution time needed to execute T_i in this case is given by $t(i,j,k)$ while its energy consumption is given by $e(i,j,k)$. In addition, for a task T_i and its predecessor T_p, if they are executed on different processors, a communication time of $C(p,i)$ is incurred. Let d be the deadline (the latest possible time) by which all the N_t tasks in the task precedence graph must be completed. In our model, when a task is assigned to a processor, we force it to run to completion on the same processor without task migration. In addition, we consider two scenarios depending on whether intratask voltage scaling is used. In the first scenario, we do not allow intratask voltage scaling and the task has to run at the same voltage level until completion. In the second scenario, we allow the task to run at more than one voltage levels during its execution on the same processor. Then, the total energy consumption of the N_t dependent tasks is given by

$$E = P_b \cdot t_c + \sum_{i=1}^{N_t}\sum_{j=1}^{N_p}\sum_{k=1}^{N_v}(x(i,j,k) \cdot e(i,j,k)), \tag{5}$$

where t_c denotes the total duration of time for which the bus is used to transfer data. For the scenario without intratask voltage scaling, we define $x(i,j,k)$ as follows:

$$x(i,j,k) = \begin{cases} 1 & \text{if } T_i \text{ is scheduled on } PE_j \text{ at} \\ & \text{the voltage level } V(j,k), \\ 0 & \text{otherwise.} \end{cases} \quad (6)$$

On the other hand, when intratask voltage scaling is used, $x(i,j,k)$ will denote the fraction of T_i that is scheduled on PE_j at the voltage level $V(j,k)$.

3 Energy-Aware Heterogeneous Embedded Multiprocessor Scheduling Algorithms

In this section, we describe our energy-aware heterogeneous embedded multiprocessor scheduling algorithms. The first algorithm, Energy Gradient-based Multiprocessor Scheduling algorithm (EGMS), is designed to schedule task precedence graphs under deadline constraints. The second algorithm, Energy Gradient-based Multiprocessor Scheduling with Intratask Voltage scaling algorithm (EGMSIV), extends EGMS and utilizes an LP method for intratask voltage scaling. We first define the terms that are used in this section:

- *Makespan*. The makespan of a schedule is the period of time required to completely process all the tasks.
- *Slack*. The slack is the time interval between the time when the execution of a task at a processor is completed and the deadline requirement of that task.

3.1 Design of EGMS

In energy-aware scheduling of task precedence graphs on heterogeneous embedded multiprocessor platforms, there are three main factors that affect the quality of the solution obtained: the mapping of tasks onto processors, the ordering of the tasks, and the selection of the voltage levels of the processors. Most literature [3], [6], [7], [19], [20] considers the ordering of tasks and voltage scaling in an integrated approach for the TSVS subproblem. However, the mapping of tasks to processors is usually considered separately. During TM optimization, the TSVS algorithm that is used has to be invoked repeatedly to obtain the best feasible and energy-efficient schedule for every TM that is generated in the process, regardless of the quality of the TM. In addition, only the final schedule that is generated by the TSVS for each TM is considered during this optimization process. We feel that the TSVS process itself may also be useful in guiding the TM optimization process toward a more energy-efficient schedule at a faster rate. This is one of the main factors that we consider in the design of our algorithms.

Our EGMS algorithm takes into consideration TM, task ordering, and voltage scaling in an integrated manner. A schedule is first generated based on an initial TM. In each optimization step, we try to remap a task to a new processor and/or voltage level such that we reduce the total energy consumption of the schedule as much as possible while decreasing the slack/increasing the makespan as little as possible. In this way, we are optimizing the TM as well as the TSVS at the same time based on the current partially optimized schedule. In doing so, we hope to arrive at an optimized energy-efficient schedule in a shorter time.

Before we describe the EGMS algorithm in detail, we define some notations that we would be using in Fig. 2. The pseudocode of our EGMS algorithm is presented in Algorithm 1. We first generate an initial schedule by assigning tasks to the processors that can complete their execution in the shortest amount of time at the highest voltage level (lines 2-5). We then schedule the tasks using our Critical Path-based Task Ordering (CPTO) algorithm (line 6). In each iteration of the while loop (lines 8-31), we first select a task to be remapped to a new processor and/or a new voltage level such that the total energy consumption is reduced and no deadlines are violated by using the SELECTREMAPTASK() algorithm (line 9). If such a task can be found, the current schedule is updated (lines 10-14). This process continues until no tasks can be remapped without violating the deadlines. When this happens, the energy consumption of the schedule cannot be reduced further. If the current schedule is feasible and has a lower energy consumption than the best schedule obtained so far, we update the best schedule with the current schedule (lines 18-26). However, this best schedule may not be the global optimum schedule. In order to obtain a better solution, we randomly reassign 50 percent of the tasks in the current schedule to other processors at the highest voltage levels and generate a new initial schedule (lines 28-29). The whole process of task remapping is then repeated starting from the new initial mapping until there is no significant improvement in the energy consumption (> 1 percent) of n successive schedules. Here, n is a user-defined parameter that determines the terminating condition of our algorithm. It shall be noted that by reassigning the tasks and applying the algorithm repeatedly, we try to lower the total energy consumption further at the expense of an increase in optimization time.

Notations used in EGMS

$M_p(i)$	:	Processor to which T_i is currently mapped, $1 \leq i \leq N_t$
$M_v(i)$	:	Voltage level to which T_i is currently mapped, $1 \leq i \leq N_t$
e	:	Energy consumption of the current schedule
m_s	:	Makespan of the current schedule
$M_{p_{best}}(i)$	:	Processor to which T_i is mapped in the best schedule generated so far, $1 \leq i \leq N_t$
$M_{v_{best}}(i)$	:	Voltage level to which T_i is mapped in the best schedule generated so far, $1 \leq i \leq N_t$
e_{best}	:	Minimum energy consumption of the best schedule generated so far
$m_{s_{best}}$	:	Makespan of the best schedule generated so far
$numIter$	:	Number of successive iterations without significant improvement
$T_{selected}$	:	Task selected to be re-mapped
$P_{selected}$	:	Processor to which $T_{selected}$ is mapped
$V_{selected}$	:	Voltage level to which $T_{selected}$ is mapped
$e_{selected}$	:	Total energy consumption when $T_{selected}$ is re-mapped
$m_{s_{selected}}$	:	Makespan of schedule when $T_{selected}$ is re-mapped
$pr_{selected}$	:	Priority calculated when $T_{selected}$ is re-mapped to $P_{selected}$ at voltage $V_{selected}$

Fig. 2. Notations used in EGMS algorithm.

