ENERGY AWARE TASK SCHEDULING ON HETEROGENEOUS SYSTEMS

Rabab Farouk Abdel-Kader

Certificate of Approval:

ENERGY AWARE TASK SCHEDULING ON HETEROGENEOUS SYSTEMS

Rabab F. Abdel-Kader

A Dissertation

Submitted to

the Graduate Faculty of

Auburn University

in Partial Fulfillment of the

Requirements for the

Degree of

Doctor of Philosophy

Auburn, Alabama
December 17, 2007

ENERGY AWARE TASK SCHEDULING ON HETEROGENEOUS SYSTEMS

Rabab F. Abdel-Kader

Signature of Author

Date of Graduation

DISSERTATION ABSTRACT

ENERGY AWARE TASK SCHEDULING ON HETEROGENEOUS SYSTEMS

Rabab F. Abdel-Kader

Doctor of Philosophy December 17, 2007
(M.S. Tuskegee University 2002)
(B.S. Suez Canal University, 1998)

128 Typed Pages

Directed by Sanjeev Baskiyar

We consider the problem of scheduling directed a-cyclic task graphs (DAG) on heterogeneous distributed processor systems 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 applications. But due to many new applications on embedded systems such as high performance DSP in image processing, multimedia, and wireless security, there is a strong need for scheduling algorithms which lower energy consumption and yet attain good finish times.

In this research, we employ dynamic voltage scaling (DVS) within the scheduling heuristics to achieve the twin objectives. The processors used can run on different discrete operating voltages. Processors can scale down their voltages to slow down in order to reduce energy consumption whenever they idle due to task dependencies. Specifically, we combine Decisive Path Scheduling (DPS) and Heterogeneous N-predecessor Duplication (HNPD) with DVS. Using simulations, we show average energy consumption reductions of 40% over DPS and 28% over HNPD.

The simulations used large number of randomly generated DAGs with various characteristics as well as DAGs of real world problems. Energy savings increased with increasing number of nodes or increasing Communication to Computation Ratios (CCR) whereas it decreased with increasing parallelism (out-degree) or increasing number of available processors. Increasing nodes, increase tasks dependencies and thus idle times. When CCR increases, processors are idle longer due to communication between tasks. Our algorithms used such idle times to achieve energy savings. An increase in out-degree resulted in smaller average energy savings. A larger out-degree allows more processors to run in parallel, reducing idle times.

Style manual: IEEE Standard

Software used: Microsoft Word 2007, Microsoft Excel 2007, Microsoft Visual Studio 2003,and C++

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

We consider scheduling on heterogeneous distributed computing systems interconnected by high-speed networks. 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 [Reut97]. Therefore, researchers have proposed many static, dynamic and even hybrid algorithms to minimize execution time of applications running on a heterogeneous system [Iver98][Kwok99] [Radu00][Seig97][Topc99][Wang97]. Another challenge facing distributed computing is energy consumption [Dong01]

There are many applications which require both low finish time and low energy consumption. Energy consumption is a major issue in many real-time distributed embedded systems. Furthermore, most applications running on an energy limited system inherently constrain the finish time. Low-cost, low-energy sensor networks composed of Smart Dust Mote [Smar06] are examples of such systems. New wireless communication systems are expected to evolve using this system. These networks are distributed networks operating on energy constraints, also called energy-aware distributed systems

(PADS). Hence there is a need for scheduling algorithms which would effectively reduce the overall energy consumed and yet attain the best possible finish time.

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. 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 energy schedules can help run such applications 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, applications may not require the maximum achievable speed at all times. The top energy consumers in a computer system [Kump94] are display (68%), disk (20%), and CPU (12%). There seems little which can be done to minimize screen energy-consumption, beyond employing a screen-saver and relying on hardware improvements. Disk energy consumption is minimized by spinning down the disk when it has been inactive.

However, in the future, we may well see ubiquitous computing devices with neither disks nor conventional displays. For such devices, minimizing the energy consumed by the CPU will be critical if the replacements of disks and displays consume relatively smaller fractions of total energy.

A study by Argonne National Laboratory has indicated that a 2.5 petaflop supercomputer, made of over a hundred thousand CPUs, will be available by 2010. The study predicts that such a system will cost $16 million and would require 8 mega watts of energy to operate at a cost of about $8 million per year. Hence, high energy prices and rising concerns about the environmental impact of electronics systems highlight the importance of incorporating low energy design schemes at all levels of such systems. Current microprocessors from AMD, Intel and Transmeta allow the speed of the microprocessor to be set dynamically.

Another study by NASA in 1998 predicted energy need of 25 megawatts for Japan's NEC earth simulator, which is capable of executing 40 Tflops. That amount increased to 100 megawatts in a more recent study that much energy is enough to light 1.6 million 60-watts light bulbs, the lighting requirements of a small city.

Reducing systems components energy and energy consumption decrease systems expenses. Assuming a rate of $100 per megawatt a Pflops machine consuming 100 megawatts of energy would cost $10,000 per hour approximate $85 million dollar a year.

These estimates do not include air cooling expenses which are commonly 40% of systems operating cost. For such systems even small reduction in overall energy consumption would significantly impact Pflops systems' operational costs.

1.1 Contributions

In this dissertation we presented two scheduling algorithms for scheduling Directed Acyclic Graphs on a distributed computing system for low energy.

The first proposed algorithm combines Decisive Path Scheduling (DPS) with dynamic voltage scaling for the twin objectives of low energy consumption and minimum execution time. We call it Energy Aware DAG Scheduling (EADAGS). The second algorism proposed combines Heterogeneous N-predecessor Duplication (HNPD) with dynamic voltage scaling to minimize both finish time and consumed energy, we identify that algorithm as Energy Aware Graph Scheduling with Duplication (EAGS-D).

In both cases first the initial algorithm is completed using either DPS or HNPD for minimum finish time, then the amount of consumed energy is estimated and the voltage scaling algorithm is simulated to minimize the consumed energy without affecting the finish time.

The remainder of this dissertation is organized as follows. In the next Chapter, we describe different types of scheduling on both homogenous and heterogeneous systems, energy estimation and energy optimization techniques, and prior work on scheduling for low energy are discussed. Chapter 3 explains the concept of dynamic voltage scaling and how it can be used to reduce the consumed energy in computing systems. Chapter 4 defines DAG and explains some of the definitions and terminology used by the scheduling algorithms. Chapter 5 introduces both scheduling algorithms presented in this work EADAGS and EAGS-D. Chapter 6 is for the simulation and analysis of results. Finally the conclusion and suggestions for future work are presented in Chapter 7.

CHAPTER 2

RELATED WORK

Traditional scheduling algorithms did not consider the amount of energy consumption. Instead, they focus on performance or fairness. Recently, low energy system design has gained significant attention largely due to demands from the portable electronics industry. System design for low energy is also very important for other industries such as automotive, telecommunications, information technology, etc. This is due to the fact that low energy designs can offer significant reduction in system packaging costs and improvement in reliability.

Two main design aspects of scheduling are how to build the scheduling queue and how to choose the optimal processor. 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 so as to get closer to the objectives of schedule length, number 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 [Topc02].

Heterogeneous N-predecessors Decisive Path (HNPD) is based on DPS but with Task duplication. The performance of HNPD was proven to outperform two of the best existing heuristics, Heterogeneous Earliest Finish Time (HEFT) and Static Task Duplication Scheduling (STDS), in terms of finish time and the number of processors employed over a wide range of parameters.

Low energy scheduling research can be classified in two major categories

1. Energy estimation techniques (energy model)
2. Energy optimization techniques.

2.1 Power Estimation Techniques

Existing energy estimation methodologies can be classified based on their level of abstraction, namely instruction level, architecture level, and gate level.

- Instruction level: application for this can be for embedded processing systems as presented by Nikoladis in [Niko02]. He uses an assembly or machine level program as input and gives an estimate of the energy consumed for that specific program on a specific processing system. This provided an accurate estimation of energy consumption even in the presence of instantaneous energy supply variation.

- Architecture level: it provides cycle-by-cycle energy consumption data of the architecture on the basis of the instruction/data flow stream. At the architecture level a technique presented by Landman uses the black box energy model for the architecture level components to estimate the energy consumed while preserving the accuracy of the gate or circuit level estimation [Land94].
- Gate level: Ishehara summarized and compared different techniques for energy estimation and proved that gate level estimation of energy consumption is the most accurate measurement [Ishe96]. The techniques for energy estimation at the gate level and low levels of abstraction can be classified into
    - Simulation based techniques: the earliest techniques proposed suggested monitoring both the supply voltage and current waveforms. These techniques were to slow to handle very large circuits so other techniques were introduced assuming that the supply and ground voltages are constant and estimate only the supply current waveform.
    - Probabilistic techniques: most of the probabilistic techniques are

applicable to combinational circuits only. In these techniques user supplied input signal probabilities are propagated into the circuit. To achieve this, special models for the components have to be developed and stored in a module library.

    - Statistical techniques: they do not require any specialized model for the components. The idea is to simulate the circuit with randomly generated input vectors until energy converges to the average energy. The convergence is tested by statistical mean estimation techniques.

2.2 Power Optimization Techniques

Competition is driving the requirement for energy optimization. Systems are designed with low energy consumption as one of the important criteria. Energy optimization can be achieved through both hardware and software.

- Hardware Optimization:
    - Behavior level: transformations, scheduling, resource allocation, etc.
    - Architecture level: low energy flip-flops, low energy adder, etc.
    - Circuit level: low energy circuits.
- Software Optimization
    - Instruction level: low energy compiling, low energy instruction scheduling.

- o System level: Dynamic energy management, low energy memory management, etc.

2.3 Software energy optimization on system level

There are two techniques that can reduce energy consumption on system level scheduling: Dynamic Energy Management (DPM) and Dynamic Voltage Scaling (DVS).

The DPM technique 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 in DPM. 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 period, under the assumption that it is highly likely that a component remains idle if it has been idle for the timeout period.

The fundamental premise for the applicability of DPM is that systems (and their components) experience non-uniform workloads during operation time. Such an assumption is valid for most systems, both when considered in isolation and when systems are connected via internet. A second assumption of DPM is that it is possible to predict, with a certain degree of confidence, the fluctuations of workload. The analytical process of prediction should not consume significant energy. Typically, a energy manager (PM) implements a control procedure based on some observations and/or assumptions on the workload.

In DVS technique, computation and communication tasks are run at reduced voltages and clock frequencies which fill idle periods but 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. It is a trade-off between processor speed and energy consumption which is especially useful in real-time systems. The energy consumption of a processor running at high speed and high voltage is much larger than running at low speed and low voltage. For example, reducing the supply voltage from 5 V to 3.3 V in some cases has reduced energy by 56% [Pouw01].

DVS essentially fills the slack times by elongated computation or communication times. There are two types of slack times: 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 WVST can be known only after execution.

2.4 Scheduling to lower energy consumption

For uniprocessor real-time systems, many schemes have been proposed to manage energy consumption. Mosse et. al. [Moss00] proposed and analyzed several schemes to dynamically adjust processor speed for slack reclamation. They used a compiler to assist the operating system in changing the CPU operating levels to reduce energy consumption.

Weiser [Weis94] discussed several methods for varying the clock speed dynamically under control of the operating system. He proved that by adjusting the clock speed at a fine grain, substantial CPU energy can be saved with a limited impact on performance.

Chandrakasan [Chan96] 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 [Yang01] proposed a two-phase scheduling scheme which contains a design time scheduler and a runtime scheduler that minimizes energy consumption while meeting timing constraints. By choosing different scheduling options at compile time they achieved 20-40% average energy savings.

Zhang and Chen [Zhan02] proposed a priority based task mapping and scheduling for a fixed task graph applying the earliest deadline first scheduling and formulating the voltage scaling problem as an integer programming problem. They proved that their framework can slow 8% of cycles in very short time.

The main concern in DVS is to increase slack time utilization as much as possible and to make resultant energy consumption as low as possible. Two types of slack time are defined as worst slack time (WST) and workload-variation slack time (VST). WST results from processors utilization that is less than 100%. Low processor utilization is always the case even if all tasks exhibit their worst-case execution time. VST occurs due to execution time variations caused by data-dependent computation. WST can be estimated from the scheduling results before task execution whereas VST can be known only after actual task execution.

Shin et. al. [Shin01] proposed a low-energy, 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 to utilize both WST and WVST.

Shang et. al. [Shan03] 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. Such mechanisms resulted in 46% average energy savings at the cost of 15.2% increase in network latency and 2.5% decrease in network throughput.

Theoretical investigations of speed scaling algorithms were initiated by Yao, Demers, and Shankar [Yao 95]. Yao et al. propose formulating speed scaling problems as scheduling problems. They assumed that each task has a release time when it arrives into the system, an amount of work that must be performed to complete the task and also a deadline that specifies the time by which the task should be completed. A schedule specifies which task to run at each time, and at what speed that task should be run. In some settings, for example, the playing of a video or other multimedia presentation, there may be natural deadlines for the various tasks imposed by the application. In other settings, the system may impose deadlines to better manage tasks or ensure a certain quality of service to each task. They studied the problem of minimizing the total energy used subject to the deadline feasibility constraints

Few other literatures have considered energy saving in addition to the performance. Lu, Benini and Micheli [Lu00] presented a greedy on-line scheduling algorithm to facilitate energy 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 energy savings of 33%.

Mishra et. al. [Mish03] proposed two novel techniques for energy management in distributed systems. The first is a static technique which uses a greedy algorithm to manage energy 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 energy management technique.

