Cluster Comput (2010) 13: 373–383
DOI 10.1007/s10586-009-0119-6

Energy aware DAG scheduling on heterogeneous systems

Sanjeev Baskiyar · Rabab Abdel-Kader

Received: 10 March 2008 / Accepted: 28 December 2009 / Published online: 29 January 2010

Abstract We address the problem of scheduling directed a-cyclic task graph (DAG) on a heterogeneous distributed processor system with the twin objectives of minimizing finish time and energy consumption. Previous scheduling heuristics have assigned DAGs to processors to minimize overall run-time of the application. But applications on embedded systems, such as high performance DSP in image processing, multimedia, and wireless security, need schedules which use low energy too.

We develop a new scheduling algorithm called Energy Aware DAG Scheduling (EADAGS) on heterogeneous processors that can run on discrete operating voltages. Such processors can scale down their voltages and slow down to reduce energy whenever they idle due to task dependencies. EADAGS combines dynamic voltage scaling (DVS) with Decisive Path Scheduling (DPS) to achieve the twin objectives. Using simulations we show average energy consumption reduction over DPS by 40%. Energy savings increased with increasing number of nodes or increasing Communication to Computation Ratios and decreased with increasing parallelism or increasing number of available processors. These results were based on a software simulation study over a large set of randomly generated graphs as well as graphs for real-world problems with various characteristics.

Keywords DAG · Scheduling · Cluster · Energy aware · Makespan

S. Baskiyar (✉)
Department of Computer Science and Software Engineering, Auburn University, Auburn, AL 36849, USA
e-mail: baskiyar@auburn.edu

R. Abdel-Kader
Faculty of Engineering, Suez Canal University, Port Said, Egypt
e-mail: rababfarouk@gmail.com

1 Introduction

In the present work, scheduling on heterogeneous distributed computing systems interconnected by high-speed networks is considered. Such systems are promising for fast processing of computationally intensive applications with diverse computation needs. One of the challenges in heterogeneous computing is to develop scheduling algorithms that assign the tasks of applications to processors [17]. Therefore, researchers have proposed many static, dynamic and even hybrid algorithms to minimize execution time of applications running on a heterogeneous system; we can mention [6, 8, 16, 20, 22, 24]. As many applications require both low finish time and low power consumption, the power consumption is another challenge facing distributed computing [5]. Power consumption is a major issue in many real time distributed embedded systems. Most applications running on a power limited system inherently constrain the finish time. The explosive interest in sensor networks is the result of the development of low-cost, low-power multifunctional sensor devices, such as the Smart Dust Mote [21]. The DAG structure is important as it occurs in many regular and irregular applications in forms of Cholesky factorization, LU decomposition, Gaussian elimination, FFT, Laplace transforms and instruction level parallelism. Such low power schedules can also be useful in multi-hop sensor radio networks.

Traditionally, priority has been on performance, and consequently the supply voltage has been set at the maximum allowable level based on device breakdown potentials to enable fast operation. However, processors do not require the maximum achievable speed at all times. The top power-consumers in a computer system are display (68%), disk (20%), and CPU (12%) [9]. There seems little which can be done to minimize screen power-consumption, beyond em-

ploying a screen-saver and relying on hardware improvements. Disk power consumption is minimized by spinning down the disk when it has been inactive for several seconds. In the future, we may well see ubiquitous computing devices with neither disks nor conventional displays. For such devices, minimizing power consumed by the CPU will be particularly critical if the replacements of disks and displays consume relatively smaller fractions of total power.

The remainder of this paper is organized as follows. In Sect. 2, related work on different scheduling heuristics on both homogenous and heterogeneous systems, power estimation and optimization techniques, and scheduling for low energy consumption has been described. In Sect. 3, we define the problem and terminology used to describe our scheduling algorithm. In Sect. 4, we outline our scheduling algorithm. In Sect. 5, results and analysis of the software simulation which was conducted to validate the algorithm has been given. Finally, conclusions and suggestions for future work have been presented in Sect. 6.

2 Related work

Two main design aspects of scheduling are how to build the scheduling queue and how to choose the optimal processor. Concerning scheduling, list and cluster scheduling are primary techniques to schedule tasks on heterogeneous systems. In list scheduling tasks are ordered in a scheduling queue based on the priority assigned to free tasks. List scheduling algorithms have been shown to have good cost-performance trade-offs. Cluster scheduling involves merging nodes/paths to form clusters that can be scheduled on the same processor to get closer to the objectives of schedule length, no. of processors etc. Several algorithms for static scheduling on heterogeneous multiprocessors systems are available: Dynamic Level Scheduling (DLS), Generalized Dynamic Level Scheduling (GDLS), Best Imaginary Level (BIL), Mapping Heuristics (MH), Heterogonous Earliest Finish Time (HEFT), Task Duplication Scheduling (TDS), Static Task Duplication Scheduling (STDS), Fast Critical Path (FCP), and Fast Load Balancing (FLB). Among the above TDS and STDS employ task duplication to suppress communication whereas others do not. A brief description of these algorithms is available in [23].

2.1 Run-time power reduction

There are two techniques that can reduce power consumption on system level scheduling: Dynamic Power Management (DPM) and Dynamic Voltage Scaling (DVS). DPM dynamically reconfigures an electronic system by reducing number of active components and/or load on such components while providing services. DPM is used in various forms usually in portable devices. However, the complexity of interfacing heterogeneous components has limited designers to simple solutions. An example of a simple policy, mostly applied to laptops and PDA, is a timeout policy, which turns off a component after a fixed inactivity time, under the assumption that it is highly likely that a component remains idle if it has been idle for the timeout time. In DVS, computation and communication, tasks are run at reduced voltages and clock frequencies to fill idle periods and reduce energy dissipation, while providing required performance. The key idea of DVS is to dynamically scale the supply voltage of CPU while meeting total computation time and/or throughput. For example, reducing the supply voltage from 5 V to 3.3 V in some cases has reduced power by 56% [15]. DVS essentially fills the slack times by elongated computation or communication times. There are two types of slack time: Worst Slack Time (WST) and Workload-Variation Slack time (WVST). WST results from low processor utilization. WVST occurs due to execution time variations caused by data-dependent computation. WST can be roughly estimated from the scheduling results before task execution whereas VST can be known only after execution.