Algorithm 1: EGMS()

```
1: $e_{best} \leftarrow \infty$
2: for all $T_i$ do /* Assign tasks to fastest processors */
3:     $M_p(i) \leftarrow$ Fastest-executing processor for $T_i$
4:     $M_v(i) \leftarrow$ Maximum voltage level
5: end for
6: CPTO$(M_p, M_v, e, m_s)$ /* Schedule tasks based on initial processor and voltage mapping */
7: $numIter \leftarrow 0$
8: while $numIter < n$ do
9:     $T_{selected} =$ SELECTREMAPTASK$(P_{selected}, V_{selected}, e_{selected}, m_{s_{selected}})$ /* Find a task to be remapped */
10:    if $T_{selected} \neq -1$ then /* Task to be remapped is found, update current mapping */
11:        $M_p(T_{selected}) \leftarrow P_{selected}$
12:        $M_v(T_{selected}) \leftarrow V_{selected}$
13:        $e \leftarrow e_{selected}$
14:        $m_s \leftarrow m_{s_{selected}}$
15:    else /* No tasks can be remapped without violating deadline */
16:        $numIter++$
17:        if feasible schedule found then
18:            if $e < e_{best}$ then /* Better schedule is found, update the best schedule found so far */
19:                if $\frac{e}{e_{best}} < 0.99$ then /* Improvement $> 1\%$ */
20:                    $numIter \leftarrow 0$
21:                end if
22:                $e_{best} \leftarrow e$
23:                $m_{s_{best}} \leftarrow m_s$
24:                $M_{p_{best}} \leftarrow M_p$
25:                $M_{v_{best}} \leftarrow M_v$
26:            end if
27:        end if
28:        Randomly assign 50 percent of tasks to other processors at maximum voltage level
29:        CPTO$(M_p, M_v, e, m_s)$
30:    end if
31: end while
```

We shall now give the complexity of our EGMS algorithm. Let C_{CPTO} and C_{SEL} be the complexities of CPTO() and SELECTREMAPTASK(), respectively. (We shall show the complexities of these two algorithms later.) Lines 2 to 5 execute in $O(N_t \cdot N_p)$ time. In the while loop, we repeat the steps from lines 9 to 14 until the current schedule cannot be optimized further. Let η be the average number of times these steps are repeated. When we cannot optimize the current schedule further, we execute the steps from lines 15 to 30 to generate a new initial schedule before repeating the steps from lines 9 to 14 again. The steps from lines 15 to 30 are repeated $O(n)$ times. The complexity of the while loop is therefore given by $O(n(\eta \cdot C_{SEL} + N_t + C_{CPTO}))$. Hence, the total complexity of EGMS is $O(N_t \cdot N_p + C_{CPTO} + n(\eta \cdot C_{SEL} + N_t + C_{CPTO}))$. We shall show later that this complexity is dominated by C_{SEL} and so it can be given as $O(n \cdot \eta \cdot C_{SEL})$.

Algorithm 2 shows the CPTO() algorithm that we use to generate a schedule. Based on the given processor and voltage level mapping, we first replace all the communication edges between two tasks that are mapped on different processors using tasks where the execution time is equal to the communication time (line 3). For each task, we calculate the length of the critical path from that task and initialize its start and end times (lines 4-8). We then schedule the tasks based on their critical paths (lines 9-26). Tasks with longer critical paths have higher priorities and are scheduled first. The makespan and energy consumption of the schedule are also calculated in the process. The complexity of CPTO() is given as follows: Let N_a be the total number of computational and communication tasks and N_{succ} be the average number of successors of a task. Let us assume that the tasks are already in topological order. For lines 4-8, the lengths of all the critical paths can be calculated in $O(N_a \cdot N_{succ})$ time. We use a Fibonacci heap to implement the priority queue Q. Therefore, insertion into Q (line 9) is $O(1)$ and removal from Q (line 11) is $O(logN_a)$ amortized time. Line 24 requires $O(N_{succ})$ time for execution. The while loop is executed N_a times. Thus, the total complexity of CPTO() is

$$O(N_a \cdot N_{succ} + N_a \cdot (logN_a + N_{succ})) = O(N_a \cdot (logN_a + N_{succ})).$$

Algorithm 2: CPTO(M_p, M_v, e, m_s)

```
1: $e \leftarrow 0$
2: $m_s \leftarrow 0$
3: Replace the communication between any 2 tasks that are scheduled on different processors with a task, where the execution time is the communication time
4: for all $T_j$ do /* Both computational and communication tasks */
5:     Calculate $l_{cp}(j)$ /* Length of critical path starting from $T_j$ */
6:     $t_{start}(j) \leftarrow 0$ /* Initialize start time of $T_j$ to 0 */
7:     $t_{end}(j) \leftarrow 0$ /* Initialize end time of $T_j$ to 0 */
8: end for
9: Insert tasks with no incoming edges into priority queue $Q$, where tasks are sorted in decreasing values of $l_{cp}$
10: while $Q$ is not empty do
11:     Remove $T_j$ from front of $Q$. /* Get task with largest critical path length */
12:     if $T_j$ is communication task then
13:         $e \leftarrow e +$ Communication energy
14:         $t_{start}(j) \leftarrow$ Earliest time communication bus is free
15:         $t_{end}(j) \leftarrow t_{start}(j) +$ Communication time
16:     else /* $T_j$ is computational task */
17:         $e \leftarrow e + e(j, M_p(j), M_v(j))$
18:         $t_{start}(j) \leftarrow$ Earliest time $PE_{M_p(j)}$ is free
19:         $t_{end}(j) \leftarrow t_{start}(j) + t(j, M_p(j), M_v(j))$
20:     end if
21:     if $t_{end}(j) > m_s$ then
22:         $m_s \leftarrow t_{end}(j)$
23:     end if
24:     Remove $T_j$ and all outgoing edges from task graph
25:     Insert tasks with no incoming edges into priority queue $Q$
26: end while
```

Algorithm 3 shows the SELECTREMAPTASK() algorithm that is used to select the best task to be remapped, as well as the processor and voltage level it should be remapped to. In this algorithm, we consider the cases when T_i is remapped to processor P_j at voltage level $V(j,k)$ for all i, j and k. For each $< T_i, P_j, V(j,k) >$ triplet, we invoke CPTO() to obtain the energy consumption e' and makespan m'_s of the new schedule generated by the remapping (lines 7-11). We then calculate the priority p_r if this new schedule is feasible and has a lower energy consumption than the current schedule:

Algorithm 3:
SELECTREMAPTASK($P_{selected}, V_{selected}, e_{selected}, m_{s_{selected}}$)

```
1:  $T_{selected} \leftarrow -1$
2:  $p_{r_{selected}} \leftarrow -\infty$
3:  for $i \leftarrow 1$ to $N_t$ do
4:      for $j \leftarrow 1$ to $N_p$ do
5:          Select the highest voltage $l$ such that the total
            energy consumption is reduced
6:          for $k \leftarrow l$ to 1 do /* No need to consider
            voltage levels higher than $l$ */
7:              $curProc \leftarrow M_p(i)$, $M_p(i) \leftarrow j$ /* Re-map $T_i$ to
                $PE_j$ at $V(j,k)$ */
8:              $curVoltage \leftarrow M_v(i)$, $M_v(i) \leftarrow k$
9:              CPTO($M_p, M_v, e', m'_s$) /* Obtain schedule
                based on new mapping */
10:             $M_p(i) \leftarrow curProc$ /* Revert back to
                original mapping */
11:             $M_v(i) \leftarrow curVoltage$
12:             if $m'_s \leq d$ then /* Schedule is feasible */
13:                 Calculate priority $p_r$ using (7)
14:                 if $p_r > p_{r_{selected}}$ then /* Highest priority
                    found so far */
15:                     $p_{r_{selected}} \leftarrow p_r$
16:                     $T_{selected} \leftarrow i$
17:                     $P_{selected} \leftarrow j$
18:                     $V_{selected} \leftarrow k$
19:                     $e_{selected} \leftarrow e'$
20:                     $m_{s_{selected}} \leftarrow m'_s$
21:                 end if
22:             else /* Deadline violated, no need to
                consider lower voltage levels */
23:                 break
24:             end if
25:         end for
26:     end for
27: end for
28: return $T_{selected}$
```

$$p_r = \begin{cases} \frac{e-e'}{m'_s - m_s} & \text{if } m'_s > m_s, \\ w \times (e-e') & \text{if } m'_s \leq m_s. \end{cases} \quad (7)$$