Shiue and Chakrabarti in [Shiu00] presented polynomial time algorithms for (i) resource-constrained scheduling and (ii) latency-constrained scheduling for the case when the resources operate at multiple voltages. Both scheduling schemes try to reduce the overall energy consumption. The resource-constrained scheduling scheme tries to balance the conflicting requirements of reducing the latency and maximally utilizing resources operating at reduced voltages. The latency-constrained scheduling scheme assigns as many nodes as possible to the resources operating at low voltages without violating the timing constraint.

Kirovski and M. Potkonjak [Kiro97] developed a system-level approach for energy minimization of cost-constrained hard real-time designs. The approach simultaneously optimizes all three degrees of freedom for energy minimization, namely switching activity, effective capacity and supply voltage.

Srivastava in [Sriv96] described three broad architectural approaches for energy efficient programmable computation: predictive shutdown, concurrency driven supply voltage reduction, and switching activity reduction. A significant reduction in energy consumption can be obtained by employing these architectural techniques. They have shown that an aggressive shut down strategy based on a predictive technique can reduce the energy consumption by a large factor compared to the straightforward conventional schemes where the energy decision is based solely on a predetermined idle time threshold.

An example where the idea embodied in their techniques can be applied is the combination of parallelism-driven voltage reduction with switching activity reduction to increase the energy efficient of memory operations when the access pattern is sequential in nature. Such sequential access patterns occur, for example, when fetching video data from a frame buffer memory or when writing a page of virtual memory back to disk. Instead of accessing data from memory in a serial fashion, several words can be read from memory and the memory can be clocked at a lower rate for the same throughput. If the serial implementation runs at a supply of 3V to meet a given throughput, then the parallel version can run at a supply voltage of 1.3 V while meeting throughput requirements.

Chaeseok and Ha. [ImHa04] proposed an energy efficient real-time multi-task scheduling by the use of buffers with DVS. They saved an average energy 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.

CHAPTER 3

DYNAMIC VOLTAGE SCALING

For most energy-conscious designs, a major source of energy savings is voltage scaling, which scales operating voltages of processors and corresponding maximum clock speeds. The dominant source of energy consumption in digital CMOS circuits is the dynamic energy dissipation *P*, characterized by

$$P \propto CV^2 f$$

where *C* is the effective switching capacitance, *V* is the supply voltage, and *f* is the clock speed [Burd96].

Since energy varies linearly with the clock speed and the square of the voltage, adjusting the voltage can result in significant energy reductions, at least in theory. However, reducing the supply voltage requires a corresponding decrease in clock speed and increase in task execution latency.

The settling time for a gate is proportional to the voltage; the lower the voltage drop across the gate, the longer the gate takes to stabilize. To lower the voltage and still operate correctly, the cycle time must be lowered first. When raising the clock rate, the voltage must be increased first. Given that the voltage and the cycle time of a chip could be adjusted together, it should be clear now that the lower-voltage, slower-clock chip will dissipate less energy per cycle. If the voltage level can be reduced linearly as the clock rate is reduced, then the energy savings per instruction will be proportional to the square of the voltage reduction. Of course, for a real chip it may not be possible to reduce the voltage linear with the clock reduction. However, if it is possible to reduce the voltage at all by running slower, then there will be a net energy savings per cycle.

Currently manufacturers do not test and rate their chips across a smooth range of voltages. However, some data is available for chips at a set of voltage levels. For example, a Motorola CMOS 6805 microcontroller is rated at 6 MHz at 5.0 Volts, 4.5 MHz at 3.3 Volts, and 3 MHz at 2.2 Volts. This is a close to linear relationship between voltage and clock rate. Thus there is seemingly no technical objection to designing a variable-voltage system provided that the input reference voltage to the processor's voltage regulator may be a digital word writable by the processor.

The other important factor is the time it takes to change the voltage. The main time-cost would be for the converter or regulator to ramp the supply voltage up or down. The ramping time is determined by the time constants of the converter. The frequency for voltage regulators is on the order of 200 KHz [Weis94], so we speculate that it will take a few tens of microseconds to boost the voltage on the chip. Moreover the CPU should be able to continue working during a voltage ramp and ramping should not have any substantial energy cost.

Finally, why run slower? Suppose a task has a deadline in 100 milliseconds, but it will only take 50 milliseconds of CPU time when running at full speed to complete. A normal system would run at full speed for 50 milliseconds and then idle for 50 milliseconds (assuming there were no other ready tasks) as in Figure 3.1. During the idle time the CPU can be stopped altogether by putting it into a mode that wakes up upon an interrupt, such as from a periodic clock or from an I/O completion.

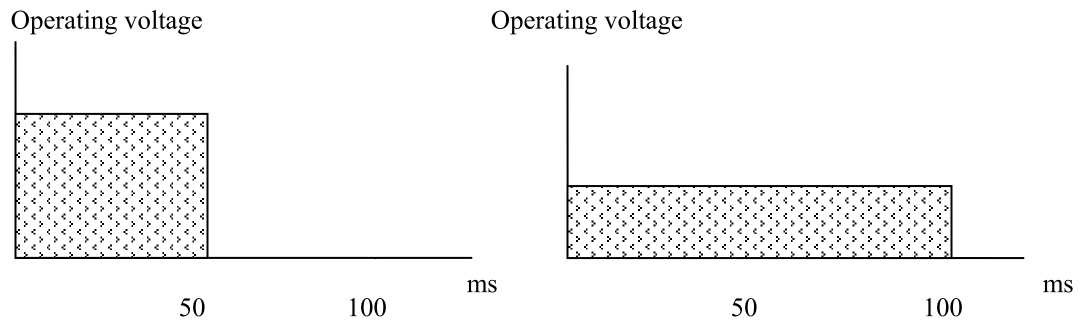
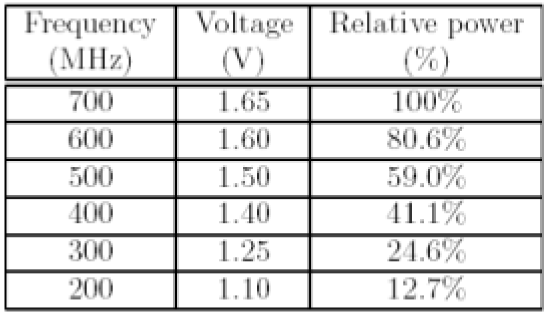


Figure 3.1. Different operating rates for the same task

Now, compare this to a system that runs the task at half speed so that it completes just before its deadline. If it can also reduce the voltage by half, then the task will consume 1/4 the energy of the normal system, even taking into account stopping the CPU during the idle time. This is because the same number of cycles is executed in both systems, but the modified system reduces energy use by reducing the operating voltage. Another way to view this is that idle time represents wasted energy, even if the CPU is stopped.

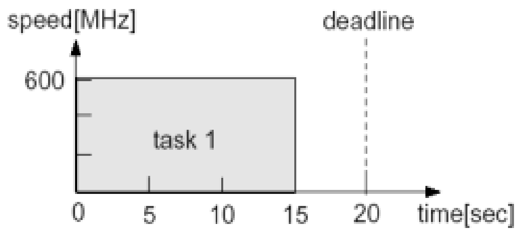
The relation between clock speed, supply voltage, and energy dissipation for Transmeta's Crusoe TM5400 microprocessor as reported in its data sheet [Tiwa96] are shown in Table 3.1. For a program running for time duration of *T*, its total energy consumption E is approximately equal to $E = P_{avg} \times T$, Where P_{avg} is the average energy consumed. Researchers have proposed many ways of determining "appropriate" operating clock rate. The basic idea behind these energy saving approaches is to slow down the tasks as much as possible without violating the deadline. This "just-in-time" strategy can be illustrated through a voltage scheduling graph as in Figure 3.2 [Much97].

In a voltage scheduling graph, the X-axis represents time and the Y-axis represents processor speed. The total amount of work for a task is defined by the area of the task "box." For example, task 1 in Figure 3.2 has a total workload of 8,000 cycles. By "stretching" it out all the way to the deadline without change of the area, we are able to decrease the CPU speed from 600MHz down to 400MHz. As a result, 23.4% of total (CPU) energy may be saved on a Crusoe TM5400 processor.

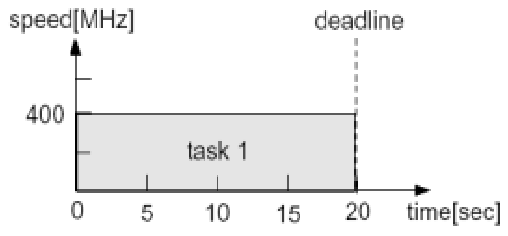


Frequency (MHz)	Voltage (V)	Relative power (%)
700	1.65	100%
600	1.60	80.6%
500	1.50	59.0%
400	1.40	41.1%
300	1.25	24.6%
200	1.10	12.7%

Table 3.1. The relationship between clock frequencies, supply voltage, and energy dissipation of Transmeta's Crusoe TM5400 microprocessor.



(a) Original schedule



(b) Voltage scaled schedule

Figure 3.2. Voltage scheduling graph

3.1 Power management implementation

There are several ways of achieving energy reduction. In this section, we study the various approaches.

3.1.1 Simple shutdown

Energy shutdown to a component is a radical solution that eliminates all sources if energy dissipation, including leakage.

Energy consumption of idle processors can be avoided by energying off the unit. This radical solution requires controllable switches. An advantage of this approach is the wide applicability to all kind of electronic components. A major disadvantage is the wake-up time or the recovery time because the processor operation must be reinitialized.

3.1.2 Multiple and variable energy supplies

Dynamic energy management is also applicable to processors that are not idle, but whose performance requirement varies with time. The implementation technology can then be based on the slowdown. The slowdown is achieved by lowering the voltage supply, such that the machine becomes performance critical.

Dynamically-varying supply voltages may be quantified [Chan96] and thus be restricted to a finite number of values, or may take values in a continuous range. In the former case it is possible to identify a finite number of energy states for the system.

3.1.3 OnNow approach

The OnNow approach uses energy-management hardware to put the PC into a low-energy sleep state instead of shutting down completely, so that the system can quickly resume working [Micr04]. While in the sleep state, the PC's processor is not executing code and thus no work is being accomplished for the user. However, events from both hardware devices (such as modem ring or network request) and the real-time clock can be enabled to cause the system to wake up.

Each device in the system has its own energy states, and these are independently managed by the device driver (or other policy owner) while the system is in the working state. The device's policy integrates any particular application's needs with device capabilities and other operating system information to conserve energy without adversely affecting the work that the user is doing.

CHAPTER 4

DIRECTED A-CYCLIC GRAPH DAG

We define a DAG as an a-cyclic graph with nodes representing tasks and edges representing execution precedence between tasks. A weight is associated to each node and edge. The node weight represents the task execution time and the edge weight represents the communication time between connected tasks. This communication time is zero if the tasks are executed on the same processor. Each DAG has a root node which is a node with no incoming edges and a sink node which is a node with no outgoing edges.

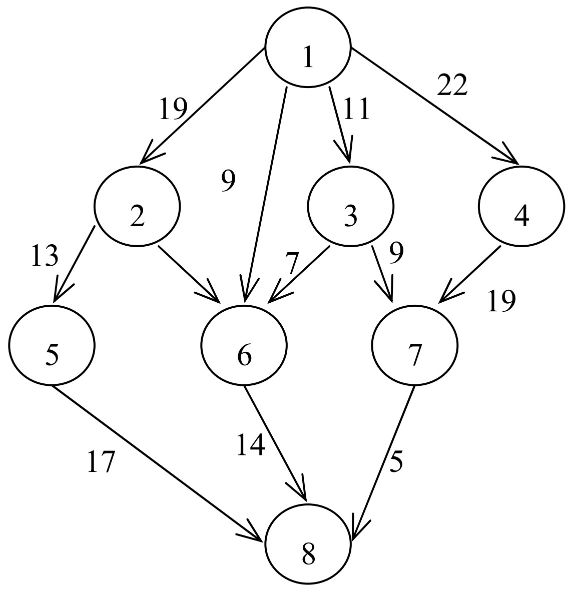


Figure 4.1. A sample DAG

The DAG structure occurs in many regular and irregular applications such as Cholesky factorization, LU decomposition, Gaussian elimination, FFT, Laplace transforms, and instruction level parallelism.

Along the lines of [Bask03] a DAG is represented by the tuple *G=(V, E, M, T, C, P)* where:

- *V* is the set of *n* nodes.
- *E* is the set of *e* edges between the nodes.
- *M* is a set of *m* machines or processors.
- *E(n, c)* is an edge between nodes *n* and *c*.
- *T* is the set of costs *T(n,k)*, represents the computational time of task *n* on machine *k*.
- *C* is the set of costs *C(n,c)*, which represents the communication cost associated with the edges *E(n,c).* Since intra-processor communication is insignificant compared to inter-processor communication, *C(n,c)* is considered to be zero if *n* and *c* are executed on the same processor.
- *P* is the set of costs *P(n,k),* which presents the consumed power 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.

Node *n* is a *predecessor* of node *c* if there is a directed edge originating from *n* and ending at *c*. In figure 4.1, node 1 is a *predecessor* of nodes 2, 3, 4 and 7.

Likewise, node *s* is a *successor* of node *n* if there is a directed edge originating from *n* and ending at *s*. From figure 4.1, node 6 is a *successor* of nodes 1, 2, and 3. We can further define *pred(n)* as the set of all predecessors of *n* and *succ(n)* as the set of all successors of *n* as an example *pred(6)={1,2,3}* and *succ(6)={8}*.

An *ancestor* of node *n* is any node *c* that is contained in *pred(n)*, or any node *a*, that is also an ancestor of any node *c* contained in *pred(n)*.