2.2 Scheduling to lower energy consumption

For uniprocessor real-time systems many schemes have been proposed to manage energy consumption. Mosse [12] proposed and analyzed several schemes to dynamically adjust processor speed for slack reclamation. They proved that using a compiler to assist the operating system in changing the CPU operating levels can reduce energy consumption. Chandrakasan [3] has shown that for periodic tasks, a few voltage/frequency levels are sufficient to achieve almost the same energy savings as infinite voltage/speed levels. Yang [26] proposed a two-phase scheduling scheme that minimizes energy consumption while meeting timing constraints. By choosing different scheduling options at compile time they achieved 20–40% average power savings.

Zhang, Hu and Chen [27] proposed a two phase process for energy managed scheduling of fixed task graphs on multiprocessors. In the first phase, they use a priority based task ordering and scheduling. Tasks *after* scheduling are modeled as a DAG. In the second phase, the voltage scaling problem is modeled as an integer programming (IP) problem. They show that the IP problem is solvable in polynomial time for the continuous voltage and particular discrete voltage cases. For tasks sets composed of 10–500 tasks and a target of up to eight processors they showed that their framework can slow down 8–98% of cycles. However, they do not consider inter-task communication within the DAG. Also, in addition to precedence they mandate deadlines for individual tasks. However, in this work, we consider *communication between tasks* in a DAG and consider up to 1,000 tasks.

Furthermore, we test our schedules on synthetic as well as particular DAGs.

Pruhs, Stee and Uthaisombut [14] consider the problem of voltage scaling a set of tasks with precedence constraints to satisfy the dual objectives of makespan and energy minimization. They show that the search space can be restricted to those with constant power schedules (i.e. the sum of powers at which machines run is constant over time). They then show how to reduce this problem to obtain $\ln(m)$ approximation algorithms. However, they do not deal with the problem where there is inter-task communication in DAGs.

Shin et al. [19] proposed low-power priority-based scheduling which consists of two parts: an off-line component which determines minimum processor speed while guaranteeing deadlines of all tasks and an online component which dynamically varies processor speed in order to utilize both WST and VST. Shang et al. [18] proposed a history-based DVS for interconnected networks. Their technique leverages network history to predict future network needs, judiciously controlling the frequency (and voltage) of links to track actual network utilization. Those mechanisms resulted in 46% average power savings at the cost of 15.2% increase in network latency and 2.5% decrease in network throughput.

Lu, Benini and Micheli [10] presented a greedy on-line scheduling algorithm to facilitate power management for multiple devices. They ordered the execution of tasks so that devices can have continuous long idle periods during which they can be shut down. They achieved an average power savings of 33%. Mishra et al. [11] proposed two novel techniques for power management in distributed systems. The first is a static technique which uses a greedy algorithm to manage power in presence of parallelism. The second technique uses task reallocation that enhances the first algorithm by allowing out-of-order execution where preemption is allowed. Their technique saved an average of 10–20% more savings than a simple static power management technique. Chaeseok and Ha [7] proposed an energy efficient real-time multi-task scheduling by the use of buffers with DVS. They saved an average of 44% with reasonable machine specifications. The buffers increase CPU utilization by averaging the workload. Their technique was designed for multimedia applications where a slight buffering delay is tolerable.

3 Problem definition

We consider the problem of scheduling a directed a-cyclic task graph (DAG) on a heterogeneous distributed processor system with the twin objectives of minimizing finish time and energy consumption.

DAG is an a-cyclic graph with nodes representing tasks and edges representing execution precedence between tasks. A weight is associated with each node and edge. The node weight represents the task execution time and the edge weight represents the communication time between connected tasks. Along the lines of [2] a DAG is represented by the Tuple $G = (V, M, E, T, C,$ and $P)$; where, V is the set of n nodes, M is a set of m machines or processors in the system, E is the set of e edges between the nodes $E(n, c)$ represents the edge between nodes n and c, and T is the set of costs $T(n, k)$ which denotes the computation time of task n on processor k. Furthermore, $C(n, c)$ is the communication cost associated with $E(n, c)$ and it is zero if n and c are executed on the same processor. P is the set of costs $P(n, k)$, which represents the power consumed when task n is executed on processor k.

The length of a path is defined as the sum of node and edge weights in that path. $EST(n)$ and $EFT(n)$ represent the *earliest start time* and the *earliest finish time* over all processors, respectively. The *critical path* (*CP*) is the longest path from an entry node to an exit node. The *top distance* of a given node is the longest distance from an entry node to that node, excluding the computation cost of that node. The *bottom distance* of a node is the longest distance from the node to an exit node. Each task's mean execution cost over all processors is used to calculate *CP*, the *top distance*, and *bottom distance*. The *makespan* is defined as the time at which all nodes finished executing. In our case, the *makespan* will be equal to $EFT(y)$, where y is the exit node in the graph.

4 EADAGS algorithm

In this work a new algorithm for scheduling DAGs on distributed computing systems has been introduced. EADAGS combines Decisive Path Scheduling (DPS) with DVS to minimize both finish time and energy consumption. DPS [13], since it is one of the most efficient algorithms, was chosen. The new algorithm is called Energy Aware DAG Scheduling (EADAGS). It consists of two phases. In the first phase, after DPS is run on the DAG to provide a low finish time, the energy consumed is estimated for all processors. In the second phase, voltage scaling is applied during slack times to reduce energy while maintaining the schedule length.

EADAGS transforms a DAG to one with a single entry node and a single exit node, if not so already. This transformation is accomplished by adding a dummy entry node and/or exit node with zero costs. Next, the top and bottom distances from each node are calculated. The top and bottom distances are calculated using the mean computation value for each node. After building the *DP* for each node, EADAGS begins creating the scheduling queue, *ScheduleQ*, in a top-down fashion starting with the DAGs entry node and traversing down the *CP* (which is the DP of the exit

node). Nodes are prioritized based on the lengths of their *DPs*. The priorities are decided as follows: EADAGS puts the *CP* nodes into the *ScheduleQ* in the ascending order of their *top-distances*. A node is added to the queue only if all its predecessors have been added. If not, EADAGS attempts to schedule its predecessors first. The first predecessors added to the queue are those included in the nodes' *DP* other are sorted and added to *ScheduleQ* in increasing *top-distance*.