Here, we consider two cases. In the first case, the new schedule has a lower energy consumption but a longer makespan. Here, we use the concept of energy gradient to calculate the priority so that schedules that give the largest reduction in energy consumption with the least increase in makespan will be assigned higher priorities. Most of the schedules will be in this case. In the second case, the new schedule has both a lower energy consumption and a shorter makespan. In this case, we assign higher priorities to schedules that result in larger reduction of energy consumption. We use an arbitrary large constant w in the calculation of the priority so as to assign higher priorities to these schedules compared to the schedules in the first case. This is because schedules that reduce both the energy consumption and the makespan are much more preferred than those in the first case.

It shall be noted that we do not need to invoke CPTO() and calculate the priorities for all $< T_i, P_j, V(j,k) >$ triplets. For a task that is remapped to a particular processor, we can obtain the highest voltage level l such that the total energy consumption is lower than the current consumption for all voltages $\leq l$ (line 5). This step takes $O(logN_v)$ time. Therefore, we do not need to consider the schedules for voltages greater than l in the innermost k-loop (lines 6-25). In addition, whenever an infeasible schedule is generated for a particular voltage level, we do not need to consider the lower voltage levels as well (lines 22-23). As the schedule becomes more optimized, the number of feasible voltage levels that can be mapped to also decreases. Although the overall worst-case complexity of SELECTREMAPTASK() is $O(N_t \cdot N_p \cdot (logN_v + N_v \cdot C_{CPTO})) = O(N_t \cdot N_p \cdot N_v \cdot C_{CPTO})$, the average-case complexity is usually much smaller. This shall be shown from the results of our simulation experiments in Section 4.

3.2 Design of EGMSIV

We extend our EGMS algorithm to scenarios where intratask voltage scaling shall be used. In intratask voltage scaling, a fraction of a task can be executed at a particular voltage level while the rest of the task is executed at another voltage level on the same processor. We assume that there is negligible overhead involved when a processor changes its voltage level.

In order to introduce intratask voltage scaling in our algorithm, we need to address the following issue. Given the processor mapping and ordering of each task, how should we assign the voltage levels to the tasks so that the total energy consumption is minimized and all tasks meet their deadline requirements?

This can be formulated into an LP problem in the following way. Let $M_p(i)$ be the given processor mapping of each task T_i. For any two tasks that are connected by an edge in the task precedence graph and are scheduled on different processors, a communication task C_j with communication time $t_c(j)$ is required to transfer data between the two processors. Let N_c be the total number of such communication tasks to be scheduled on the bus. Let $y(i, M_p(i), k)$ denote the fraction of T_i that executes on the given processor $M_p(i)$ at the voltage level $V(M_p(i), k)$. Let $t_s^{T_i}$ and $t_e^{T_i}$ denote the start and end times of the execution of T_i. In addition, we denote the start and end times of the execution of C_j using $t_s^{C_j}$ and $t_e^{C_j}$, respectively. The voltage scaling problem is then formulated as shown in Fig. 3.

This LP formulation of the voltage scaling problem can be solved optimally in polynomial time using currently available LP solvers if the formulation is feasible. The solution to this problem will give the optimized energy consumption, the fraction of tasks to be scheduled on each

LP Formulation for Voltage Scaling Problem

Minimize

$$E = \sum_{i=1}^{N_t} \sum_{k=1}^{N_v} (y(i, M_p(i), k) \cdot e(i, M_p(i), k)) + P_b \cdot \sum_{j=1}^{N_c} (t_e^{C_j} - t_s^{C_j})$$

Subjected to

1) Fraction of T_i assigned to $V(M_p(i), k) \leq 1$):
$0 \leq y(i, M_p(i), k) \leq 1$,
for $i = 1, 2, \cdots, N_t$; $k = 1, 2, \cdots, N_v$
2) Fractions of T_i must add up to exactly one:
$\sum_{k=1}^{N_v} y(i, M_p(i), k) = 1$,
for $i = 1, 2, \cdots, N_t$
3) End time of T_i = Start time of T_i + Execution time of T_i:
$t_e^{T_i} = t_s^{T_i} + \sum_{k=1}^{N_v} (y(i, M_p(i), k) \cdot t(i, M_p(i), k)) \leq d$,
for $i = 1, 2, \cdots, N_t$
4) End time of C_j = Start time of C_j + Communication time of C_j:
$t_e^{C_j} = t_s^{C_j} + t_c(j) \leq d$,
for $j = 1, 2, \cdots, N_c$
5) Tasks scheduled on the same processor must not overlap:
$t_s^{T_i} \geq t_e^{T_a}$,
for $i = 1, 2, \cdots, N_t$, where T_a and T_i are scheduled on the same processor and T_i is the next task to be executed after T_a.
6) Communications scheduled on the bus must not overlap:
$t_s^{C_j} \geq t_e^{C_b}$,
for $j = 1, 2, \cdots, N_c$, where C_b and C_j are scheduled on the bus and C_j is the next edge to be scheduled after C_b.
7) Precedence constraints are observed:
$t_s^{T_i} \geq t_e^{C_p}$, $t_s^{C_p} \geq t_e^{T_q}$,
for $i = 1, 2, \cdots, N_t$, where T_q and T_i are scheduled on different processors, C_p is the communication task between them, and T_i is the immediate successor of T_q.

Fig. 3. Voltage scaling using LP formulation.

voltage level, the start and end times of the tasks on the processors that they are mapped to, as well as the start and end times of the communication tasks on the bus. However, it should be noted that the optimality of this solution applies only to the given processor mapping and ordering of the tasks.

With this LP formulation of the voltage scaling problem, we can extend our EGMS algorithm to introduce intratask voltage scaling in the following way. When the current schedule is feasible and cannot be optimized further (line 17 of EGMS()), the processor mapping and task ordering of the current schedule is based on the assumption that no intratask voltage scaling is used. In order to obtain the energy consumption for the case with intratask voltage scaling, we apply our LP formulation based on the same processor mapping and ordering of the tasks as the current schedule. This additional step of applying the LP formulation is inserted between lines 17 and 18 of the EGMS algorithm. Since the current schedule is feasible, the LP formulation is also feasible. As such, the final schedule will reflect the minimized energy consumption obtainable using our algorithm for scheduling the tasks with intratask voltage scaling. In addition, our EGMS algorithm is employed repeatedly until there is no significant improvement in the solution after n successive iterations. Hence, one needs to apply this formulation $O(n)$ times to avoid the solution being trapped in local minima.