The earliest execution start time of node *n* on processor *k* is represented as *EST(n,k)*. Likewise, the earliest execution finish time of node *n* on processor *k* is represented as *EFT(n,k)*. *EST(n)* and *EFT(n)* represent the earliest start time upon any processor and the earliest finish time upon any processor, respectively. R_{nk} is defined as the earliest time that processor *k* will be ready to begin executing task *n*. We can mathematically define these terms as follows:

$EST(n, k) = max\{ R_{nk} , EFT(c, m)+C(c, n)\}$, Where, $c \in pred(n)$

$EFT(c, m) = EFT(c)$ and $C(c, n) = 0$ when $k = m$,

$EFT(n, k) = T (n, k)+EST(n, k)$,

$EST(n) = min (EST(n, k), k \in M)$,

$EFT(n) = min (EFT(n, k), k \in M)$.

The maximum clause finds the latest time that a predecessor's data will arrive at processor *k*. If the predecessor finishes earlier on a processor other than *k*, communication cost must also be included in this time. In other words, the earliest start time of any task *n* on processor *k, EST(n,k)* is the maximum of times at which processor *k* becomes available and the time at which the last message arrives from any of the predecessors of *n*.

The main goal for any scheduling technique is to minimize the *makespan* of the DAG. The *makespan* is defined as the time at which all nodes finish executing. In our case, the *makespan* will be equal to *EFT(y),* where *y* is the exit node in the graph. From Figure 4.1 the *makespan* is *EFT(8).*

The *critical path (CP)* is the longest path from an entry node to an exit node. The *critical path excluding communication cost (CPEC)* is the longest path from an entry node to an exit node, not including the communication cost of any edge traversed. In our work we assume that each task's mean execution cost across all processors is used to calculate *CP* while each task's minimum execution cost from any processor is used to calculate the *CPEC*.

The *top distance* for a given node is the longest distance from an entry node to the node, excluding the computation cost of the node itself. The *bottom distance* for a given node is the longest distance from the node to an exit node. Again we assume that each task's mean execution cost is used to calculate the *top distance* and *bottom distance*. The bottom distance is also referred to as the *upper rank* or the *blevel*.

The *Decisive Path (DP)* is defined as the *top distance* of a given node plus the *bottom distance* of the node. The *DP* is defined for every node in the DAG. The critical path, *CP,* then becomes the largest *DP* for an exit node.

4.1 Performance

Our algorithms are for scheduling directed acyclic weighted task graph running on a bounded number of heterogeneous processors with the twin objectives of minimizing the amount of energy consumed and minimizing the finish time.

In other words, tasks arrive with given execution time and need to meet certain execution deadlines as well as minimize the consumed energy.

Using simulations, we evaluated the performance of EADAGS and EAGS-D. The first test suite consists of random directed a-cyclic graphs. The input parameters used to generate the graphs were:

- Number of nodes (tasks) in the graph, *n*.
- Shape parameter of the graph, α. If $\alpha = 1.0$, the graph is balanced. A DAG with high parallelism can be generated by selecting $\alpha >> 1$. Whereas $\alpha << 1$ 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*. *CCR* is the ratio of the average communication to average computation cost. If a DAG's *CCR* is less than 1, it is a computation-intensive application; if it's CCR 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, the parameters were varied as follows:

n= {10, 20, 40, 60, 80, 100, 500, 1000}

CCR = {0.1, 0.5, 1, 5, 10}

α = {0.5, 1, 2}

Out-degree = {1, 2, 3, 4, 5, 100}

β = {0.1, 0.25, 0.5, 0.75, 1.0}

PNR = {25%, 50%, 100%}

These values produced 10,800 DAGs, which were repeated for both presented algorithms, EADAGS and EAGS-D.

The second test suite to evaluate the performance of EADAGS and EAGS-D used task graphs of real world problems, specifically Gaussian elimination [Wu90], Molecular dynamic code [Chun92], Sieve of Eratosthenes, and Fast Fourier Transform (FFT). For these problems, the shape of the DAG is fixed and the only parameters we changed were the number of available processors and *CCR*.

Processors were assumed to have three different operating voltage levels and five operating strategies 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 5V when using this voltage if the processor becomes idle, it shuts down. We will refer to this operating voltage as 5V/off. This level is used for reference only since it has physical limitations. The other two operating voltages are 2V and 3.3V, which slow the processor during task execution and during processor’s idle times.

CHAPTER 5

SCHEDULING ALGORITHIMS

Many scheduling algorithms for scheduling DAGs in a distributed computing environment have been proposed as has been mentioned in Chapter 2. However, the problem of discovering the schedule that gives the minimum finish time is NP-Complete. Among all these scheduling algorithms, some prove to perform better in term of finish time or *makespan*.

In this work two new scheduling algorithms for scheduling DAGs on distributed computing systems were presented: Energy Aware DAG Scheduling (EADAGS) and Energy Aware Graph Scheduling with Duplication (EAGS-D).

5.1 EADAGS algorithm

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 [Park97], 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:

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

$$P_1(k) = \frac{fV_1^2}{2}$$

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 $E_2 = \Sigma E_2(k)$ 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 non blocking 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 5.1.

Table 5.1. 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
```

Figure 5.1 lists the notations used by EADAGS and procedures of EADAGS (EADAGS, *AddQ, StartTime, and ScaledEnergy*).

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* 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
- $EFT(i, k)$ represent the earliest finish time of node $i \in G$ on machine k
- $EFT(i)$ represent the scheduled finish time of node $i \in G$
- $EFT_2(i)$ represent the finish time for node i with the scaled down voltage
- $EST(i)$ 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

Figure 5.1.a. Notations for EADAGS

```
EADAGS
    Transform G to a DAG with a single entry node and a single exit node
    Compute DP for each node $n \in G$                          // $O(n^2)$
    // DP of the exit node is the critical path, CP
    // Fill ScheduleQ with nodes in CP in increasing top-distance.
    $ScheduleQ \leftarrow \Phi$
    for each node $n \in CP$ do                                  // $O(n^2)$
        // Traverse in increasing top-distance.
        $ScheduleQ = addQ(n)$
    end for
    //Schedule nodes in ScheduleQ to processors
    while $ScheduleQ \neq \Phi$ do                               // $O(n)$
        Pick the head node i in ScheduleQ
        for each processor $k \in M$ do                          // $O(m)$
            $s_{ik} = StartTime(i, k)$
            $EFT(i, k) = s_{ik} + T(i, k)$
        end for
        $EFT(i) = \min_{k \in M} EFT(i, k)$
        Schedule i on processor $p \in M$ that gave minimum earliest finish time
        Remove i from ScheduleQ
    end while
    if $EFT(y) \geq \min_{k \in M} \sum_{i \in G} T(i,k)$
        Schedule all nodes on the processor $p \in M$, which provided the minimum
    end if
    $T \leftarrow EFT(y)$                // makespan
    $E_1 = \frac{fTV_1^2}{2}$
    $E_2 = ScaledEnergy()$
end EADAGS
```

Figure 5.1.b. *EADAGS* procedure

```
addQ(n)
// Adds parents of node $n \in G$ and $n$ to ScheduleQ
// Returns ScheduleQ
    for each parent $b$ of $n$ not visited        // in decreasing DP
        addQ(b)
    end for
        Add to $n$ to ScheduleQ and mark it visited
    return ScheduleQ
end addQ
```

Figure 5.1.c. *addQ* procedure

```
StartTime (node n, machine k)
// Returns the earliest available start time of node $n \in G$ on machine $k \in M$
    $s_{nk} \leftarrow R_k$
    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
```

Figure 5.1.d. *StartTime* procedure

```
ScaledEnergy( )
// Returns the total amount of energy consumption after applying voltage scaling
        for each processor $k \in M$ do                                  // $O(m)$
            for each node $n \in G$ scheduled on $k$ do   //$O(n)$ from first to last scheduled
                $T_2(n) = \frac{V_1^2}{V_2^2} \times T_1(n)$
                $EFT_2(n) = s_{nk} + T_2(n)$
                if ($EFT_2(n)+C(n,c) < \min EST(c)$) for each $c \in succ(n)$ then
                // if $c$ is scheduled on $k$, $C(n, c) = 0$
                    $EFT(n) = EFT_2(n)$      //update $EFT(n)$
                    Let $i$ be the node scheduled immediately after $n$ on $k$
                    // use scaled voltage, see Figure 5.2
                    $E(k, n)=T_2(n) \times V_2^2+(EST(i)-EFT(n)) \times V_2^2$
                else
                //original voltage for execution and scaled voltage during idle time,
                // see Figure 5.2
                    $E(k, n)=T_1(n) \times V_1^2+(EST(i)-EFT(n)) \times V_2^2$
                end if
            end for
                $E_2(k)=+E(k, n)$
        end for
            $E_2 = \sum_{k \in M} E_2(k)$
        return $E_2$
end ScaledEnergy
```

Figure 5.1.e. *ScaledEnergy* procedure

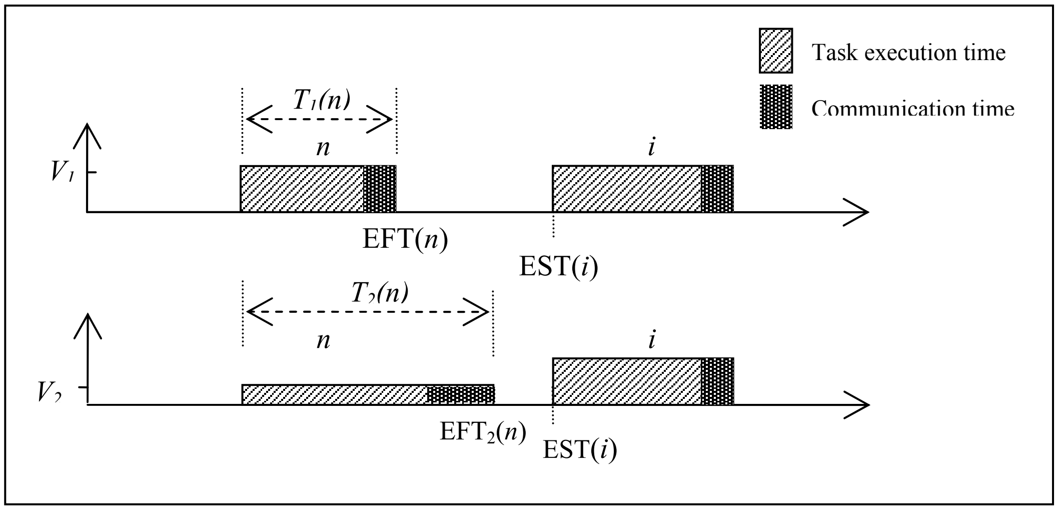


Figure 5.2. Timeline for machine *k*

5.2 EAGS-D Algorithm

Energy Aware Graph Scheduling with Duplication (EAGS-D) combines HNPD and dynamic voltage scaling. The Heterogeneous N-Predecessor Duplication (HNPD) algorithm combines the techniques of *insertion-based* list scheduling with multiple *task duplication* to minimize schedule length. The performance of HNPD was proven to outperform two of the best existing heuristics, Heterogeneous Earliest Finish Time (HEFT) and Static Task Duplication Scheduling (STDS), in terms of finish time and the number of processors employed over a wide range of parameters [Bask03]. EAGS-D works as follows: EAGS-D assigns tasks to the best available processor according to the order of tasks in a scheduling queue called *ScheduleQ*. EAGS-D assigns highest priority to critical path nodes (CPN) and then to those predecessors of CPNs that include the CPN in their *DP*. Among these predecessors it gives higher priority to nodes with higher *DP* values. This is because the nodes with the higher *DP* values are likely to be on longer paths. In the order of tasks in *ScheduleQ*, EAGS-D uses *EFT(n)* to select the processor for each task *n*. As it is an insertion-based algorithm, it calculates ready time of any machine $k \in M$, R_k to be the earliest idle time slot large enough to execute *T(n, k)*. In other words, it looks for a possible insertion between two already scheduled tasks on the given processor without violating precedence relationships. Once tasks have been assigned to processors, it attempts to duplicate predecessors of the tasks. Predecessors are selected for duplication from most favorite to least and by descending *top distance*. The goal of duplicating predecessors is to decrease the length of time for which the node is awaiting data by making use of the processor's slack time. Predecessors of node *n* are duplicated on processor *k*, on which node *n* is scheduled on. If the duplication result in a lower finish time, duplication is retained; otherwise it is discarded. If there is idle time between the recently assigned task *n* and the preceding task on processor *k*, EAGS-D attempts to duplicate each predecessor *j*. If *j* is not already scheduled on processor *k*, it is duplicated if *EFT(j,k)* is less than *EFT(j)+C(j, n)*. The duplication is retained if *EFT(n,k)* decreases. Otherwise, it is discarded. The same duplication procedure is repeated for each predecessor in order of most favorite to least. After EAGS-D attempts to duplicate each predecessor, it recursively attempts to duplicate the predecessors of any duplicated tasks. Duplication recursively continues until no further duplication is possible.

Then, EAGS-D computes the consumed energy. The total energy consumed when no voltage scaling is used, E_1 is first calculated by the following equations:

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

$$P_1(k) = \frac{fV_1^2}{2}$$

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 EAGS-D, 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 $E_2 = \Sigma E_2(k)$ where:

- $E_2(k) = \dfrac{T_1 f_1 V_1^2 + T_2 f_2 V_2^2}{2}$ 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 non blocking 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 = $\dfrac{E_1 - E_2}{E_1} \times 100$. A high level description of EAGS-D appears in Table 5.2.

Table 5.2. EAGS-D algorithm

```
Let G represent a DAG
Let M be the set of m processors in the system

EAGS-D
    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
        Duplicate predecessors of i on p if doing so results in a shorter schedule
        //duplicate executions performed within slacks of the schedule of p
    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 EAGS-D