Next, EADAGS assign tasks in *ScheduleQ* to processors. At each step of the assignment, the selected processor provides the earliest finish time for the task under consideration, taking into account all the communications from the task's parents. If *EFT* of the exit node is larger than the sum of all the computation costs of the nodes on the best processor, EADAGS assigns all nodes to that processor and exits. The time complexity of first phase of EADAGS is $O(n^2)$.

Next, EADAGS computes the consumed energy. The total energy consumed when no voltage scaling is used, E_1 is first calculated by the following equations:

$$E_1 = T \times \sum_{k \in M} P_1(k)$$

$$P_1(k) = \frac{f V_1^2}{2}$$

where:

- $P_1(k)$ is the amount of power consumed by processor $k \in M$,
- f is the operating frequency of machine k,
- V_1 is the operating voltage of machine k,
- T is the *makespan*.

In the second phase of EADAGS, voltage scaling is applied to all processors during their *idle times* by reducing the execution rate to f_2 by lowering the voltage to a predetermined level V_2. Such voltage scaling is applied to a task only if slowing its execution would not increase the *makespan*. We also reduce the voltage level of processors during all remaining slack times. The total energy consumed after applying voltage scaling is $E_2 = \sum E_2(k)$ where:

- $E_2(k) = \frac{T_1 f_1 V_1^2 + T_2 f_2 V_2^2}{2}$ represents the energy consumed by processor $k \in M$ when voltage scaling is used
- $T_1 = \sum_i T(i) + \sum_{ij} C(i, j)$ is the total task and communication time when operating at V_1
- $T(i)$ is the computation time of task i on the chosen processor
- $C(i, j)$ is the communication cost between tasks i and j if i and j are not scheduled on the same processor.
- T_2 is the total time processor k operates at V_2 (includes idle times during which the processor operates at V_2)

A nonblocking send protocol has been assumed in which only the sending processor has to process the communication while the receiving processor has a buffer to receive all transmitted data without interrupting its job. The difference between E_1 and E_2 represents the energy that could be saved. The percentage average energy savings $= \frac{E_1 - E_2}{E_1} \times 100$. A high level description of EADAGS appears in Table 1 and a detailed description is given in Fig. 1.

5 Simulation and results

In the simulation, the first test suite uses random directed a-cyclic graphs to evaluate EADAGS. The input parameters used to generate the graphs were:

- Number of nodes (tasks) in the graph, n.
- Number of available processors in the system, m.
- Shape parameter of the graph, α. We assume that the height of the DAG is randomly generated from a uniform distribution with mean equal to $\alpha \times \sqrt{n}$. The width of the DAG is also randomly selected from a uniform distribution with mean equal to $\frac{\sqrt{n}}{\alpha}$. If $\alpha = 1$, the graph is balanced. A DAG with high parallelism can be generated by selecting $\alpha \gg 1$. Whereas $\alpha \ll 1$ will generate a long DAG with small degree of parallelism.
- Out-degree of a node, *out-degree*, represents the average number of outgoing edges from each node. Each node's *out-degree* is randomly generated from a uniform distribution with mean equal to *out-degree.*
- Communication to Computation ratio, *CCR*, is the ratio of the average communication to average computation cost. If a DAG's *CCR* is less than 1, it is computation-intensive; if it is much greater than 1, it is communication-intensive.
- Computation Range, β, represents the range of computation costs on processors. A high β causes significant difference of node's computation costs among processors, whereas a low β means that the expected execution times of a node on any processor are almost equal.
- Processor to node ratio, *PNR*, represents the availability of processors with respect to number of nodes. A *PNR* of 100% means the number of processors is equal to the number of nodes.

In generating random DAGs, we get 10,800 DAGs when the parameters were varied as follows:

$n = \{10, 20, 40, 60, 80, 100, 500, 1000\}$
$CCR = \{0.1, 0.5, 1, 5, 10\}$
$\alpha = \{0.5, 1, 2\}$
$Out\text{-}degree = \{1, 2, 3, 4, 5, 100\}$
$\beta = \{0.1, 0.25, 0.5, 0.75, 1.0\}$
$PNR = \{25\%, 50\%, 100\%\}$

Table 1 EADAGS algorithm

Let G represent a DAG
Let M be the set of m processors in the system
EADAGS
    Transform G to a DAG with a single entry node and a single exit node
    Compute DP of each node $n \in G$
    //DP of the exit node is the critical path, CP
    Fill *ScheduleQ* with nodes
    //Starting from the entry node traversing CP in increasing *top-distance*.
    while $ScheduleQ \neq \Phi$ **do**
        $i \leftarrow$ head ($ScheduleQ$)
        Schedule i on processor $p \in M$ that provides earliest finish time of i.
        Remove i from *ScheduleQ*
    end while
    if scheduling all nodes on the fastest processor provides a shorter *makespan*,
    do so and discard prior schedule
    $T \leftarrow makespan$

    Total energy consumed before voltage scaling $E_1 = \frac{fTV_1^2}{2}$
    Total energy consumed when employing voltage scaling, $E_2 = ScaledEnergy()$
end *EADAGS*

***ScaledEnergy*( )**
// Returns the total amount of energy consumption on all processors when voltage scaling has been applied
    for each processor $p \in M$ **do**
        for each node $n \in G$ scheduled on p **do** //traverse first scheduled to last
            if (executing n on scaled voltage fits within the next slack) **then**
                Scale down the operating voltage during execution of n
            end if
        end for
    Energy consumed by processor p = sum of energy consumed by all nodes scheduled on p
    end for
    E = Sum of energy consumed by all processors
    return E
end *ScaledEnergy*

Processors were assumed to have three different operating voltage levels based on the Motorola CMOS 6805 microcontroller which is rated at 6 MHz at 5.0 Volts, 4.5 MHz at 3.3 Volts, and 3 MHz at 2.2 Volts. First operating voltage was 5 V; when using this voltage if the processor becomes idle, it is shut down. We will refer to this operating voltage as 5 V/off this level is used for reference only since it has physical limitations. The other two operating voltages are 2 V and 3.3 V which slow the processor during task execution.