3.3 Individual TM and TSVS Algorithms

While EGMS and EGMSIV consider TM, task ordering, and voltage scheduling in an integrated way, they can be modified to address the TM as well as the TSVS subproblems separately. Due to space constraints, we shall only give a brief description of the changes required to modify them:

1. *EGMS for TM Only (EGMS-TM).* To use EGMS for TM only, we shall replace our CPTO() algorithm (lines 6 and 29 of EGMS(), line 9 of SELECTREMAPTASK()) with any TSVS algorithms that the user desires. In addition, since the voltage scaling is done by the TSVS algorithms, we do not need to consider the voltage level mapping. We shall therefore remove lines 4, 12, and 25 from EGMS(), as well as lines 5, 8, 11, and 18 from SELECTREMAPTASK(). In addition, in SELECTREMAPTASK(), we do not need to consider the innermost k-loop. Instead, the steps inside the k-loop should be moved to the outer j-loop.
2. *EGMS/EGMSIV for TSVS Only (EGMS-TSVS/EGMSIV-TSVS).* To use EGMS/EGMS-IV for TSVS only, we shall assume that the TM is already given. Since no TM is required, we can remove lines 3, 11, and 24 from EGMS() as well as lines 5, 7, 10, and 17 from SELECTREMAPTASK(). We also remove the j-loop in SELECTREMAPTASK(). In addition, since fast TSVS algorithms are often required, we shall apply the steps in the k-loop (lines 8-21) for the next lower voltage level only.

4 SIMULATION RESULTS

In this section, we describe the simulation study performed to evaluate the performance of our algorithms in terms of energy minimization as well as the optimization time.

4.1 Energy Optimization without Task Mapping

We consider the TSVS subproblem where the mapping of tasks to processors is assumed to be given. We shall compare our EGMS-TSVS and EGMSIV-TSVS algorithms with ASG-VTS [3], [6]. ASG-VTS is an ultrafast algorithm that produces energy-efficient schedules without intratask voltage scaling. An online version of the DVS tool that uses ASG-VTS is available in [2]. We implemented the algorithms using C++ in a Cygwin environment on a Pentium-IV/3.2-GHz/2-Gbyte RAM PC running Windows XP. We randomly generated 40 task graphs comprising a maximum of 100 tasks using TGFF [23]. We use the lp_solve [1] library for solving the LP formulation of the voltage scaling problem in our EGMSIV-TSVS algorithm. The time required to execute the tasks on the processors at the highest voltage level was defined in an expected time to compute (ETC) matrix which was generated using the method described in [21]. We generated the ETC matrix for high task as well as high machine heterogeneity ($V_{task} = 0.5, V_{mach} = 0.5$) condition. We assumed that each processor

TABLE 1
Energy Optimization Using ASG-VTS, EGMS-TSVS, and EGMSIV-TSVS for TSVS Based on a Given Mapping of Tasks to Processors

Task Graph	No. of tasks /edges /processors	Energy Savings (%)			Optimization Time (secs)		
		ASG-VTS	EGMS-TSVS(1)	EGMSIV-TSVS(1)	ASG-VTS	EGMS-TSVS(1)	EGMSIV-TSVS(1)
1	10/9/2	29.43	28.26	35.57	0.011	0.011	0.010
2	24/27/3	34.76	38.50	44.55	0.012	0.016	0.034
3	40/46/4	32.88	41.71	52.14	0.012	0.081	0.113
4	19/21/2	30.50	35.09	40.52	0.011	0.012	0.016
5	39/46/4	34.85	43.51	47.00	0.012	0.063	0.090
6	19/23/2	29.12	34.81	38.73	0.012	0.015	0.016
7	19/30/2	31.97	40.35	44.08	0.012	0.013	0.025
8	17/19/2	21.70	31.14	33.52	0.013	0.012	0.015
9	15/16/2	34.25	38.61	39.65	0.011	0.012	0.014
10	15/18/2	32.19	37.08	40.15	0.011	0.011	0.014
11	29/34/3	35.34	47.21	50.54	0.012	0.034	0.056
12	35/41/4	36.10	46.38	51.68	0.015	0.056	0.078
13	36/42/4	36.17	47.17	50.49	0.016	0.057	0.090
14	24/27/3	38.70	48.28	50.06	0.010	0.022	0.034
15	39/47/4	36.89	42.75	52.07	0.015	0.063	0.088
16	30/36/3	33.36	40.01	44.47	0.015	0.034	0.059
17	28/33/3	23.44	36.20	38.78	0.012	0.022	0.037
18	11/11/2	33.33	35.05	40.39	0.011	0.011	0.012
19	13/14/2	30.03	31.57	35.26	0.012	0.010	0.012
20	18/20/2	32.13	34.34	39.23	0.012	0.013	0.016
21	69/82/7	41.71	52.59	59.43	0.019	0.487	0.588
22	99/119/10	40.58	56.24	61.44	0.029	1.562	1.772
23	84/102/9	39.17	54.48	61.00	0.028	0.878	1.032
24	79/96/8	38.18	42.49	50.23	0.019	0.516	0.653
25	48/58/6	48.04	54.59	61.21	0.016	0.200	0.247
26	9/8/2	32.28	34.01	42.59	0.011	0.010	0.011
27	16/19/2	32.96	37.19	40.40	0.011	0.010	0.014
28	19/22/2	28.93	32.36	33.86	0.013	0.012	0.019
29	24/28/3	35.47	43.64	48.24	0.011	0.025	0.041
30	29/35/3	31.90	42.79	45.96	0.013	0.031	0.053
31	35/43/4	35.75	44.07	51.89	0.017	0.053	0.078
32	39/47/4	38.19	49.14	52.89	0.013	0.093	0.116
33	44/52/5	38.41	47.52	51.98	0.012	0.097	0.141
34	51/61/6	41.39	53.55	58.04	0.019	0.203	0.253
35	55/65/6	47.14	56.13	63.51	0.019	0.275	0.340
36	12/17/2	38.16	41.77	45.52	0.012	0.010	0.012
37	22/38/3	48.31	57.53	57.96	0.015	0.025	0.047
38	33/35/4	37.16	46.60	55.19	0.012	0.041	0.065
39	43/47/5	36.14	47.65	52.96	0.017	0.097	0.137
40	54/61/6	48.84	59.77	63.62	0.016	0.237	0.297
Average		35.65	43.30	48.17	0.014	0.136	0.169

has four voltage levels at 0.9, 1.7, 2.5, and 3.3 V. The mean task execution time (μ_{task}) was set as 10 and the mean power consumption of each processor at maximum voltage level (μ_{power}) was set as 100. The maximum power ratings for the processors were randomly generated using a gamma distribution with α and β parameters calculated as follows:

$$\alpha = \frac{1}{V_{mach}^2}, \quad (8)$$

$$\beta = \frac{\mu_{power}}{\alpha}. \quad (9)$$

The power ratings of the processors at other voltage levels were calculated using (1). As in most literature [9], [11], [13], [15], [16], [22], we also set the velocity saturation index to be 2.