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
```

Figure 5.3.a lists the notations used by EAGS-D while Figures 5.3.b, 5.3.c, 5.3.d, 5.3.e, 5.3.f and 5.3.g shows the detailed algorithm procedures: EAGS-D, *AddQ, StartTime, DuplicatePred, Duplicate, and ScaledEnergy* respectively.

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* 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
- $EFT(i, k)$ represent the earliest finish time of node $i \in G$ on machine k
- $EFT(i)$ represent the scheduled finish time of node $i \in G$
- $EFT_2(i)$ represent the finish time for node i with the scaled down voltage
- $EST(i)$ 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

Figure 5.3.a. EAGS-D notations

```
EAGS-D
    Transform G to a DAG with a single entry node and a single exit node
    Compute DP for each node $n \in G$                               // $O(n^2)$
    // DP of the exit node is the critical path, CP
    // Fill ScheduleQ with nodes in CP in increasing top-distance.
    $ScheduleQ \leftarrow \Phi$
    for each node $n \in CP$ do                                      // $O(n^2)$
        // Traverse in increasing top-distance.
        $ScheduleQ = addQ(n)$
    end for
    //Schedule nodes in ScheduleQ to processors
    while $ScheduleQ \neq \Phi$ do                                   // $O(n)$
        Pick the head node i in ScheduleQ
        for each processor $k \in M$ do                              // $O(m)$
            $s_{ik} = StartTime\ (i, k)$
            $EFT(i, k) = s_{ik} + T(i, k)$
        end for
        $EFT(i) = \min_{k \in M} EFT(i, k)$
        Schedule i on processor $p \in M$ that gave minimum earliest finish time
        Remove i from ScheduleQ
        // Duplicate predecessors of i on p if it results in a shorter schedule
        $DuplicatePred\ (i, pred(i), p)$
    end while
    if $EFT(y) \geq \min_{k \in M} \sum_{i \in G} T(i,k)$
        Schedule all nodes on the processor $p \in M$, which provided the minimum
    end if
    $T \leftarrow EFT(y)$        // makespan
    $E_1 = \frac{fTV_1^2}{2}$
    $E_2 = ScaledEnergy(\,)$

end EAGS-D
```

Figure 5.3.b. EAGS-D procedure

```
addQ(n)
// Adds parents of node $n \in G$ and $n$ to ScheduleQ
// Returns ScheduleQ
        for each parent $b$ of $n$ not visited      // in decreasing DP
                addQ(b)
        end for
                Add to $n$ to ScheduleQ and mark it visited
        return ScheduleQ
end addQ
```

Figure 5.3.c. *addQ* procedure

```
StartTime (node n, machine k)
// Returns the earliest available start time of node $n \in G$ on machine $k \in M$
        $s_{nk} \leftarrow R_k$
        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
```

Figure 5.3.d. *StartTime* procedure

```
DuplicatePred (node n,  pred(n), machine k)
        for each $q \in pred(n)$                   // in order from most favorite to least
                Duplicate (n, q, k)
                if $q$ was duplicated then
                        DuplicatePred (q, pred(q), k)
                end if
        end for
end DuplicatePred
```

Figure 5.3.e. *DuplicatePred* procedure

```
Duplicate (node n, node i, machine k)
// Attempts to duplicate node $n$, a predecessor of node $i$, on machine $k$ which executes node $i$
    if $n$ has not been scheduled on $k$ then
        $r_{nk} = StartTime(n, k)$
        // Attempt to insert node $n$ in the first available slack in the schedule of
        Processor $k \in M$
        for each node $j$ already scheduled on $k \in M$ do
        //scan from last to first scheduled
            if $j \notin pred(n)$ then
                if $j$ is the first node scheduled on $k$ then
                    if $EST(j, k) >= r_{nk} + T(n, k)$ then //see Figure 5.4
                        $s_{nk} = r_{nk}$
                        end if
                Let $p$ be the node immediately scheduled prior to $j$ on $k$
                else if $(EST(j, k)-EFT(p, k)>= T(n,k))$ and $(EFT(p, k)>= r_{nk})$ then
                    $s_{nk} = EFT(p, k)$                // see Figure 5.5
                end if
            $EFT(n, k) = s_{nk} + T(n, k)$
            end if
        end for
        //If $EFT(i)$ did not decrease due to duplication, discard duplication
        if $EFT(n ,k) < min_{succ(n)}(EFT(n)+C(n,succ(n)))$ then
            Insert and schedule $n$ on $k$
            Recalculate $EFT(i, k)$
            if $EFT(i ,k)$ improves then
                Keep $n$ scheduled on $k$
            else
                Discard the duplication of node $n$ on $k$
            end if
        end if
    end if
end Duplicate
```

Figure 5.3.f. *Duplicate* procedure