5.1 Results for random DAGs

The energy consumption was measured for EADAGS and DPS for different random DAGs. Then, test sets were created by combining results from DAGs with similar properties, such as the number of nodes or the *CCR*.

The first test set was achieved by combining DAGs with respect to number of nodes. The energy savings were averaged over DAGs with varying *CCR*, α, β, *out-degree*, and *PNR*. Figure 6 shows the average energy saved by EADAGS over DPS with respect to number of nodes. The energy savings increased with increasing number of nodes. Average energy savings is 28% for a DAG of 10 nodes which gradually increased to 46% for 1,000 nodes DAG. As number of nodes increase more dependencies between tasks come into play, which provide greater opportunities to use slack time to save energy. Average energy savings ranges between 30–46% for the 5 V/off, 29–48% when 2 V is used, and 28–46% when using 3.3 V.

The second test set combines DAGs with respect to *CCR*. The energy savings were averaged over different DAGs with varying n, α, β, *out-degree*, and *PNR*. In Fig. 7 the

Fig. 1 Algorithm *EADAGS*

```
EADAGS
    Transform G to a DAG with a single entry node and a single exit node
    Compute DP for each node n ∈ G                                   // O(n^2)
    // DP of the exit node is the critical path, CP
    // Fill ScheduleQ with nodes in CP in increasing top-distance.
    ScheduleQ ← Φ
    for each node n ∈ CP do                                          // O(n^2)
        // Traverse in increasing top-distance.
        ScheduleQ = addQ(n)
    end for
    //Schedule nodes in ScheduleQ to processors
    while ScheduleQ ≠ Φ do                                           //O(n)
        Pick the head node i in ScheduleQ
        for each processor k ∈ M do                                  // O(m)
            s_ik = StartTime(i, k)
            EFT(i, k) = s_ik + T(i, k)
        end for
        EFT(i) = min_k ∈M EFT(i, k)
        Schedule i on processor p ∈ M that gave minimum earliest finish time
        Remove i from ScheduleQ
    end while
    if EFT(y) ≥ min_k∈M Σ_i∈G T(i, k)
        Schedule all nodes on the processor p ∈ M, which provided the minimum
    end if
    T ← EFT(y)        // makespan
    E_1 = fTV_l^2 / 2
    E_2 = ScaledEnergy( )
end EADAGS
```

Table 2 Notations

Let

- G represent a DAG
- $y \in G$ be the exit node of G
- M be the set of m processors in the system
- R_k represent the ready time of machine k
- r_n represent the ready time of node n
- f be the frequency of operation
- s_{ik} represent the start time of node i on machine k
- $Succ(n)$ represent the list of all successor nodes of node $n \in G$
- $pred(n)$ is the list of all predecessors of node $n \in G$
- *ScheduleQ* is the queue of tasks in order of execution
- $T(i, k)$ represent the execution time of node $i \in G$ on machine k
- $C(n, c)$ represent the communication cost from node n to node c
- $\mathrm{EFT}(i, k)$ represent the earliest finish time of node $i \in G$ on machine k
- $\mathrm{EFT}(i)$ represent the scheduled finish time of node $i \in G$
- $\mathrm{EFT}_2(i)$ represent the finish time for node i with the scaled down voltage
- $\mathrm{EST}(i)$ represent the scheduled start time of node $i \in G$
- T represent the *makespan* of G
- V_1 be the voltage of operation
- V_2 be the scaled down voltage of operation
- $T_1(n)$ and $T_2(n)$ represent the execution time for any node $n \in G$ before and after voltage scaling respectively
- $E(k, n)$ represent the energy consumed by processor k to execute node n
- E_1 represent the total energy consumption before voltage scaling
- E_2 is the total energy consumption after scaling down voltage
- $E_1(k)$ and $E_2(k)$ represent energy consumed by processor $k \in M$ before and after voltage scaling respectively

average energy savings has been plotted with respect to *CCR*. The average energy savings increased with increasing *CCR*. When *CCR* increases, processors are idle longer due to communication between tasks. EADAGS is able to use such slack times to achieve power savings. The average energy savings over DPS ranges from 35% for $CCR = 0.1$ to 41% when $CCR = 10$ for 5 V/off technique. Savings are 28–44% when processors are scaled down to 2 V. Sav-

Fig. 2 Procedure *addQ*

```
addQ(n)
// Adds parents of node n ∈ G and n to ScheduleQ
// Returns ScheduleQ
    for each parent b of n not visited          // in increasing top-distance
        addQ(b)
    end for
    Add to n to ScheduleQ and mark it visited
    return ScheduleQ
end addQ
```

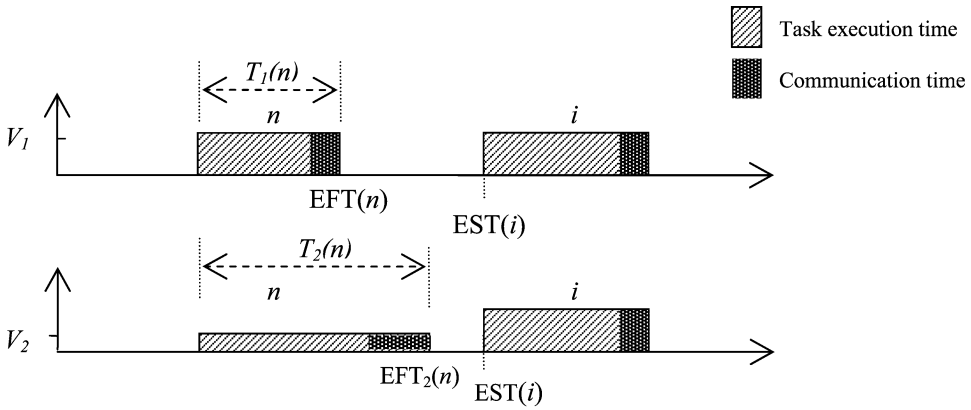
Fig. 3 Procedure *StartTime*

```
StartTime (node n, machine k)
// Returns the earliest available start time of node n ∈ G on machine k ∈ M
    s_nk ← R_k
    for each parent b of n do                   // O(n)
        s_nk = max(s_nk, EFT(b) + C(b, n))      // if b is scheduled on k, C(b, n) = 0
    end for
    return s_nk
end StartTime
```