Table 1 shows the average energy savings and the optimization times achieved by ASG-VTS and our EGMS-TSVS and EGMSIV-TSVS algorithms over five runs of our simulation experiments. We observe that the average optimization time of ASG-VTS is about 10 times faster than both EGMS-TSVS and EGMSIV-TSVS. However, our algorithms perform better than ASG-VTS in terms of energy minimization. On the average, there is an improvement of about 21 percent in terms of the amount of energy saved when we compare EGMS-TSVS to ASG-VTS. Our EGMSIV-TSVS performs even better, obtaining an improvement of about 35 percent compared to ASG-VTS as a result of using intratask voltage scaling.

4.2 Energy Optimization with Task Mapping

For energy optimization with TM, task ordering, and voltage scaling, we compare our EGMS and EGMSIV algorithms to the nested GA approach [15] as well as the ASG-VTS algorithm [3], [6]. In the nested GA approach, a GA-based task scheduling algorithm EE-GLSA is nested within another GA-based TM algorithm EE-GMA in order to obtain the best processor mapping, task ordering, and voltage level mapping. We also compare our algorithms with the ASG-VTS algorithm by inserting it into the EE-GMA algorithm and

TABLE 2
Scheduling Strategies Compared in the Simulation Study for Energy Optimization with TM

Strategy	Task Mapping Algorithm	Task Scheduling & Voltage Scaling Algorithm	Intra-task Voltage Scaling
EE-GMA(ASG-VTS)	EE-GMA	ASG-VTS	No
EE-GMA(EGMS-TSVS)	EE-GMA	EGMS-TSVS	No
EGMS-TM(ASG-VTS)	EGMS-TM	ASG-VTS	No
EGMS	EGMS-TM	EGMS-TSVS	No
Nested GA	EE-GMA	EE-GLSA	Yes
EE-GMA(EGMSIV-TSVS)	EE-GMA	EGMSIV-TSVS	Yes
EGMSIV	EGMS-TM	EGMSIV-TSVS	Yes

our EGMS-TM algorithm for processor mapping optimization. We denote these approaches as EE-GMA(ASG-VTS) and EGMS-TM(ASG-VTS), respectively. The communication delays are uniformly distributed between 1 and 5, while the power consumption of the communication bus is set at 10. We also insert our EGMS-TSVS and EGMSIV-TSVS algorithms into EE-GMA and compare the results. These approaches are denoted as EE-GMA(EGMS-TSVS) and EE-GMA(EGMSIV-TSVS), respectively. A summary of the features of the various approaches used in the simulation is shown in Table 2.

In the nested GA approach [15], we omitted the area penalty in the calculation of the fitness function in EE-GMA, since processor area is not a constraint in our analysis. Hence, the fitness function used in our experiment is given by

$$F_s = E \cdot \left(1 + \frac{\sum_{i=1}^{N_s}(max(0, t_e^i - d))^2}{d^2}\right), \qquad (10)$$

where t_e^i is the end time of the sink task T_i, and N_s is the total number of sink tasks. Here, the sink tasks refer to those tasks in the task precedence graph that have no successors. The inner EE-GLSA algorithm terminates when there is no improved solution (improvement > 1 percent) being produced for 10 successive generations. The outer EE-GMA algorithm terminates when there is no improvement in the solution for 25 successive generations. In addition, just as in [15], the calculation of the energy consumption obtained using the nested GA approach assumes the use of intratask voltage scaling. The EE-GMA algorithm that is used in combination of other TSVS algorithms also terminates when there is no improvement in the solution for 25 successive generations. For our EGMS-TM algorithm, we set $n = 25$ as well.

We use the same set of task graphs as in Section 4.1 and obtain the energy consumption and optimization time required by the various approaches. For our EGMS and EGMSIV algorithms, we consider the cases when n is 1, 100, and 500, respectively. For the purpose of comparison, we normalized the energy consumption and the optimization time obtained using the various methods by those obtained using the nested GA approach. The results are averaged over five simulation runs. However, we are unable to find a feasible schedule for the nested GA approach for task graphs 21-24 and 40. Figs. 4 and 5 show the average normalized energy consumption and optimization time required by the various algorithms for the remaining 35 task graphs. We also show the 95 percent confidence intervals in our figures. The first five algorithms in the figures use intratask voltage scaling, while the last six algorithms do not employ intratask voltage scaling.

For the first five algorithms that use intratask voltage scaling, we observe that when we replace the EE-GLSA algorithm in the nested GA approach with our EGMSIV-TSVS algorithm, we are able to reduce the energy consumption further. Our EGMSIV algorithm performs even better. Compared to the nested GA approach, our EGMSIV algorithm is able to reduce the energy consumption by 7 percent to 10 percent when n increases from 1 to 500. Among the remaining six algorithms that do not consider intratask voltage scaling, EE-GMA(ASG-VTS) consumes the most energy, followed by EGMS-TM(ASG-VTS), EE-GMA(EGMS-TSVS), EGMS(1), EGMS(100), and EGMS(500). We observe that when we individually replace either the EE-GMA or ASG-VTS algorithms in EE-GMS(ASG-VTS) by our EGMS-TM or EGMS-TSVS algorithms, respectively, we are able to reduce the energy consumption further. However, our EGMS algorithm is able

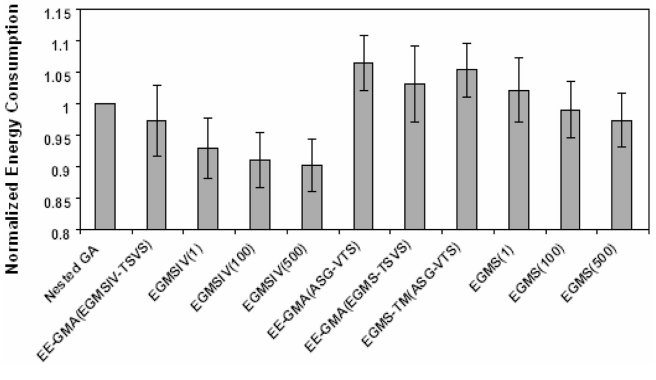


Fig. 4. Average normalized energy consumption by the various algorithms for mapping optimization with 95 percent confidence intervals.

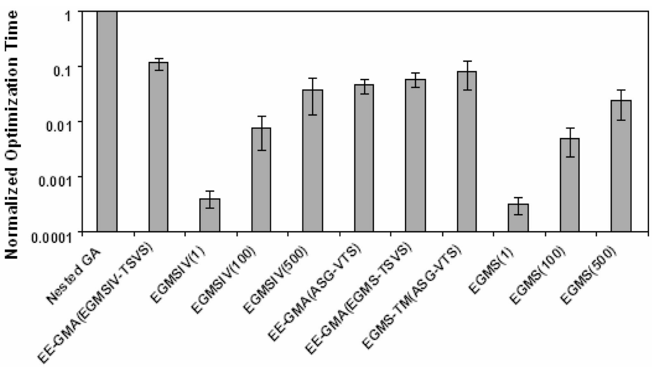


Fig. 5. Average normalized optimization time required by the various algorithms for mapping optimization with 95 percent confidence interval.