```
ScaledEnergy( )
// Returns the total amount of energy consumption after applying voltage scaling
    for each processor $k \in M$ do                                   // O(m)
        for each node $n \in G$ scheduled on $k$ do   //O(n) from first to last scheduled
            $T_2(n) = \frac{V_1^2}{V_2^2} \times T_1(n)$
            $EFT_2(n) = s_{nk} + T_2(n)$
            if ($EFT_2(n) + C(n, c) < \min EST(c)$) for each $c \in succ(n)$ then
            // if $c$ is scheduled on $k$, $C(n, c) = 0$
                $EFT(n) = EFT_2(n)$      //update $EFT(n)$
                Let $i$ be the node scheduled immediately after $n$ on $k$
                // use scaled voltage, see Figure 5.6
                $E(k, n) = T_2(n) \times V_2^2 + (EST(i) - EFT(n)) \times V_2^2$
            else
            //original voltage for execution and scaled voltage during idle time,
            // see Figure 5.6
                $E(k, n) = T_1(n) \times V_1^2 + (EST(i) - EFT(n)) \times V_2^2$
            end if
        end for
            $E_2(k) =+ E(k, n)$
    end for
        $E_2 = \sum_{k \in M} E_2(k)$
    return $E_2$
end ScaledEnergy
```

Figure 5.3.g. *ScaledEnergy* procedure

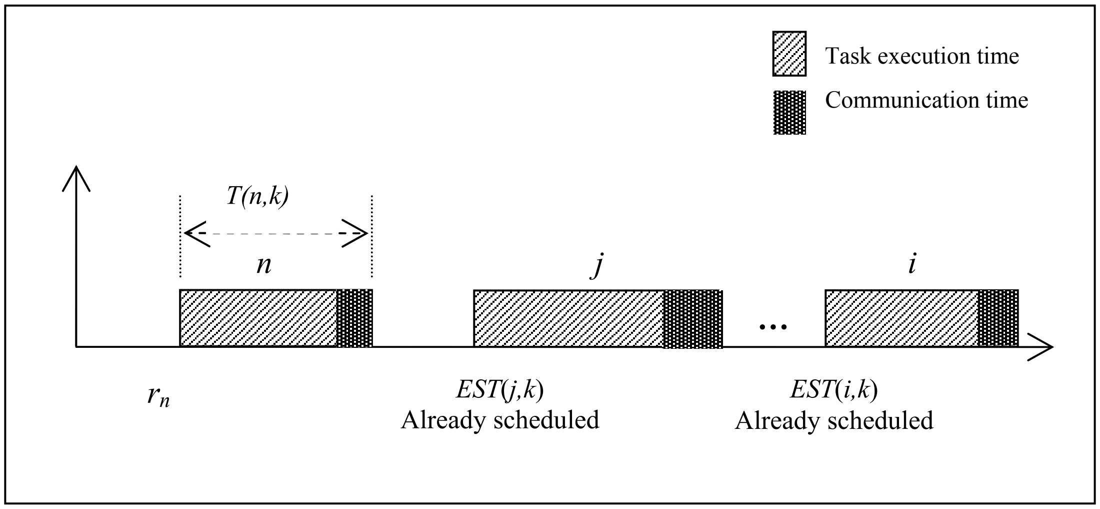


Figure 5.4. Timeline for machine *k* (case 1)

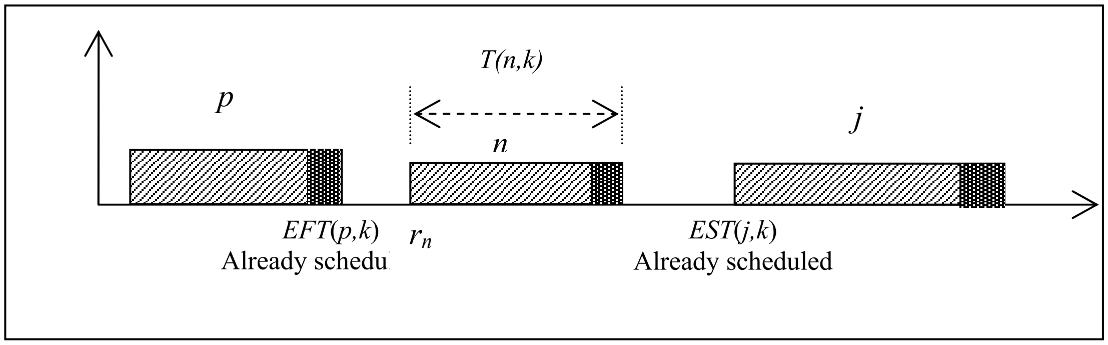


Figure 5.5. Timeline for machine *k* (case 2)

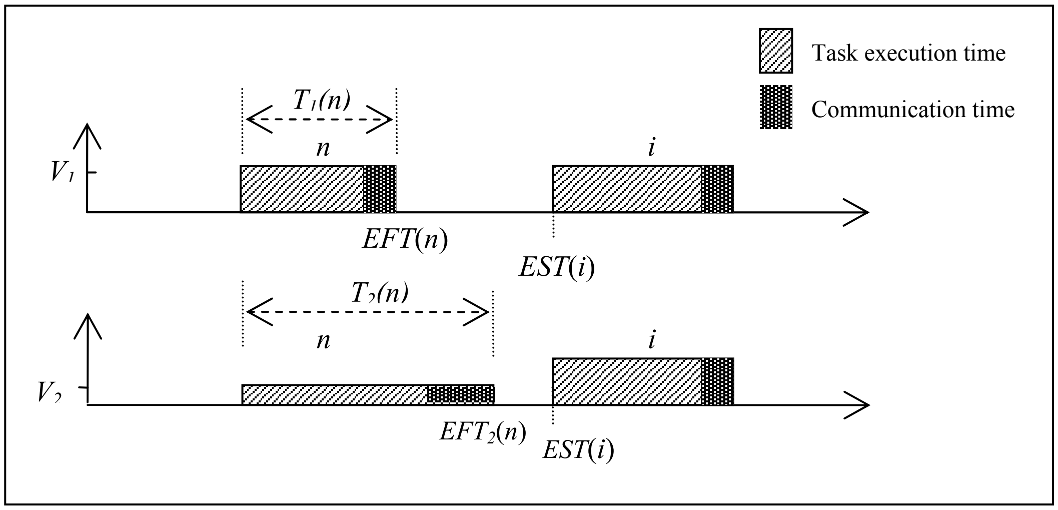


Figure 5.6. Timeline for machine *k*

5.3 Voltage scaling Strategies

Executing any task with a slower rate will require longer time to finish. And since we are dealing with DAGs in which tasks are depending on other tasks. Changing the finish time of a task may affect the start time of its predecessors resulting in a change of the schedule *makespan*.

In our algorithm we want to make use of processor's slack time to save energy without changing the *makespan* of the original scheduling algorithms. 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. We tested for five different strategies:

1. 5V/off: execute tasks with regular voltage level and turn off processors during their idle time. This technique is used for comparison reasons only since turning processors off each time they incur slack time has physical limitations.
2. 2V during idle: involves executing tasks with regular execution rate and lowering the processor's voltage level to 2V only during its idle time.
3. 3.3V during idle: involves executing tasks with regular execution rate and lowering the processor's voltage level to 3.3V during all its idle time.
4. 2V scale: we check if the slack time is long enough to scale down the execution rate of tasks to 2V without altering the finish time. If so, execute the task with 2V rate and also lower the voltage during the remaining idle time. If not, execute task with regular voltage rate and lower the voltage rate during the idle time.
5. 3.3V scale: we check if the idle time is long enough to lower the execution rate to 3.3V of tasks without affecting the *makespan*. If so, execute the task with 3.3V rate and also lower the voltage during the remaining idle time. If not, execute task with regular voltage rate and lower the voltage rate during the idle time.

6.3.1. Voltage scaling algorithm

The task in Figure 5.7 has finish time of t_1 when executed with V_1 and a finish time of t_2 when executed with V_2. This task can be executed with the lower voltage level only if $t_2 <$ min (start time of all its children).

For the task in Figure 5.7(a), the execution rate can be slowed down to V_2 and reduce the amount of energy consumed, while the task in Figure 5.7(b) cannot be scaled down to V_2 since the new finish time is greater than the earliest children's start time. So tasks need to be executed with the original voltage level. The voltage for the processors will be scaled down during idle time x only.

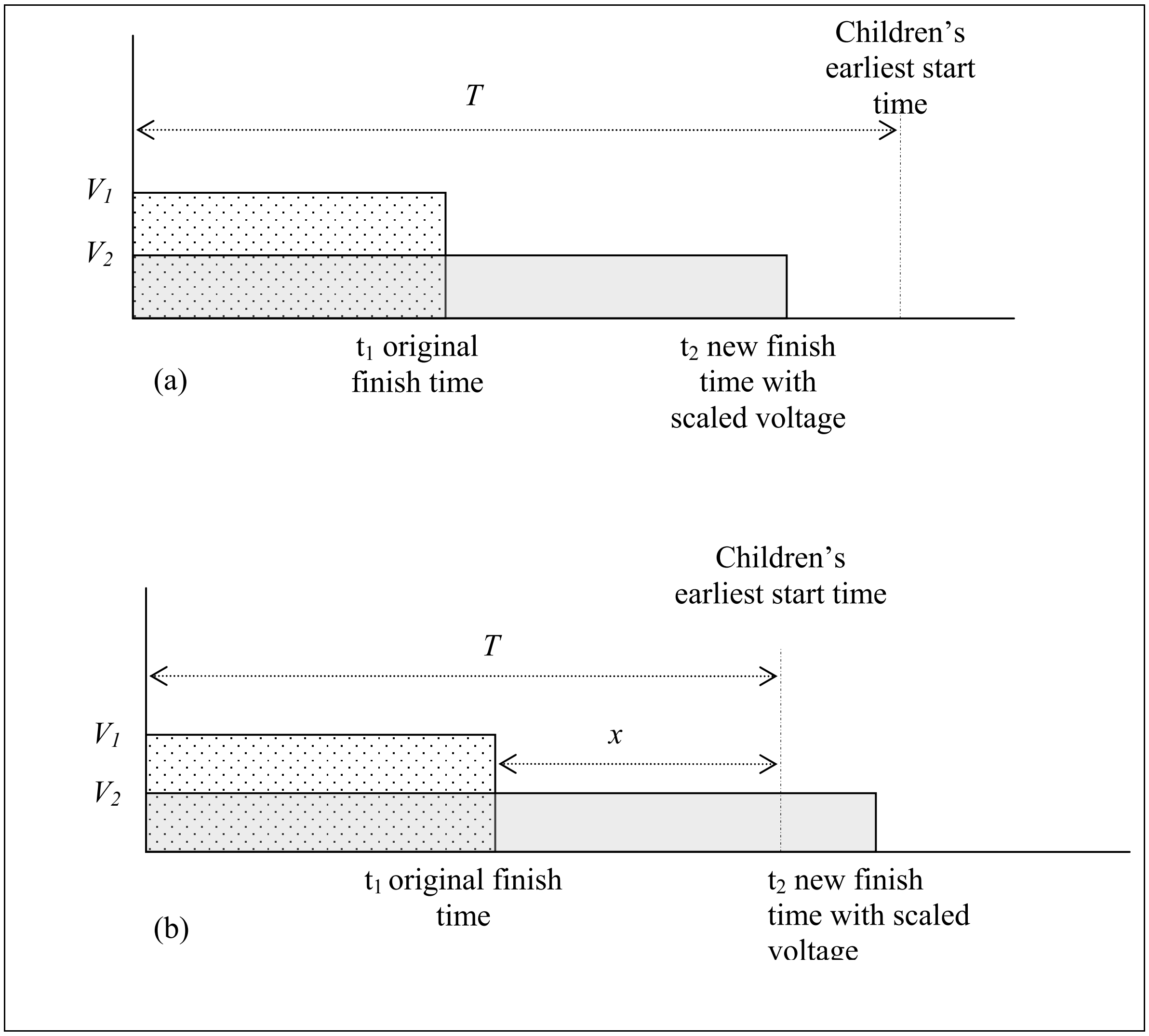

Children's earliest start time
T
V_1
V_2
t_1 original finish time
t_2 new finish time with scaled voltage
(a)
Children's earliest start time
T
V_1
V_2
x
t_1 original finish time
t_2 new finish time with scaled voltage
(b)

Figure 5.7. Different finish times with different execution rates

Although we choose to do the energy scaling as a second phase after the scheduling of all tasks, it could have been done in parallel with the scheduling phase resulting in the same schedule and the same amount of energy savings.

This could be explained as follows: each task is scheduled on the processor that results in the earliest finish time. The finish time of each node on each processor is determined by the first available start time of that node on that processor and the execution time of the node on that specific processor, taking into consideration the finish time of its predecessors and all communication cost between the task and any other node. Then its successors are scheduled based on the same criteria. So immediately after the scheduling of all the task children on the appropriate processors, the decision for scaling down this task can be made based on the finish time of the task and the start time of its successors. A task can be scaled down only if its finish time after executing with a lower voltage level is earlier than the start time of all its successors. Executing the task with the scaled down voltage level would not affect the scheduling decision for the next nodes since it only affect the finish time of the scaled down task.

The advantage of doing it as a second phase is that we do not have to check if all the task's successors have been scheduled before deciding whether to scale that task down or not resulting in a lower complexity algorithm.

CHAPTER 6

RESULTS AND DISCUSSION

The experimental simulation estimates the energy gains of using voltage scaling techniques on scheduling DAG tasks on multiple processors on top of two of the best known high performing scheduling algorithms, DPS and HNPD.

As been stated in Chapter 5, 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 [Weis94]. First operating voltage was 5V; when using this voltage, if the processor becomes idle, it shuts down. We will refer to this operating voltage as 5V/off. This level is used for reference only since it has physical limitations. The other two operating voltages are 2V and 3.3V, which slow the processor during task execution.

The algorithms were tested using random DAGs and DAG for real world problems; more specifically we tested on DAGs for Molecular Dynamics code, Gaussian elimination, Sieve of Eratosthenes, and Fast Fourier Transform.

6.1 Results for random DAGs

A set of random generated DAGs were generated to evaluate the voltage scaling technique. Those DAGs had different input parameters (number of nodes, communication to computation ratio, shape parameter, out-degree, and computation range) varied as follows:

n = {10, 20, 40, 60, 80, 100, 500, 1000}

CCR = {0.1, 0.5, 1, 5, 10}

α = {0.5, 1, 2}

out-degree = {1, 2, 3, 4, 5, 100}

β = {0.1, 0.25, 0.5, 0.75, 1.0}