Fig. 4 Procedure *ScaledEnergy*

```
ScaledEnergy( )
// Returns the total amount of energy consumption after applying voltage scaling
    for each processor k ∈ M do                                   // O(m)
        for each node n ∈ G scheduled on k do                     // O(n) from first to last scheduled
            T2(n) = (V1^2 / V2^2) × T1(n)
            EFT2(n) = s_nk + T2(n)
            if EFT2(n) + C(n, c) < min EST(c) for each c ∈ succ(n) then
                // if c is scheduled on k, C(n, c) = 0
                EFT(n) = EFT2(n) //update EFT(n)
                Let i be the node scheduled immediately after n on k
                // use scaled voltage, see Fig. 5
                E(k, n) = T2(n) × V2^2 + (EST(i) − EFT(n)) × V2^2
            else
                //Use original voltage for execution and scaled voltage during idle time,
                // see Fig. 5
                E(k, n) = T1(n) × V1^2 + (EST(i)−EFT(n)) × V2^2
            end if
        end for
        E2(k) = +E(k, n)
    end for
    E2 = Σ_{k∈M} E2(k)
    return E2
end ScaledEnergy
```

Fig. 5 Timeline for machine k



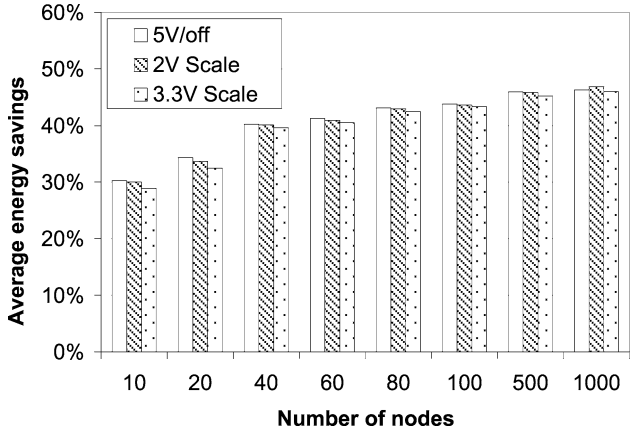


Fig. 6 Average energy savings vs. number of nodes

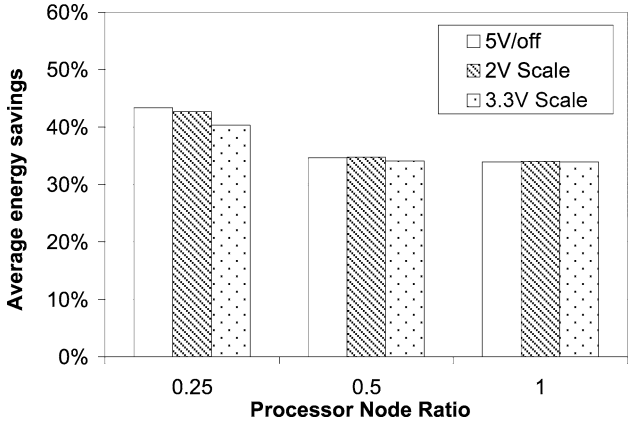


Fig. 8 Average energy savings vs. *PNR*

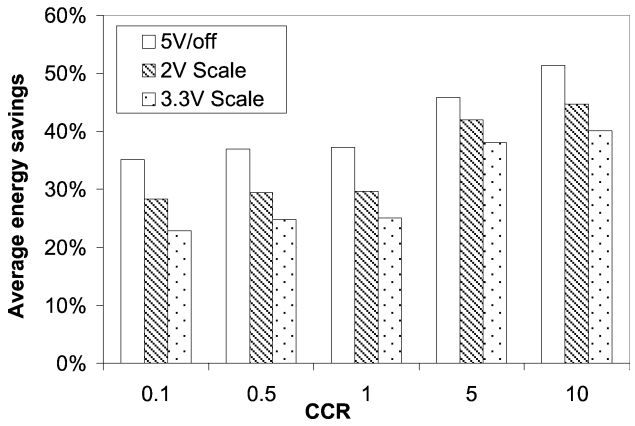


Fig. 7 Average energy savings vs. *CCR*

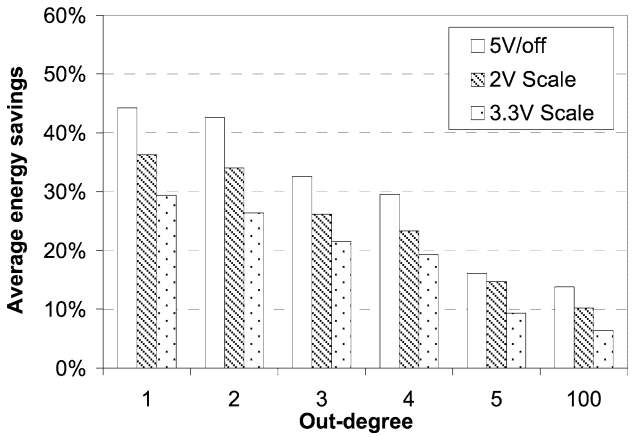


Fig. 9 Average energy savings vs. out-degree

ings are between 22–40% when processors are scaled down to 3.3 V.

The third test set combines DAGs with respect to processor to node ratio, *PNR*. The energy savings is averaged over randomly generated DAGs with varying n, *CCR*, α, β, and *out-degree*. Figure 8 shows average energy savings with respect to *PNR*. It shows decrease in the average energy savings with increasing *PNR*. This is because increasing number of processors allows several parallel task executions, thus minimizing the wait times. Average energy savings measured were 43% for $PNR = 25\%$ in the 5 V/off technique, 42% when processors operate at 2 V and 40% if they operate at 3.3 V. For $PNR = 100\%$ the average energy savings was 33%.