TABLE 3
Normalized Energy Consumption Achieved for Mapping Optimization Using Real-Life Applications Used in [17]

Task Graph	Normalized Energy Consumption						
	Without Intra-task Voltage Scaling				With Intra-task Voltage Scaling		
	EE-GMA (ASG-VTS)	EE-GMA (EGMS-TSVS)	EGMS-TM (ASG-VTS)	EGMS(500)	Nested GA	EE-GMA (EGMSIV-TSVS)	EGMSIV(500)
fft1	1.073	0.992	1.052	0.885	1.000	0.900	0.792
fft3	1.207	1.051	1.192	0.989	1.000	1.017	0.957
karp10	1.081	0.946	1.031	0.894	1.000	0.900	0.873
qmf4	1.034	1.043	1.015	1.014	1.000	0.980	0.968
meas	1.029	1.029	1.029	1.029	1.000	0.999	0.999

to obtain an even better performance. Compared to EE-GMA(ASG-VTS), EGMS reduces the energy consumption by 4 percent to 9 percent when n increases from 1 to 500.

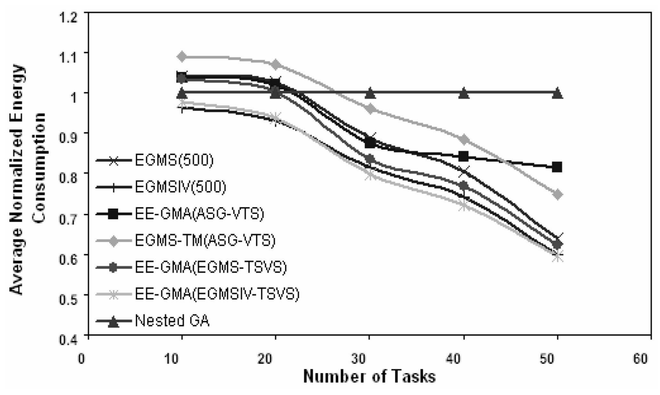
Next, let us look at the optimization time required by the various algorithms to derive a feasible schedule. Nested GA requires between 6 and 19,765 seconds to derive a feasible schedule, while EE-GMA(ASG-VTS) requires between 1 and 77 seconds. EGMS-TM(ASG-VTS) requires between 1 and 52 seconds, while EE-GMA(EGMS-TSVS) and EE-GMA(EGMSIV-TSVS) require up to 328 and 868 seconds for optimization, respectively. The best optimization times are achieved by our EGMS and EGMSIV algorithms. Both EGMS and EGMSIV take about 0.006 to 1.6 seconds, 0.2 to 3.5 seconds, and 1 to 19 seconds for optimization when $n = 1$, $n = 100$, and $n = 500$, respectively. Compared to nested GA and EE-GMA(ASG-VTS), there is a reduction of 96 percent and 47 percent in the optimization time, respectively, even when $n = 500$. From these results, we observe that when we increase the number of iterations, both EGMS and EGMSIV are able to obtain feasible schedules with lower energy consumption at the expense of a longer optimization time. However, this optimization time is still shorter than those required by both nested GA and EE-GMA(ASG-VTS).

In addition to the hypothetical task graphs generated using TGFF [23], we also applied our algorithms to some task graphs corresponding to real-life examples. We repeat the experiment using the set of task graphs used by Bambha et al. [17]. The set of task graphs consists of two differently implemented fast Fourier transforms (fft1, fft3), a Karplus-strong music synthesis algorithm (karp10), a quadrature mirror filter bank (qmf4), and a measurement application (meas). These applications are run on multiprocessor platforms consisting of identical processors. The normalized energy consumption and normalized optimization time of the various algorithms are shown in Tables 3 and 4. Due to space constraints, we shall only show the results for our EGMS and EGMSIV algorithms when $n = 500$. Again, our EGMS and EGMSIV algorithms achieve the best results in terms of energy minimization. In terms of optimization time, EE-GMA(ASG-VTS) achieves the best results for all the task graphs except for qmf4. However, the schedules generated by EE-GMA(ASG-VTS) consume about 11 percent more energy. From the results, we observe that although our algorithms are designed for heterogeneous multiprocessors, they can be used for homogeneous multiprocessors as well.

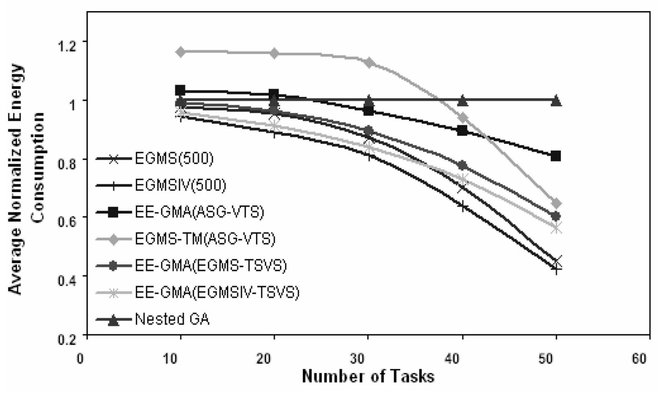
Lastly, we evaluate the performance of our algorithms when the number of tasks and processors increases. We randomly generated task graphs containing between 10 and 50 tasks each and obtained their average normalized energy consumption. The results are shown in Fig. 6 for the cases when three and six processors are used, respectively. From the figures, we observe that when the number of tasks increases, our algorithms perform better. When three processors are used, both EGMSIV(500) and EE-GMA(EGMSIV-TSVS) have the best performance, followed by EE-GMA(EGMS-TSVS), EGMS(500), EE-GMA(ASG-VTS), and nested GA. EGMS-TM(ASG-VTS) does not perform well for smaller task graphs. However, it performs better than EE-GMA(ASG-VTS) when the number of task is large. When we increase the number of processors from 3 to 6, our EGMS and EGMSIV algorithms outperform the other algorithms, especially when the number of tasks is large. From these results, we observe that when the number of tasks and processors increases, the search space becomes exponentially larger, and therefore, the EE-GMA GA used in nested GA and EE-GMA(ASG-VTS) is unable to converge fast enough before the terminating condition is met. This is also the reason why EE-GMA(ASG-VTS) outperforms EGMS-TM(ASG-VTS) for smaller task graphs but not larger ones. However, when EE-GMA is used with our EGMS-TSVS and EGMSIV-TSVS algorithms, it is able to converge to better solutions faster. On the other hand, our EGMS and EGMSIV

TABLE 4
Normalized Optimization Time Required for Mapping Optimization Using Real-Life Applications Used in [17]

Task Graph	Normalized Optimization Time						
	Without Intra-task Voltage Scaling				With Intra-task Voltage Scaling		
	EE-GMA (ASG-VTS)	EE-GMA (EGMS-TSVS)	EGMS-TM (ASG-VTS)	EGMS(500)	Nested GA	EE-GMA (EGMSIV-TSVS)	EGMSIV(500)
fft1	0.00403	0.01502	0.01546	0.00953	1.000	0.02954	0.01099
fft3	0.00025	0.00123	0.00081	0.00077	1.000	0.00111	0.00081
karp10	0.00036	0.00213	0.00144	0.00323	1.000	0.00296	0.00355
qmf4	0.1485	0.01086	0.02495	0.02242	1.000	0.05413	0.02675
meas	0.00396	0.00892	0.00188	0.00978	1.000	001390	0.01527



(a)



(b)

Fig. 6. Average normalized energy consumption as the number of tasks increases. (a) Three processors. (b) Six processors.

algorithms obtain the best performance as they are able to converge to very good solutions by considering TM, task ordering, and voltage scaling in an integrated way.