PNR = {25%, 50%, 100%}

The above values produce 10,800 DAGs, which were repeated for both presented scheduling algorithms, EADAGS and EAGS-D.

6.1.1 Results for EADAGS

The amount of energy consumed was measured for DPS and EADAGS for different test sets of 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 amount of energy consumed were measured for five different strategies; 5V/off, 2V during idle, 3.3V during idle, 2V scale and 3.3V scale (explained in Chapter 5).

The first test set was achieved by combining DAGs with respect to number of nodes. The energy saving was averaged over DAGs with varying *CCR*, *α, β*, *out-degree,* and *PNR*.

Figure 6.1 shows the average energy saved by EADAGS over DPS with respect to number of nodes. The percentage of energy savings increased with increasing number of nodes. Average energy savings were 30% for a DAG of 10 nodes and the savings gradually increases to 46% for 1000 node DAGs. Larger numbers of nodes increase the chance of dependency between tasks, causing a slight increase in the processor wait time and accordingly increasing the energy savings.

Table 6.1 lists the *makespan* and the average energy savings with respect to number of nodes. Average energy savings ranges between 30% and 46% for the 5V/off technique, 26% and 46% when 2V during idle is used, 24% and 46% when using 3.3V during idle, 29% and 46 % when 2V scale is used, and 28% and 46% when 3.3V scale is used.

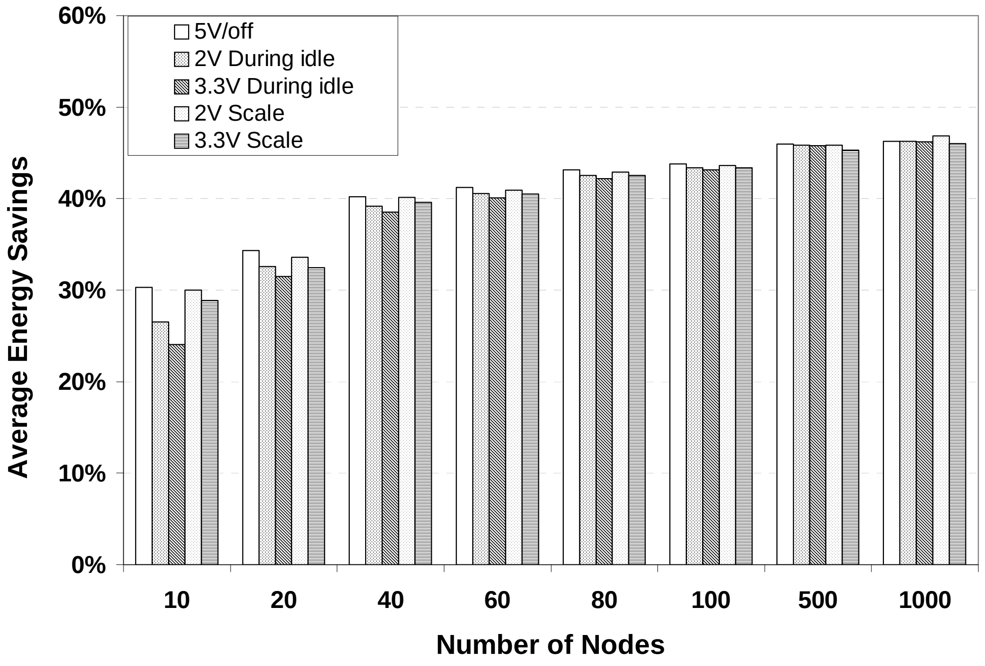


Figure 6.1. Average energy savings with respect to number of nodes

Number of nodes	*Makespan*	Percentage of energy savings				
		5V/off	2V during idle	3.3V during idle	2V Scale	3.3V Scale
10	174.7	30.28%	26.53%	24.09%	29.98%	28.84%
20	175.27	34.32%	32.6%	31.48%	33.63%	32.45%
40	321.67	40.2%	39.18%	38.51%	40.12%	39.57%
60	404.36	41.2%	40.54%	40.11%	40.93%	40.47%
80	525.5	43.14%	42.56%	42.19%	42.91%	42.51%
100	674.65	43.8%	43.39%	43.12%	43.64%	43.35%
500	835.29	45.94%	45.85%	45.8%	45.82%	45.28%
1000	1127	46.29%	46.25%	46.22%	46.88%	46.03%

Table 6.1. *Makespan* and average energy savings with respect to number of nodes

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

In Figure 6.2 the average energy savings has been plotted with respect to *CCR*. The average energy savings increased with increasing *CCR*. When *CCR* increases, processors incur longer idle times due to communication between tasks. Our algorithm was able to use such idle times to achieve energy savings.

Table 6.2 shows the *makespan* and the average energy savings for five different *CCR* values. The average energy savings over DPS ranges from 35% for *CCR*=0.1 to 51% when *CCR* = 10 for 5V/off technique. Savings are smaller for 2V during idle; they range between 25% and 38%. Savings are even smaller for the 3.3V during idle time; they are 19% for *CCR* = 0.1 and 30% for *CCR* = 10. Average energy savings are 28% for *CCR* = 0.1 and 44.7% when *CCR* = 10 for 2V scale, while for 3.3V scale energy savings is 22.8% for *CCR* = 0.1 and 40% when *CCR* = 10.

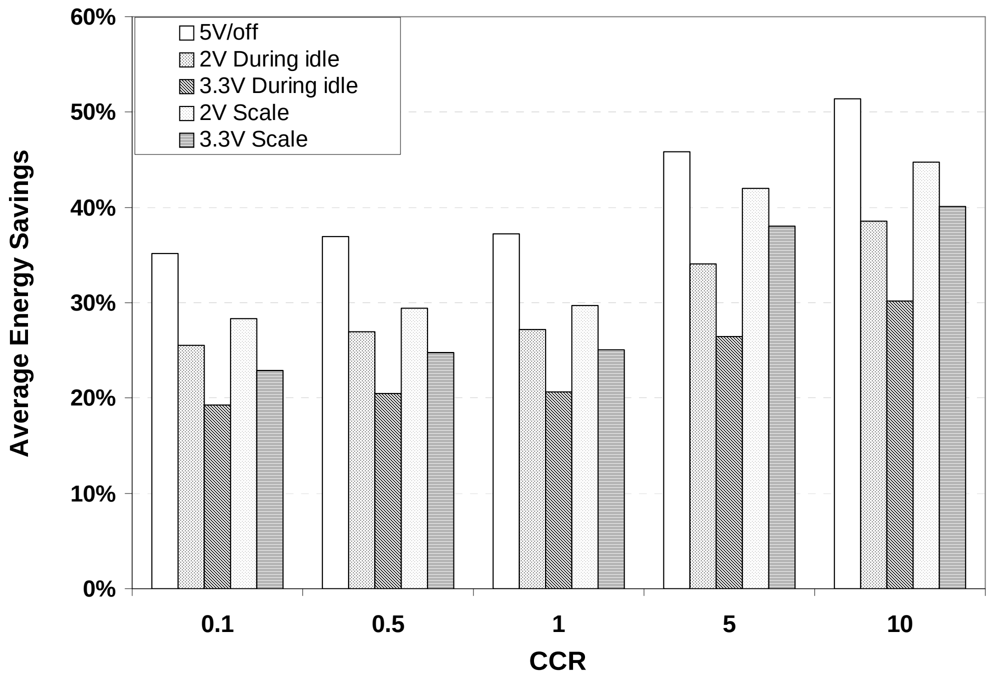


Figure 6.2. Average energy savings with respect to *CCR*

CCR	*Makespan*	Percentage of energy savings				
		5V/off	2V During idle	3.3V during idle	2V Scale	3.3V Scale
0.1	117	35.17%	25.53%	19.27%	28.33%	22.87%
0.5	124	36.93%	26.95%	20.45%	29.42%	24.77%
1	133.89	37.25%	27.2%	20.66%	29.69%	25.06%
5	209.89	45.84%	34.07%	26.42%	41.98%	38.05%
10	280	51.4%	38.52%	30.15%	44.73%	40.11%

Table 6.2. *Makespan* and average energy savings for different *CCR* values

The third test set combines DAGs with respect to processor to node ratio, *PNR*. The average energy savings is averaged over randomly generated DAGs with varying *n*, *CCR*, *α, β*, and *out-degree*. Figure 6.3 shows the average energy saving with respect to *PNR*. Figure 6.3 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. The average energy savings measured was 47% for *PNR* = 25% in the 5V/off technique, 41% when processors operate at 2V during idle, 38% if they operate at 3.3V when idle, 42% when 2V scale is used, and 39% when 3.3V scale is used. But for *PNR* = 100% the average energy savings for all three operating levels are almost equal to 33% , which is due to the significant decrease in the processor's wait time.

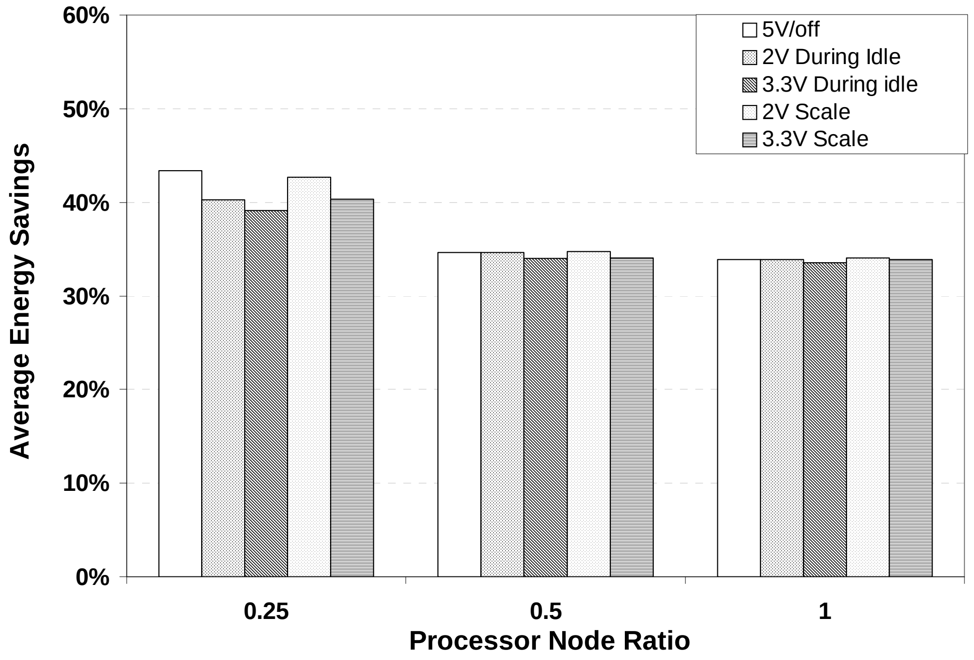


Figure 6.3. Average energy savings with respect to *PNR*

Another test set combined DAGs with respect to shape parameter *α.* The average energy savings were computed over 36,000 randomly generated DAGs with varying n, *PNR*, *CCR*, *β*, and *out-degree*.

In Figure 6.4 the average energy savings for the five operating strategies were plotted with respect to *α.* The results in this figure indicate that the overall energy savings marginally increased with increasing α. Increasing α increases parallelism in the DAG, resulting in more idle time for the processors due to the task dependency. This time can be used to reduce the consumed energy. The average energy savings was measured as 24% for $\alpha = 0.5$, 25% for $\alpha = 1$ and 25.5% when $\alpha = 2$.

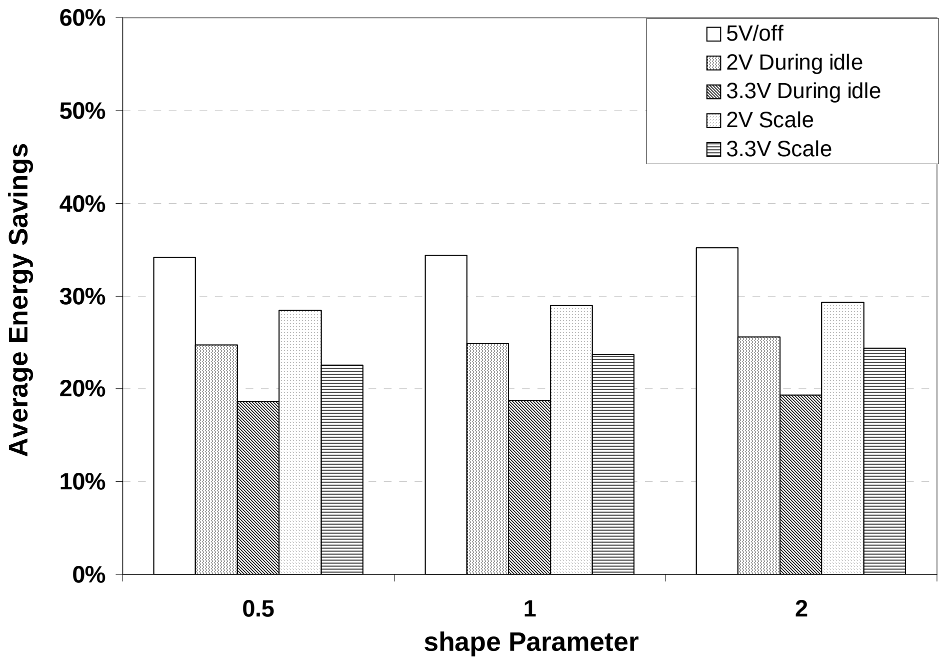


Figure 6.4. Average energy savings with respect to *α*

Shape parameter	*Makespan*	Percentage of energy savings				
		5V/off	2V during idle	3.3V during idle	2V scale	3.3V Scale
0.5	110.87	34.19%	24.75%	18.62%	28.47%	22.54%
1	189.87	34.38%	24.9%	18.74%	29.01%	23.71%
2	219.4	35.22%	25.58%	19.31%	29.33%	24.37%

Table 6.3. *Makespan* and average energy savings for different shape parameters

The last test set was for *out-degree*. The average energy savings were averaged from 21,600 randomly generated DAGs with varying *n*, *α, β*, *CCR*, *PNR*.

Figure 6.5 shows the average energy savings with respect to five values for *out-degree*. The results in this figure indicate that increase in *out-degree* results in smaller average energy savings. A larger *out-degree* allows many processors to run in parallel.

Table 6.4 lists the *makespan* and average energy savings with respect to *out-degree*. The amount of energy that could be saved ranges between 44% and 13% for processors using 5V/off, 32% and 8.5% for 2V during idle, 25% and 5% for 3.3V during idle, 36% and 10% when 2V scale is used, and 29% and 6% for a 3.3V scale.

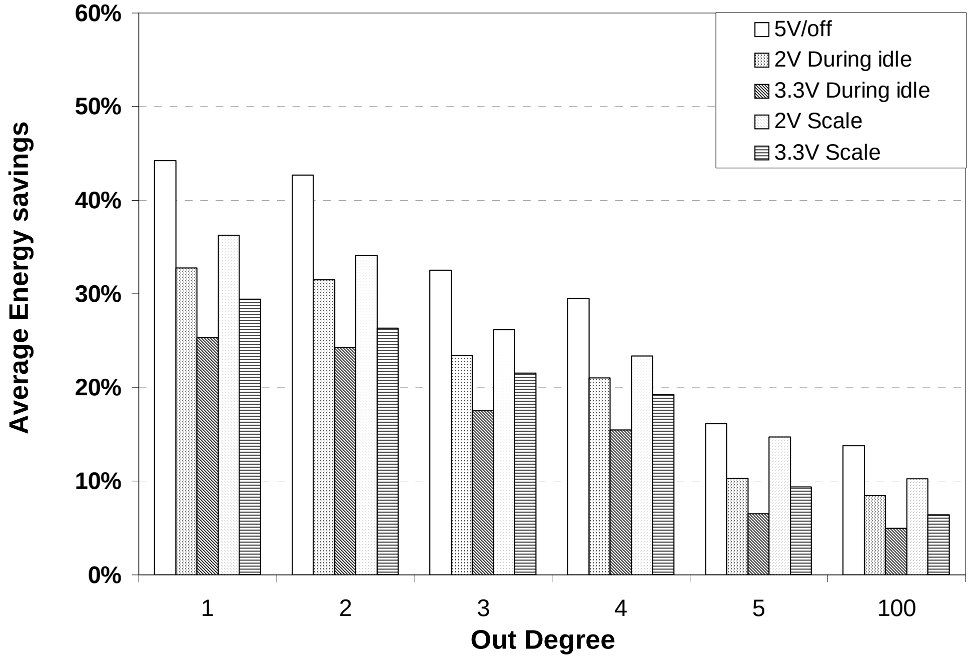


Figure 6.5. Average energy savings with respect to *out-degree*

Out-degree	*Makespan*	Percentage of energy savings				
		5V/off	2V during idle	3.3V during idle	2V scale	3.3V Scale
1	233.97	44.24%	32.79%	25.35%	36.29%	29.44%
2	321.67	42.67%	31.54%	24.3%	34.07%	26.36%
3	166.17	32.55%	23.44%	17.52%	26.18%	21.54%
4	194.29	29.53%	21.02%	15.5%	23.37%	19.28%
5	203.6	16.17%	10.33%	6.54%	14.71%	9.39%
100	157.13	13.83%	8.47%	4.98%	10.23%	6.41%

Table 6.4. *Makespan* and average energy savings with respect to *out-degree*

6.1.2 Results for EAGS-D

The amount of energy consumed was measured for HNPD and EAGS-D for different test sets of random DAGs. Test sets were created by combining results from DAGs with similar properties, such as the number of nodes or *CCR*.