Figure 9 shows the average energy savings with respect to different values for *out-degree*. The results indicate that increase in *out-degree* results in smaller average energy savings. A larger *out-degree* implies higher parallelism which in turn implies smaller slack time. The amount of energy that could be saved ranged between 44% and 13% for 5 V/off, 36% to 10% when voltage is scaled to 2 V, and 29% to 6% when 3.3 V scale is used.

5.2 Results for specific problems

The second test suite used to evaluate the performance of EADAGS used task graphs of two real world problems: Gaussian elimination [1] and Molecular dynamics code [25].

5.2.1 Gaussian elimination

Gaussian elimination is used to determine the solution of a system of linear equations, for determining the rank of a matrix, and for calculating the inverse of an invertible square matrix [4]. Gaussian elimination systematically applies elementary row operations on set of linear equations in order to convert it to upper triangular form. Once the coefficient matrix is in upper triangular form, back substitution is used to find a solution. The computational complexity of Gaussian elimination is $O(n^3)$ where $n \times n$ is the size of the matrix. The total number of tasks in a graph is $\frac{n^2+n-2}{2}$. The DAG for the Gaussian elimination algorithm for $n = 5$ is shown in Fig. 10 where n is the matrix size.

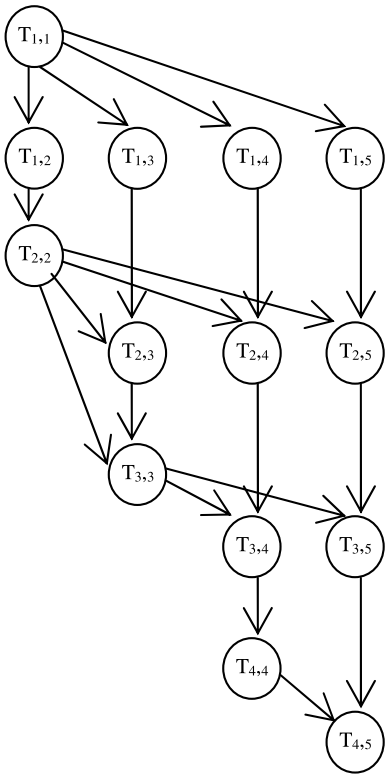


Fig. 10 Gaussian elimination task graph for a matrix of size 5

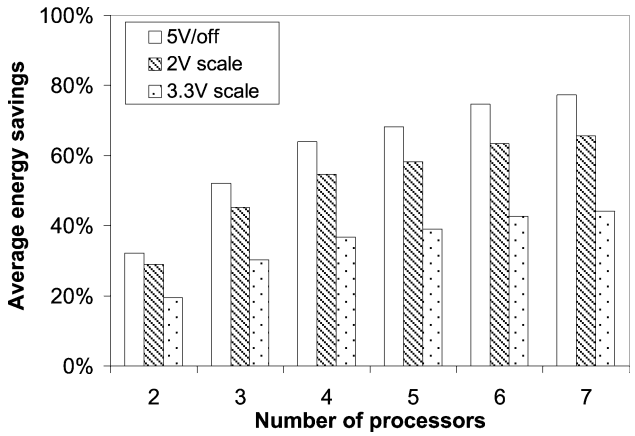


Fig. 11 Average energy savings for Gaussian elimination

In the simulation, a matrix of size 8×8 has been used to evaluate EADAGS. Since the structure of the graph is fixed only the number of processors and the *CCR* values were varied. For a matrix of size 8 the total number of tasks in the graph is 35 and largest number of tasks at a single level is 7 so the number of processors is bounded to 7. *CCR* values were 0.1, 0.5, 1.0, 5.0, and 10. In this experiment since the same operation is executed at every processor and the same information is communicated from one processor to another, a uniform computation cost for all tasks and equal communication cost for all communication links were assumed.

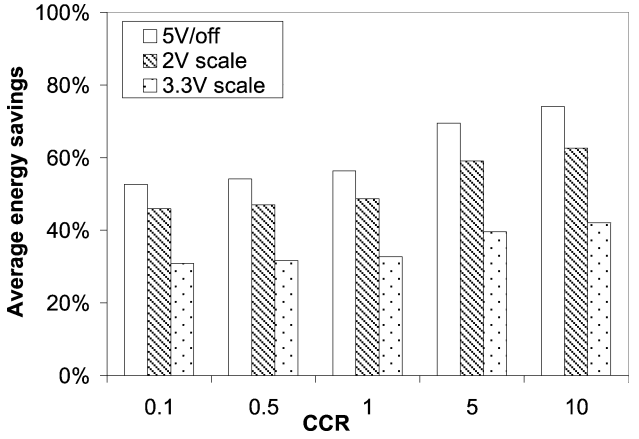


Fig. 12 Average energy savings for Gaussian elimination

Figures 11 and 12 show the average energy savings using EADAGS over DPS with respect to number of processors and *CCR* values. Figure 11 shows an increase in the average energy savings with increasing number of processors. This is because at each level only a certain number of tasks can be executed at the same time so increasing the available processors number produces more idle time, thus increasing the energy savings. The average energy savings measured was 32% for 2 processors and 60% when 7 processors are used.

Figure 12 plots the average energy savings with respect to different *CCR* values. The average energy savings increase with increasing *CCR*. When *CCR* increases, processors are idle longer due to communication between tasks. EADAGS is able to use such idle times to achieve energy savings. The average energy savings of EADAGS over DPS ranges from 52% for $CCR = 0.1$ to 74% when $CCR = 10$ for 5 V/off technique. Savings are smaller for 2 V scale; they range between 45% and 62%. Savings are even smaller for the 3.3 V scale; they are 30% for $CCR = 0.1$ and 42% for $CCR = 10$.

5.2.2 *Molecular dynamics code*

Figure 13 represents the DAG of a molecular dynamics code as given in [1]. Again, since the graph has a fixed structure and fixed number of nodes, the only parameters that could be varied were the number of processors and *CCR* values.

Since there are at most seven tasks at any level in Fig. 13, the number of processors was bound to seven. We assumed that the computation costs of all nodes are not equal and the communication costs were also not equal for all links since the task computed at each node and the data communicated from one node to another is different. Five values for *CCR* were used in our experiments: 0.25, 0.5, 1, 5, and 10.