5 Conclusions and Possible Extensions

In this paper, we have proposed two novel heuristic energy-aware heterogeneous embedded multiprocessor scheduling algorithms for scheduling task precedence graphs. Unlike most scheduling algorithms in the literature that consider TM or TSVS separately, our algorithms consider them in an integrated way to derive a low-energy schedule. In EGMS, we use the concept of energy gradient to obtain an energy-efficient schedule. In each iteration of the algorithm, a task is selected to be remapped onto a new processor and/or voltage level based on the calculated energy gradient so that the new schedule has the largest decrease in energy consumption with the smallest increase in makespan. This algorithm does not consider intratask voltage scaling and is useful for scheduling tasks in situations where intratask voltage scaling cannot be used due to its high overhead. In EGMSIV, we extended our EGMS algorithm to introduce intratask voltage scaling by using an LP formulation for the voltage scaling problem. Although EGMSIV requires a slightly longer time to compute due to the additional step of solving the LP formulation for the voltage scaling problem, this increase in optimization time is not very significant. Both algorithms are applied repeatedly until there is no improvement in the solution for n successive iterations. The larger the value of n, the higher the probability of obtaining a feasible schedule with a lower energy consumption. However, the optimization time also increases as a result. We also describe modifications that can be made to our algorithms so that they can be used for TM (EGMS-TM) or TSVS (EGMS-TSVS/EGMSIV-TSVS) separately.

We compared the performance of our algorithms with the nested GA approach [15] and the ASG-VTS algorithm [3], [6] (nested inside EE-GMA [15] for mapping algorithm) using both hypothetical and real-life task graphs. The simulation results showed that our EGMS and EGMSIV algorithms are capable of obtaining energy-efficient schedules using less optimization time. In particular, we showed that for the case when $n = 500$, our algorithms are able to reduce the average energy consumption by about 9 percent to 10 percent. At the same time, the average optimization time is also reduced by 96 percent and 47 percent, respectively, when compared to the nested GA approach and the ASG-VTS algorithm. We also combined EE-GMA with our EGMS-TSVS/EGMSIV-TSVS, as well as our EGMS-TM with ASG-VTS and compared their performance. Although they are able to obtain better results than nested GA and ASG-VTS, our EGMS and EGMSIV still achieve the best results. We also showed that our EGMS and EGMSIV algorithms are able to reduce the energy consumption of larger task graphs by up to 57 percent and 47 percent when compared to the nested GA approach and the ASG-VTS algorithm, respectively. This shows that our algorithms are much more effective in reducing energy consumption for larger task graphs while still meeting the deadlines. In addition, we also demonstrated that although our algorithms are designed for use on heterogeneous multiprocessor systems, they can also be used for homogeneous multiprocessors without any loss in performance.

The simulations in this paper were based on heterogeneous processors with the same number of voltage levels and tasks with uniform power profiles. However, our proposed algorithms can be easily extended to systems with a mixture of non-DVS and DVS processors with different voltage levels, as well as for tasks with nonuniform power profiles. Also, our algorithms can be modified so that they can be used to schedule task precedence graphs in which each task has a different deadline requirement.

Acknowledgments

The authors would like to acknowledge the support for the project "*Design of Adaptive and Hybrid Energy-&-QoS Aware Heterogeneous Multiprocessor Scheduling Strategies for Embedded Systems*," Grant R-263-000-375-305(052 118 0062) by Agency for Science, Technology and Research (A*STAR) SERC, Singapore under the EHS-II Programme. The authors would like to thank Marcus T. Schmitz, Bashir M. Al-Hashimi, and Petru Eles for clarifying some technical aspects and sharing the relevant data in their earlier approach [15], as well as Bita Gorjiara, Nader Bagherzadeh, and Pai Chou for providing the relevant data in [3] and [6]. The authors would also like to thank the Editor-in-Chief of this journal and the Associate Editor of this paper for their patience and prompt replies to the authors' inquiries during the review process. Lastly, the authors would like to thank Liu Yanhong for his valuable comments on this paper.

References

[1] http://sourceforge.net/projects/lpsolve/, 2008.

[2] http://cecs02.cecs.uci.edu/DVS/, 2008.

[3] B. Gorjiara, N. Bagherzadeh, and P. Chou, "Ultra-Fast and Efficient Algorithm for Energy Optimization by Gradient-Based Stochastic Voltage and Task Scheduling," *ACM Trans. Design Automation of Electronic Systems*, vol. 12, Article 39, no. 4, Sept. 2007.

[4] Y. Yu and V.K. Prasanna, "Energy-Balanced Task Allocation for Collaborative Processing in Wireless Sensor Networks," *Mobile Networks and Applications*, vol. 10, pp. 115-131, 2005.

[5] A. Andrei, M.T. Schmitz, P. Eles, Z. Peng, and B.M. Al-Hashimi, "Overhead-Conscious Voltage Selection for Dynamic and Leakage Energy Reduction of Time-Constrained Systems," *IEE Proc.—Computers and Digital Techniques*, vol. 152, no. 1, pp. 28-38, Jan. 2005.

[6] B. Gorjiara, N. Bagherzadeh, and P. Chou, "An Efficient Voltage Scaling Algorithm for Complex SoCs with Few Number of Voltage Modes," *Proc. Int'l Symp. Low Power Electronics and Design (ISLPED '04)*, pp. 381-386, 2004.

[7] B. Gorjiara, P. Chou, N. Bagherzadeh, M. Reshadi, and D. Jensen, "Fast and Efficient Voltage Scheduling by Evolutionary Slack Distribution," *Proc. Asia and South Pacific Design Automation Conf. (ASP-DAC '04)*, pp. 659-662, Jan. 2004.

[8] L.-F. Leung, C.-Y. Tsui, and W.-H. Ki, "Minimizing Energy Consumption of Multiple-Processors-Core Systems with Simultaneous Task Allocation, Scheduling and Voltage Assignment," *Proc. Asia and South Pacific Design Automation Conf. (ASP-DAC '04)*, pp. 647-652, Jan. 2004.