The amount of energy consumed was measured for the five different strategies explained before, 5V/off, 2V during idle, 3.3V during idle, 2V scale, and 3.3V scale.

The first test set was achieved by combining DAGs with respect to number of nodes. The energy saving was averaged over DAGs with varying *CCR*, *α, β*, *out-degree,* and *PNR*. Figure 6.6 shows the average energy saved by EAGS-D over HNPD with respect to number of nodes.

The percentage of energy savings increased with increasing the number of nodes. Average energy savings is 15% for a DAG of 10 nodes and the savings gradually increases to 30% for 1000 node DAGs. Larger numbers of nodes increase the chance of dependency between tasks, causing a slight increase in the processor wait time and accordingly increasing energy savings.

Average energy savings ranges between 15% and 30% for processor using 5V/off technique, 13% and 30% for 2V during idle, 12% and 30% when using 3.3V during idle, 14% and 31% for 2V scale, and 12% and 30% for 3.3V scale.

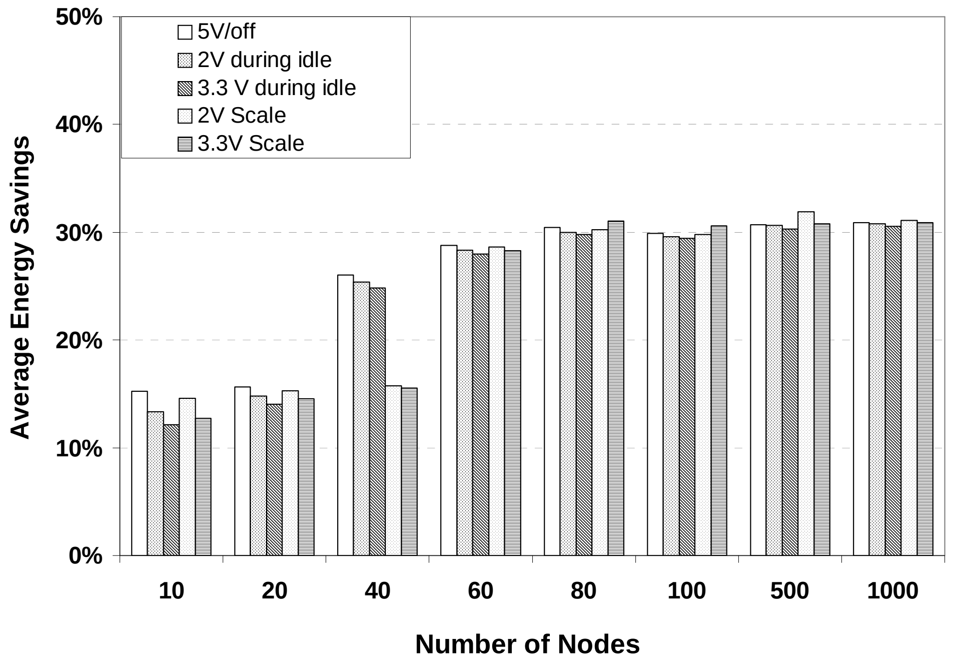


Figure 6.6. Average energy savings with respect to number of nodes

Number of Nodes	*makespan*	Percentage of energy savings				
		5V/off	2V during idle	3.3V during idle	2V scale	3.3V Scale
10	164.3	15.24%	13.36%	12.13%	14.61%	12.75%
20	169.02	15.65%	14.79%	14.05%	15.31%	14.55%
40	318.07	26.02%	25.36%	24.85%	15.75%	15.57%
60	416.31	28.81%	28.33%	27.99%	28.61%	28.27%
80	537.6	30.43%	30%	29.41%	30.26%	31.07%
100	668.29	29.88%	29.6%	29.8%	29.77%	30.58%
500	799.83	30.67%	30.66%	30.3%	31.89%	30.78%
1000	1345.7	30.88%	30.78%	30.55%	31.09%	30.89%

Table 6.5. *Makespan* and average energy savings with respect to number of nodes

The second test set combines DAGs with respect to *CCR*. The average energy savings is averaged for DAGs with varying *n*, *α, β*, *out-degree,* and *PNR* values.

In Figure 6.7 the average energy savings has been plotted with respect to *CCR*. The average energy savings increased with increasing *CCR*. When *CCR* increases, processors incur longer idle times due to communication between tasks. Our algorithm is able to use such idle times to achieve energy savings.

The average energy savings over HNPD listed in Table 6.6 ranges from 8% for *CCR* = 0.1 to 43% when *CCR* = 10 for 5V/off technique. Savings are smaller for 2V during idle; they range between 4% and 39%. Savings are even smaller for the 3.3V during idle; they are 2% for *CCR* = 0.1 and 32% when *CCR* = 10. Savings are higher for 2V scale than 2V during idle; they range from 6% when *CCR* = 0.1 to 50% for *CCR* = 10. And finally the savings are 4% for *CCR* = 0.1 and 48% when *CCR* = 10 for 3.3V scale.

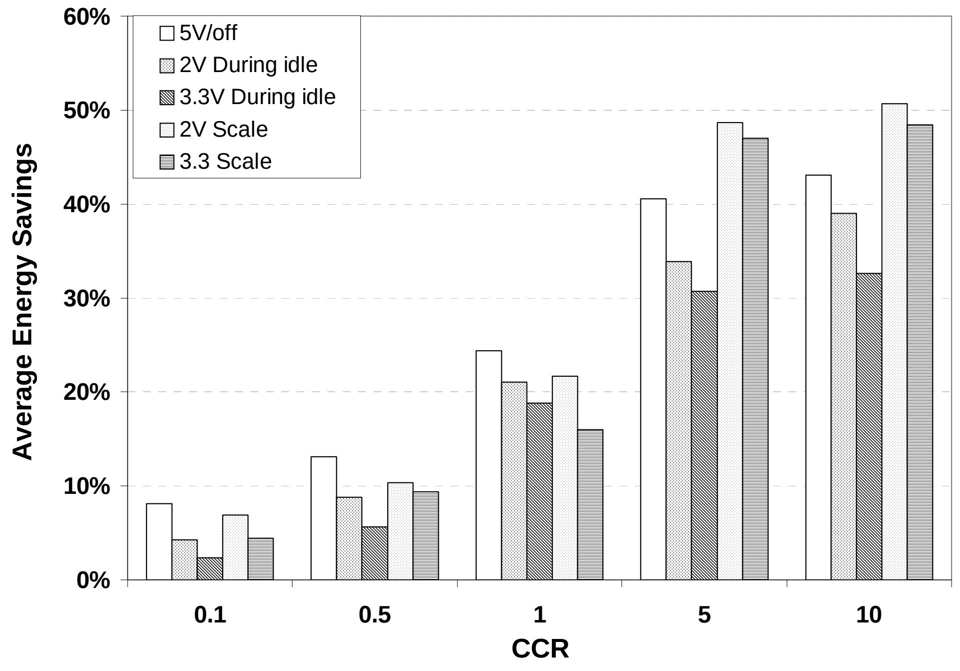


Figure 6.7. Average energy savings with respect to *CCR*

CCR	*Makespan*	Percentage of energy savings				
		5V/off	2V during idle	3.3V during idle	2V Scale	3.3V Scale
0.1	108.44	8.1%	4.25%	2.37%	6.92%	4.41%
0.5	119.11	13.11%	8.81%	5.62%	10.33%	9.38%
1	140.56	24.38%	21.03%	18.8%	21.67%	16.01%
5	206.78	40.58%	33.86%	30.74%	48.67%	47.03%
10	261.67	43.08%	38.98%	32.6%	50.71%	48.42%

Table 6.6. *Makespan* and average energy savings for different *CCR* values

The third test set combines DAGs with respect to processor to node ratio, *PNR*. The average energy savings were averaged over randomly generated DAGs with varying *n*, *CCR*, *α, β*, and *out-degree*. Figure 6.8 shows the average energy saving with respect to *PNR*.

Figure 6.8 shows a 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. The average energy savings measured were 28% for *PNR* = 25% for the 5V/off technique, 27% if processors operate at 2V during idle, 27% for 3.3V during idle, and 27% if they operate at either 2V scale or 3.3V scale. But for *PNR* = 100% the average energy savings for all five strategies were almost equal to 25%, which is due to the significant decrease in the processor's wait time.

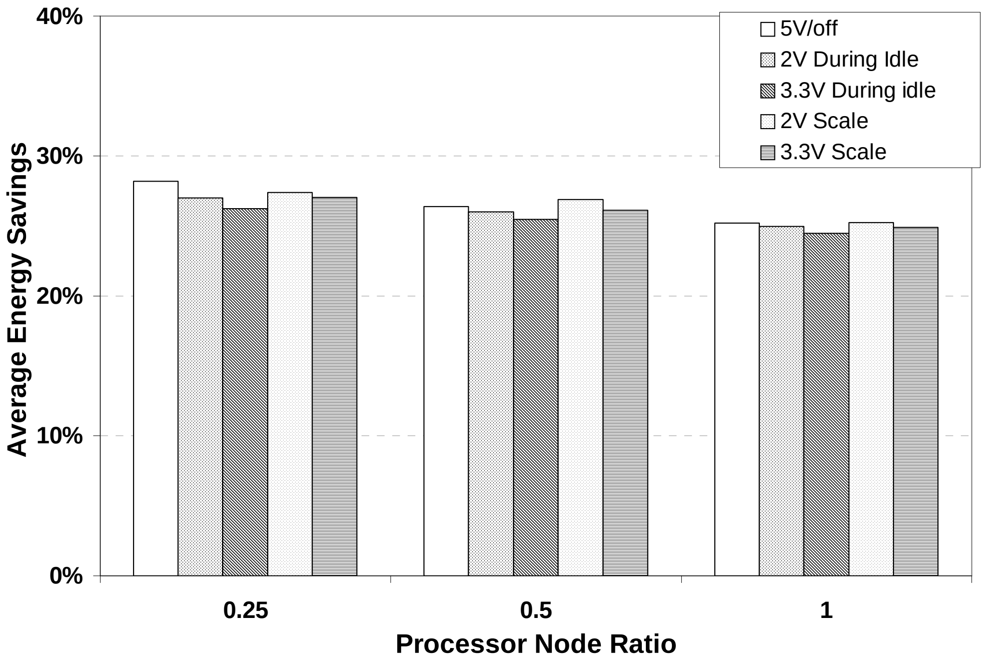


Figure 6.8 Average energy savings with respect to *PNR*

Another test set combined DAGs with respect to shape parameter *α*. The average energy savings was computed over 36,000 randomly generated DAGs with varying n, *PNR*, *CCR*, *β*, and *out-degree*.

In Figure 6.9 the average energy savings for the five operating strategies were plotted with respect to *α*. The results in this figure indicate that the overall energy savings marginally increased with increasing α from 0.5 to 1 and significantly decreased when *α = 2*. Increasing α increases parallelism in the DAG, resulting in more idle time for the processors due to the task dependency, but when *α = 2* all this idle time is being used by processors to duplicate tasks and so the average energy savings decreased, which was not the case with EADAGS. The average energy savings were measured as 18% when *α* = 0.5, 22% when *α* = 1, and 3% when *α* = 2 as in Table 6.7.

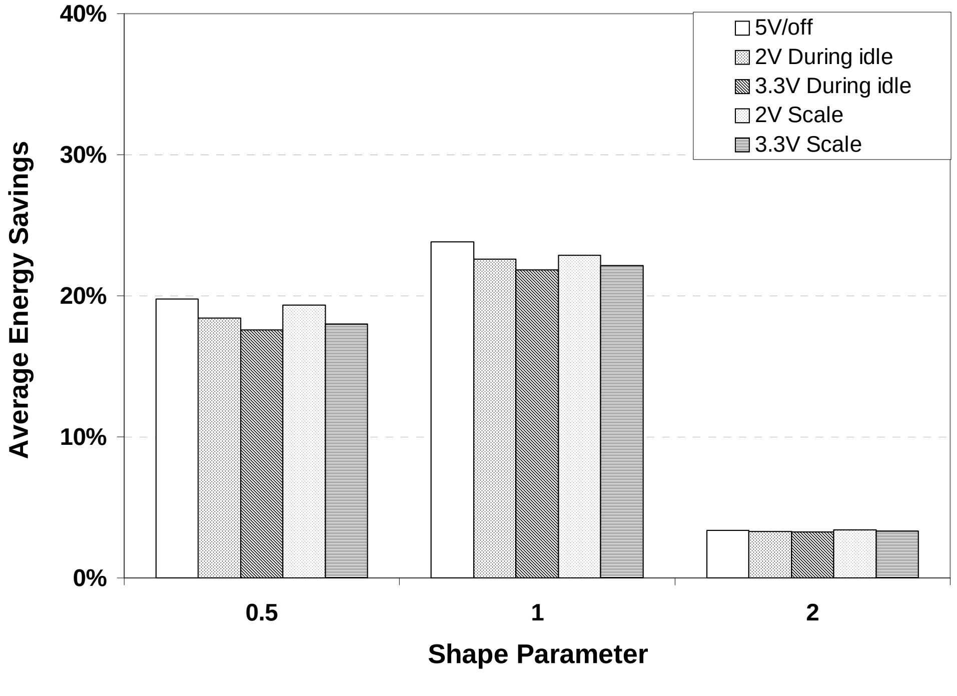


Figure 6.9. Average energy savings with respect to *α*

Shape Parameter	*Makespan*	Percentage of energy savings				
		5V/off	2V during idle	3.3V during idle	2V Scale	3.3V Scale
0.5	97.73	19.75%	18.44%	17.59%	19.33%	18.01%
1	190.6	23.82%	22.62%	21.84%	22.89%	22.15%
2	218.73	3.38%	3.3%	3.25%	3.41%	3.33%

Table 6.7. *Makespan* and average energy savings for different shape parameters

The last test set combines DAGs with respect to *out-degree*. The average energy savings were averaged over 21,600 randomly generated DAGs with varying n, *α, β*, *CCR*, *PNR*.

Figure 6.10 shows the average energy savings with respect to five values for *out-degree*. The results in this figure indicate that the increase in *out-degree* results in smaller average energy savings. A larger *out-degree* allows many processors to run in parallel.

The amount of energy that could be saved ranged between 25% and 10% for processors using 5V/off technique, 23% and 7% for 2V during idle, 20% and 5% for 3.3V during idle, 24% and 9% for 2V scale, and finally 21% to 6% when 3.3V scale is used as shown in Table 6.8.

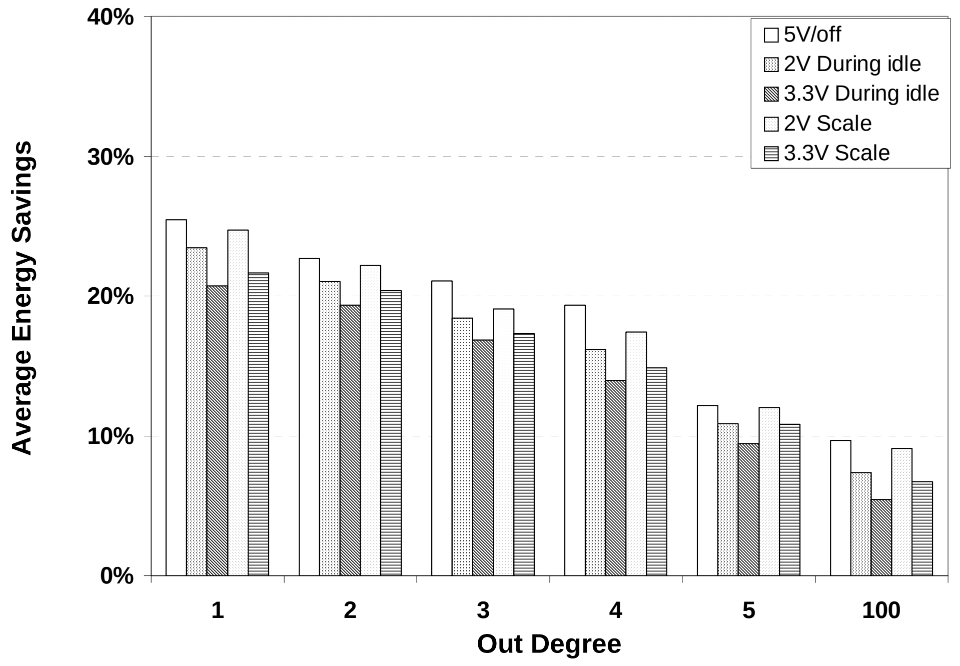


Figure 6.10. Average energy savings with respect to *out-degree*

Out-degree	*Makespan*	Percentage of energy savings				
		5V/off	2V during idle	3.3V during idle	2V Scale	3.3V Scale
1	158.9	25.44%	23.45%	20.74%	24.72%	21.64%
2	218.06	22.68%	21.04%	19.35%	22.19%	20.38%
3	144.93	21.09%	18.44%	16.84%	19.07%	17.33%
4	164.03	19.33%	16.17%	13.97%	17.41%	14.86%
5	138.27	12.16%	10.86%	9.46%	12.03%	10.82%
100	106.71	9.67%	7.36%	5.47%	9.11%	6.71%

Table 6.8. *Makespan* and average energy savings with respect to *out-degree*

Regardless of the changing parameters, the average energy savings for EADAGS is higher than the average energy savings measured for EAGS-D. That is due to the task duplication in EAGS-D. EAGS-D attempts to duplicate the predecessor of tasks to decrease the length of the time for which the node is awaiting data by making use of the processor's idle time. So the same task may be executed on several processors to reduce the *makespan*. And since we use the processor's idle time to save energy, reduction of that time reduces the average energy that could be saved.