Figures 14 and 15 show the average energy savings of EADAGS over DPS with respect to number of processors and *CCR* values respectively. Figure 14 shows increase in

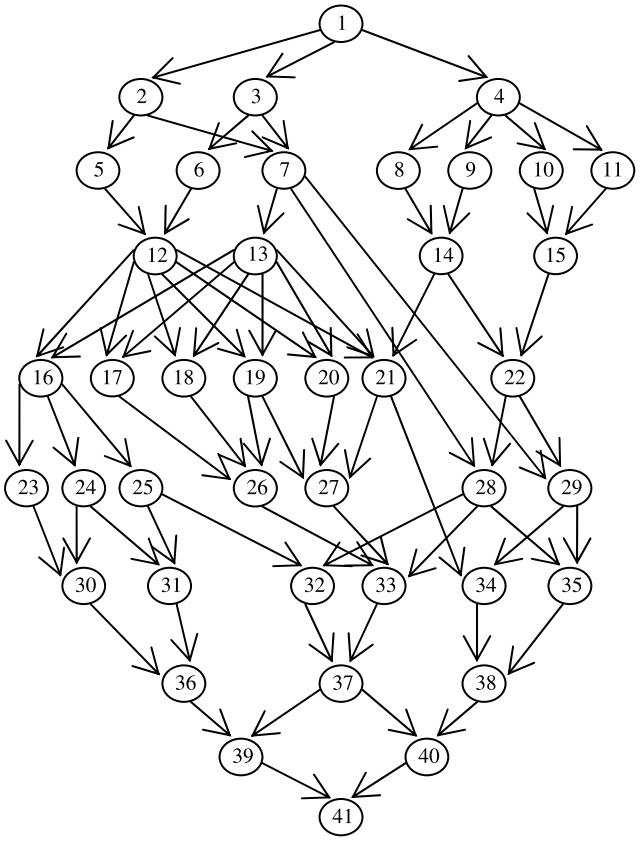


Fig. 13 Directed Acyclic graph (DAG) for a molecular dynamics code

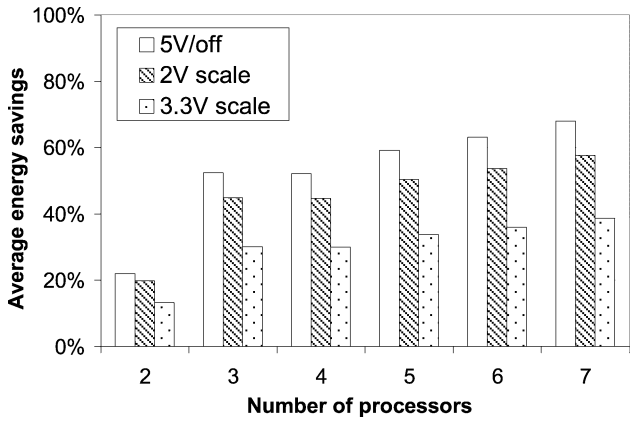


Fig. 14 Average energy savings for molecular dynamics code

average energy savings with increasing number of processors. This is because at each level only a certain number of tasks can be executed at the same time so increasing the available number of processors means more idle time, thus increasing the energy savings. The average energy savings over DPS ranges from 20% for 2 processors and increases to 55% for 7 processors. Figure 15 plots the average energy savings with respect to different *CCR* values. The average savings increase with increasing *CCR*. When

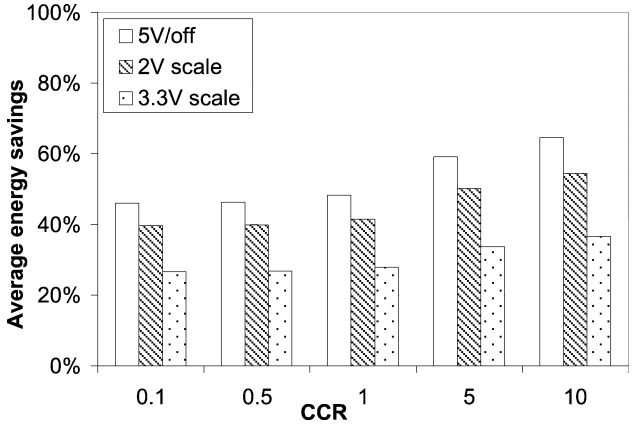


Fig. 15 Average energy savings for molecular dynamics code

CCR increases, processors are idle longer due to communication between tasks. EADAGS is able to use such idle times to achieve energy savings. The average energy savings over DPS ranges from 45% for $CCR = 0.1$ to 64% when $CCR = 10$ for 5 V/off technique. Savings are smaller for 2 V operating voltage they range between 39% and 54%. Savings are even smaller for the 3.3 V operating voltage they are 26% for $CCR = 0.1$ and 36% for $CCR = 10$.

6 Conclusion

We proposed a new scheduling algorithm, EADAGS. The algorithm tries to minimize finish time as well as energy consumption by the use of dynamic voltage scaling. On average, it provides energy savings of 40% over DPS without increasing the *makespan*. These results were based on a rigorous software simulation study over a large set of randomly generated graphs as well as graphs for real-world problems with various characteristics.

The amount of energy savings increased with increasing the number of nodes due to the increase in idle time due to task dependency. Also, increase of *CCR* resulted in an increase in the average energy savings. When *CCR* increases, processors incur longer idle times due to communication between tasks. EADAGS was able to use such idle times to achieve energy savings. The results showed that the overall energy savings decreased with increasing *PNR*. This is because increasing number of processors allows several parallel task executions, thus minimizing the wait times and the average energy savings. An increase in out-degree resulted in smaller average energy savings. A larger out-degree allows many processors to run in parallel reducing the idle times for all processors and so less power to save. Future work can involve applying the voltage scaling technique to other DAG scheduling algorithms.