[9] D. Zhu, R.G. Melhem, and B.R. Childers, "Scheduling with Dynamic Voltage/Speed Adjustment Using Slack Reclamation in Multiprocessor Real-Time Systems," *IEEE Trans. Parallel and Distributed Systems*, vol. 14, no. 7, pp. 686-700, July 2003.

[10] H. Aydin and Q. Yang, "Energy-Aware Partitioning for Multiprocessor Real-Time Systems," *Proc. 17th Int'l Parallel and Distributed Processing Symp. (IPDPS '03)*, Apr. 2003.

[11] R. Mishra, N. Rastogi, D. Zhu, D. Mossé, and R.G. Melhem, "Energy Aware Scheduling for Distributed Real-Time Systems," *Proc. 17th Int'l Parallel and Distributed Processing Symp. (IPDPS' 03)*, Apr. 2003.

[12] Y. Yu and V.K. Prasanna, "Power-Aware Resource Allocation for Independent Tasks in Heterogeneous Real-Time Systems," *Proc. Ninth Int'l Conf. Parallel and Distributed Systems (ICPADS '02)*, pp. 341-348, Dec. 2002.

[13] D. Zhu, N. AbouGhazaleh, D. Mossé, and R.G. Melhem, "Power Aware Scheduling for AND/OR Graphs in Multi-Processor Real-Time Systems," *Proc. 31st Int'l Conf. Parallel Processing (ICPP '02)*, pp. 593-601, Aug. 2002.

[14] Y. Zhang, X. Hu, and D.Z. Chen, "Task Scheduling and Voltage Selection for Energy Minimization," *Proc. 39th Design Automation Conf. (DAC '02)*, pp. 183-188, June 2002.

[15] M.T. Schmitz, B.M. Al-Hashimi, and P. Eles, "Energy-Efficient Mapping and Scheduling for DVS Enabled Distributed Embedded Systems," *Proc. Design, Automation and Test in Europe Conf. and Exposition (DATE '02)*, pp. 514-521, Mar. 2002.

[16] M.T. Schmitz and B.M. Al-Hashimi, "Considering Power Variations of DVS Processing Elements for Energy Minimisation in Distributed Systems," *Proc. Int'l Symp. Systems Synthesis (ISSS '01)*, pp. 250-255, Oct. 2001.

[17] N.K. Bhamba, S.S. Bhattacharyya, J. Teich, and E. Zitzler, "Hybrid Global/Local Search Strategies for Dynamic Voltage Scaling in Embedded Multiprocessors," *Proc. Ninth Int'l Symp. Hardware/Software Codesign (CODES '01)*, pp. 243-248, Apr. 2001.

[18] L.K. Goh, B. Veeravalli, and S. Viswanathan, "An Energy-Aware Gradient-Based Scheduling Heuristic for Heterogeneous Multiprocessor Embedded Systems," *Proc. 14th IEEE Int'l Conf. High Performance Computing (HiPC '07)*, pp. 331-341, 2007.

[19] F. Gruian and K. Kuchcinski, "LEneS: Task Scheduling for Low-Energy Systems Using Variable Supply Voltage Processors," *Proc. Asia and South Pacific Design Automation Conf. (ASP-DAC '01)*, pp. 449-455, Jan. 2001.

[20] J. Luo and N.K. Jha, "Power-Conscious Joint Scheduling of Periodic Task Graphs and Aperiodic Tasks in Distributed Real-time Embedded Systems," *Proc. Int'l Conf. Computer-Aided Design (ICCAD '00)*, pp. 357-364, Nov. 2000.

[21] S. Ali, H.J. Siegel, M. Maheswaran, D.A. Hensgen, and S. Ali, "Task Execution Time Modeling for Heterogeneous Computing Systems," *Proc. Ninth Heterogeneous Computing Workshop (HCW '00)*, pp. 185-199, May 2000.

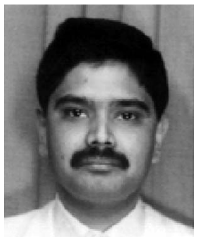
[22] T. Ishihara and H. Yasuura, "Voltage Scheduling Problem for Dynamically Variable Voltage Processors," *Proc. Int'l Symp. Low Power Electronics and Design (ISLPED '98)*, pp. 197-202, Aug. 1998.

[23] R.P. Dick, D.L. Rhodes, and W. Wolf, "TGFF: Task Graphs for Free," *Proc. Sixth Int'l Workshop Hardware/Software Codesign (Codes/CASHE '98)*, pp. 97-101, Mar. 1998.

[24] A.P. Chandrakasan, S. Sheng, and R.W. Brodersen, "Low-Power CMOS Digital Design," *IEEE J. Solid-State Circuits*, vol. 27, no. 4, pp. 473-484, Apr. 1992.

[25] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman, 1979.

Lee Kee Goh received the BEng and MEng degrees from the National University of Singapore (NUS) in 2004 and 2005, respectively. He is currently working toward the PhD degree in the area of energy and QoS-aware heterogeneous multiprocessor scheduling at NUS. He is currently a research officer in the Communication Systems Department, Institute for Infocomm Research, Singapore. His research interests include multiprocessor systems, parallel and distributed systems, and energy-aware scheduling.



Bharadwaj Veeravalli received the BSc degree in physics from Madurai-Kamaraj University, India, in 1987 and the master's degree in electrical and communication engineering and the PhD degree from the Indian Institute of Science (IISc), Bangalore, India, in 1991 and 1994, respectively. He did his postdoctoral research in the Department of Computer Science, Concordia University, Montreal, in 1996. He is currently with the Department of Electrical and Computer Engineering, Communications and Information Engineering (CIE) Division, National University of Singapore, as a tenured associate professor. His mainstream research interests include multiprocessor systems, cluster/grid computing, scheduling in parallel and distributed systems, bioinformatics and computational biology, and multimedia computing. He is one of the earliest researchers in the field of divisible load theory (DLT). He has published more than 105 papers in high-quality international journals and conferences. He had successfully secured several externally funded projects. He has coauthored three research monographs in the areas of parallel and distributed systems, distributed databases (competitive algorithms), and networked multimedia systems, in the years 1996, 2003, and 2005, respectively. He guest edited a special issue on cluster/grid computing for the *International Journal of Computers and Applications (IJCA)* in 2004. He is currently serving on the editorial boards of the *IEEE Transactions on Computers*, the *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, and is an associate editor of the *IJCA*. He had served as a program committee member and as a session chair in several international conferences and had been recently invited to contribute to *Multimedia Encyclopedia* (Kluwer Academic Publishers, 2005). He is a senior member of the IEEE and the IEEE Computer Society.

Sivakumar Viswanathan received the BE degree from the University of Madras, India, in 1989 and the MSc degree from the National University of Singapore (NUS) in 2001. He is currently working toward the PhD degree in the area of high-performance network-based computing at NUS and is the assistant department head in the Communication Systems Department, Institute for Infocomm Research, Singapore. His research interests include multiprocessor systems, cluster/grid computing, and scheduling in parallel and distributed systems. He is a student member of the IEEE and the IEEE Computer Society.