References

1. Baskiyar, S.: Scheduling DAGs on message passing m-processors systems. IEICE Trans. Inf. Syst. **E-83-D**(7), 1497–1507 (2000)
2. Baskiyar, S., Dickinson, C.: Scheduling directed A-cyclic graphs on a bounded set of heterogeneous processors using task duplication. In: Lecture Notes in Computer Science, vol. 2913, pp. 259–267. Springer, Berlin (2003)
3. Chandrakasan, A., Gutnik, V., Xanthopoulos, T.: Data driven signal processing: an approach for energy efficient computing. In: International Symposium on Low Power Electronics and Design, pp. 347–352, Aug. 1996
4. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms. MIT, Cambridge (2001)
5. Dongarra, J.J., Walker, D.W.: The quest for petascale computing. IEEE Trans. Comput. Sci. Eng. **3**(3), 32–39 (2001)
6. Iverson, M.A., Ozguner, F.: Dynamic competitive scheduling of multiple DAGS in a distributed heterogeneous environment. In: Proc. of the Workshop on Heterogeneous Processing, pp. 70–78, March 1998
7. Im, C., Ha, S.: Dynamic voltage scaling for real-time multitask scheduling using buffers. In: Proc. ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems, pp. 88–94 (2004)
8. Kwok, Y.K., Ishfaq, A.: Link contention-constrained scheduling and mapping of tasks and messages to a network of heterogeneous processors. In: Proc. of International Conference on Parallel Processing, pp. 551–558, September 1999
9. Li, K., Kumpf, R., Horton, P., Anderson, T.: A quantitative analysis of disk drive power management in portable computers. In: PTUC. Winter USENIX Conference, pp. 279–292, January 1994
10. Lu, Y.H., Benini, L., Di Micheli, G.: Low-power task scheduling for multiple devices. In: International Workshop on Hardware/Software Codesign, pp. 39–43, May 2000
11. Mishra, R., Rastogi, N., Zhu, D., Mossé, D., Melhem, R.: Energy aware scheduling for distributed real-time systems. In: Proc. Int'l Parallel and Distributed Processing Symposium, pp. 9–16, April 2003
12. Mossé, D., Aydin, H., Childers, B.R., Melhem, R.: Compiler-assisted dynamic power-aware scheduling for real-time applications. In: Proc. Workshop Compiler and OS for Low Power, October 2000
13. Park, G., Shirazi, B., Marquis, J.: Decisive path scheduling: a new list scheduling method. In: Proceedings of the International Conference on Parallel Processing, pp. 472–480, August 1997
14. Pruhs, K., Stee, R.V., Uthaisombut, P.: Speed scaling of tasks with precedence constraints. Theory Comput. Syst. **43**, 67–80 (2008)
15. Pouwelse, J., Langendoen, K., Sips, H.: Dynamic voltage scaling on a low-power microprocessor. In: Proc. of the 7th Annual International Conference on Mobile Computing and Networking, pp. 251–259, July 2001
16. Radulescu, A., Van Gemund, A.J.C.: Fast and effective task scheduling in heterogeneous systems. In: 9th Heterogeneous Computing Workshop, pp. 229–239, May 2000
17. Reuter, C., Schwiegershausen, M., Pirsch, P.: Heterogeneous multiprocessor scheduling and allocation using evolutionary algorithms. In: Proc. of the IEEE International Conference on Application-Specific Systems Architecture and Processors, pp. 294–303, July 1997
18. Shang, L., Peh, L.-S., Jha, N.K.: Dynamic voltage scaling with links for power optimization of interconnection networks. In: Proc. of the 9th International Symposium on High-Performance Computer Architecture, pp. 91–102, February 2003
19. Shin, D., Lee, S., Kim, J.: Intra-task voltage scheduling for low-energy hard real-time applications. IEEE Des. Test Comput. **18**, 20–30 (2001)
20. Tin, M., Seigel, H.J., Antonio, J.K., Li, Y.A.: Minimizing the application execution time through scheduling of subtasks and communication traffic in a heterogeneous computing system. IEEE Trans. Parallel Distrib. Syst. **8**(8), 857–870 (1997)
21. http://www.bsac.eecs.berkeley.edu/archive/users/warneke-brett/SmartDust, accessed on February 2006
22. Topcuoglu, H., Hariri, S., Wu, M.Y.: Task scheduling algorithms for heterogeneous processors. In: Proc. of the 8th Heterogeneous Computing Workshop, pp. 3–14, April 1999
23. Topcuoglu, H., Hariri, S., Wu, M.Y.: Performance-effective and low complexity task scheduling for heterogonous computing parallel and distributed systems. IEEE Trans. Parallel Distrib. Syst. **13**(3) (2002)
24. Wang, L., Siegel, H.J., Rowchowdhury, V.P., Maciejewski, A.A.: Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach. J. Parallel Distrib. Comput. **47**, 8–22 (1997)
25. Wu, M.Y., Gajski, D.D.: Hypertool: a programming aid for message-passing systems. IEEE Trans. Parallel Distrib. Syst. **1**(3), 330–343 (1990)
26. Yang, P., Wong, C., Marchal, P., Catthoor, F., Desmet, D., Verkest, D., Lauwereins, R.: Energy-aware runtime scheduling for embedded-multiprocessor SOCs. IEEE Des. Test Comput. **18**(5), 46–58 (2001)
27. Zhang, Y., Hu, X., Chen, D.: Task scheduling and voltage selection for energy minimization. In: Design Automation Conference, pp. 183–188, New Orleans, June 2002

Sanjeev Baskiyar is an Associate Professor in the Department of Computer Science and Software Engineering at Auburn University, Auburn, Alabama. He received the Ph.D. and M.S.E.E. degrees from the University of Minnesota, Minneapolis, B.E. degree in Electronics and Communications from the Indian Institute of Science, Bangalore and a B.Sc. degree in Physics with honors and distinction in Mathematics from St. Xaviers's College, India. He has taught courses in Computer Architecture, Real-time and Embedded Computing, Operating Systems, Microprocessor Programming and Interfacing and VLSI Design. His research interests are in the areas of Computer Architecture and Grid Computing. His experience includes working as an Assistant Professor at Western Michigan University, a Senior Software Engineer in Unisys and a Computer Engineer in Tata Motors.

Rabab Abdel-Kader received the Ph.D. degree from the department of Computer Science and Software Engineering at Auburn University, Auburn, AL and the MS degree in Electrical Engineering from Tuskegee University with high honors. She is currently an Assistant Professor in the Faculty of Engineering at Suez Canal University, Egypt.