6.2 Results for real world problems

To evaluate the performance of the proposed algorithms, we used task graphs of four real world problems: Gaussian elimination [Wu90], Molecular dynamics code [Chun92], fast Fourier Transform [Chun92], and Sieve of Eratosthenes [Bask00].

6.2.1 Gaussian elimination

In mathematics, Gaussian elimination or Gauss-Jordan elimination, named after Carl Friedrich Gauss and Wilhelm Jordan, is an algorithm in linear algebra for determining 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 [Corm90]. Gaussian Elimination is a systematic application of elementary row operations to a system of linear equations in order to convert the system to upper triangular form. Once the coefficient matrix is in upper triangular form, we use back substitution to find a solution. The general procedure for Gaussian Elimination can be summarized in the steps in Table 6.9.

Gaussian Elimination Steps

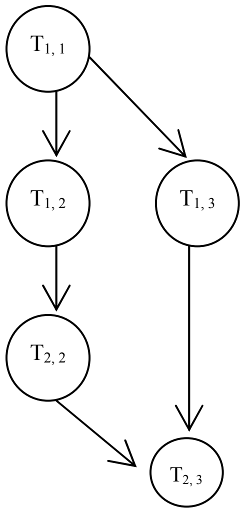
1. Write the augmented matrix for the system of linear equations.
2. Use elementary row operations on the augmented matrix [$A|b$] to transform A into upper triangular form. If a zero is located on the diagonal, switch the rows until a nonzero is in that place. If you are unable to do so, stop; the system has either infinite or no solutions.
3. Use back substitution to find the solution of the problem.

Table 6.9. Gaussian elimination algorithm steps

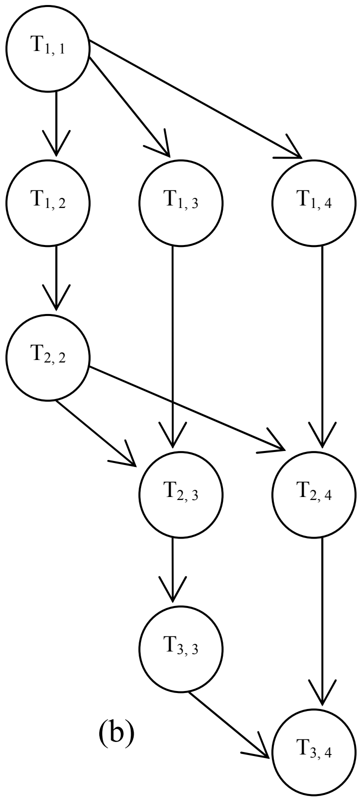
The computational complexity of Gaussian elimination is $O(n^3)$; that is, the number of operations required is (approximately) proportional to n^3 for a matrix of size n x n. The DAG for the Gaussian elimination algorithm for $n=3$, $n=4$, *and* $n=5$ is shown in Figure 6.11 where n is the matrix size. Each $T_{k,k}$ represents a pivot column operation and $T_{k,j}$ is an update operation. The total number of tasks in a graph is $\frac{n^2+n-2}{2}$, where n is the size of the matrix.

In the simulation, a matrix of size 8 x 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

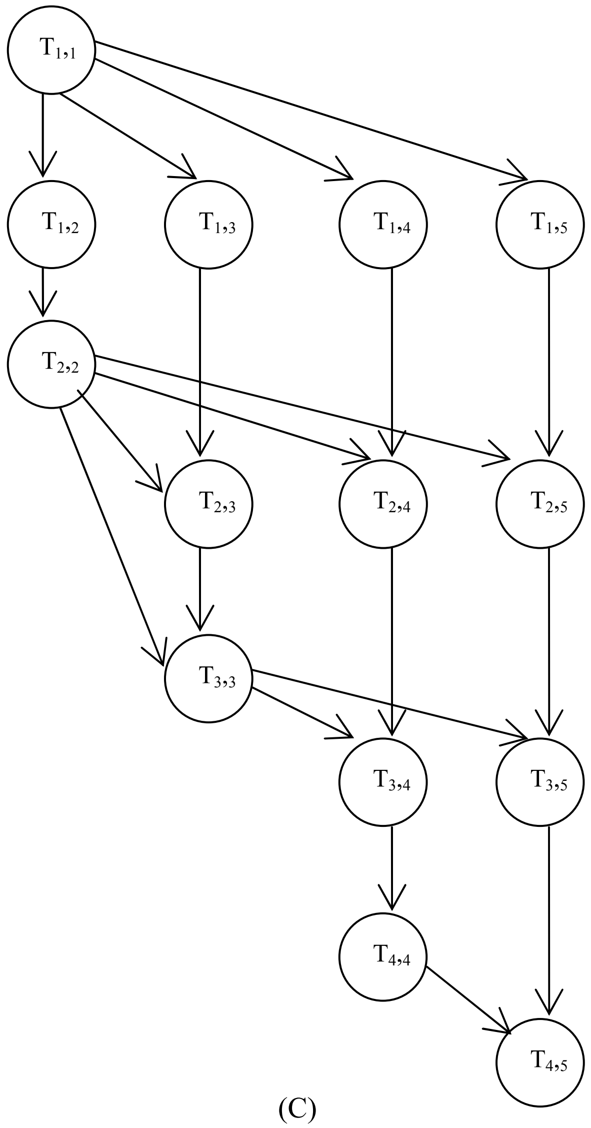
7. *CCR* values were 0.1, 0.5, 1.0, 5.0, and 10. In this experiment since the same operation is executed at every processor and the same information is communicated from one processor to another, a uniform computation cost for all tasks and equal communication cost for all communication links were assumed.



(a)



(b)



(C)

Figure 6.11. Gaussian elimination task graph (a) matrix of size 3, (b) matrix of size 4, (c) matrix of size 5

6.2.1.1 Results for EADAGS

Figure 6.12 and Figure 6.13 show the average energy savings using EADAGS over DPS with respect to number of processors and *CCR* values. Figure 6.12 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 were 32% for 2 processors using the 5V/off technique, 27% of energy savings if processors operate at 2V during idle, 18% energy savings if processors use 3.3V during idle, 29% energy savings for 2V scale, and 19% for 3.3V scale. When three processors are used, the average energy were 52% for the 5V/off technique, 43% and 29% when processors scaled down during their idle time to 2V and 3.3V respectively, and 45% when processors use the 2V scale, and 30% for the 3.3V scale. For four processors the average energy savings were 63%, 53%, 36%, 54%, and 36% for the 5V/off, 2V during idle, 3.3V during idle, 2V scale, and 3.3V scale respectively. The average energy savings were 69%, 57%, 38%, 58%, and 39% when processors use the 5V/off technique, 2V during idle, 3.3V during idle, 2V scale, and 3.3V scale respectively. For 6 processors the average energy savings were 74%, 62%, 42%, 63%, and 42% for the 5V/off, 2V during idle, 3.3V during idle, 2V scale, and 3.3V scale respectively. Finally, the average energy savings were 77%, 64%, 43%, 65%, and 44% when 7 processors are used for the 5V/off technique, 2V during idle, 3.3V during idle, 2V scale, and 3.3V scale respectively.

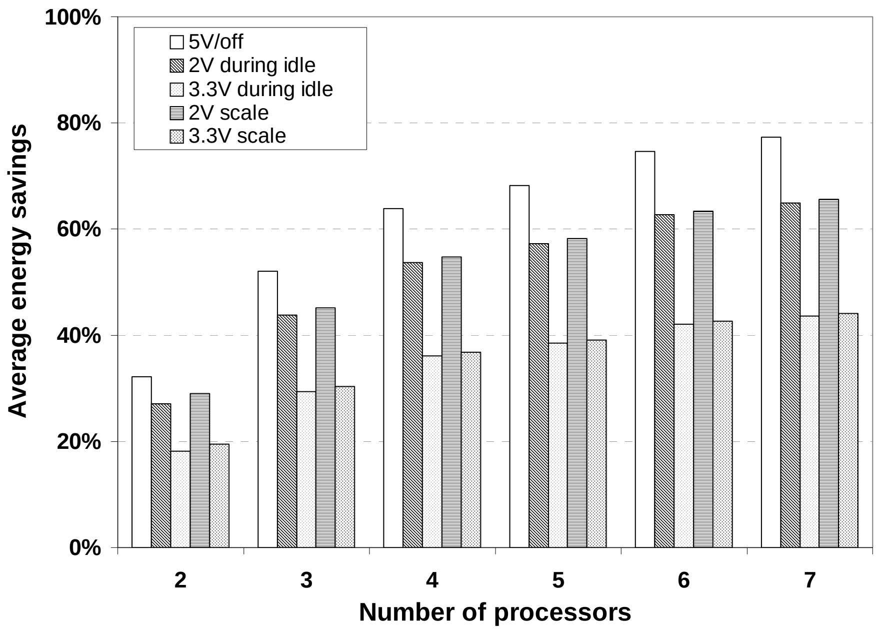


Figure 6.12. Average energy savings for Gaussian elimination algorithm with EADAGS with different number of processors

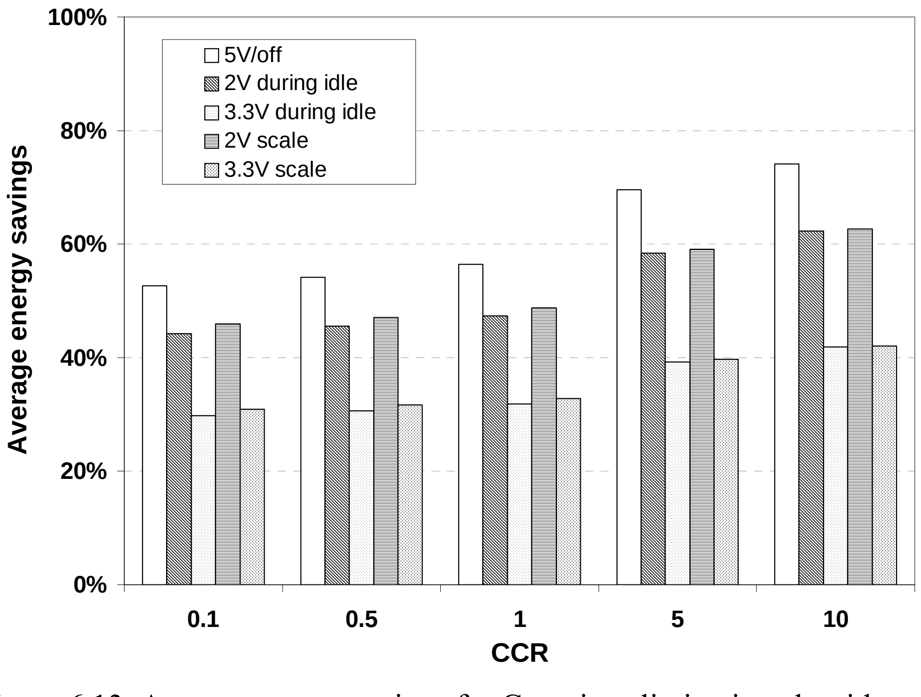


Figure 6.13. Average energy savings for Gaussian elimination algorithm with EADAGS with different *CCR* values

Figure 6.13 plots the average energy savings with respect to different *CCR* values. The average savings increased with increasing *CCR*. The average energy savings over DPS ranges from 52% for *CCR* = 0.1 to 74% when *CCR* = 10 for 5V/off technique. Savings are smaller for 2V during idle; they range between 44% and 62%. Savings are even smaller for the 3.3V during idle; they are 29% for *CCR* = 0.1 and 41% for *CCR* = 10, while for 2V scale and 3.3V scale the average energy savings over DPS ranges from 45% for *CCR* = 0.1 to 62% when *CCR* = 10 and 30% for *CCR* = 0.1 and 42% for *CCR* = 10 respectively. Table 6.10 and Table 6.11 list the *makespan* and the average energy savings for different number of processors and different *CCR* values for the EADAGS algorithm.

Number of processors	*Makespan*	Percentage of energy savings				
		5V/off	2V during idle	3.3V during idle	2V Scale	3.3V Scale
2	820	32.19 %	27.04 %	18.17 %	28.97%	19.46%
3	824.4	52.1 %	43.77 %	29.41 %	45.15%	30.34%
4	860	63.91 %	53.69 %	36.07 %	54.75%	36.79%
5	784.8	68.19 %	57.28 %	38.49 %	58.22%	39.12%
6	851.2	74.63 %	62.69 %	42.12 %	63.44%	42.63%
7	831.2	77.3 %	64.93 %	43.63 %	65.61%	44.08%

Table 6.10. *Makespan* and average energy savings for different number of processors for Gaussian elimination with EADAGS

CCR	*Makespan*	Percentage of energy savings				
		5V/off	2V during idle	3.3V during idle	2V Scale	3.3V Scale
0.1	381.33	52.69 %	44.26 %	29.74 %	45.94%	30.87%
0.5	421.67	54.19 %	45.52 %	30.58 %	47.06%	31.62%
1	480	56.41 %	47.38 %	31.84 %	48.76%	32.76%
5	1063.33	69.55 %	58.42 %	39.25 %	59.06%	39.68%
10	1796.67	74.11 %	62.25 %	41.83 %	62.63%	42.08%

Table 6.11. *Makespan* and average energy savings with respect to *CCR* for Gaussian elimination with EADAGS

6.2.1.2 Results for EAGS-D

The same number of processors and values for *CCR* were tested in the second part of this experiment to evaluate EAGS-D. Figure 6.14 shows the average energy savings for EAGS-D with respect to number of processors. The average energy savings measured were 21% for 2 processors in the 5V/off technique, 17% when processors operate at 2V during idle, 11% if they operate at 3.3V during idle, 20% for 2V scale, and 13% for 3.3V scale. These values increase gradually with increasing the number of processors due to increasing idle time. The average energy savings were 69%, 58%, 38%, 59%, and 39% when 7 processors are used for the 5V/off technique, 2V during idle, 3.3V during idle, 2V scale, and 3.3V scale respectively.

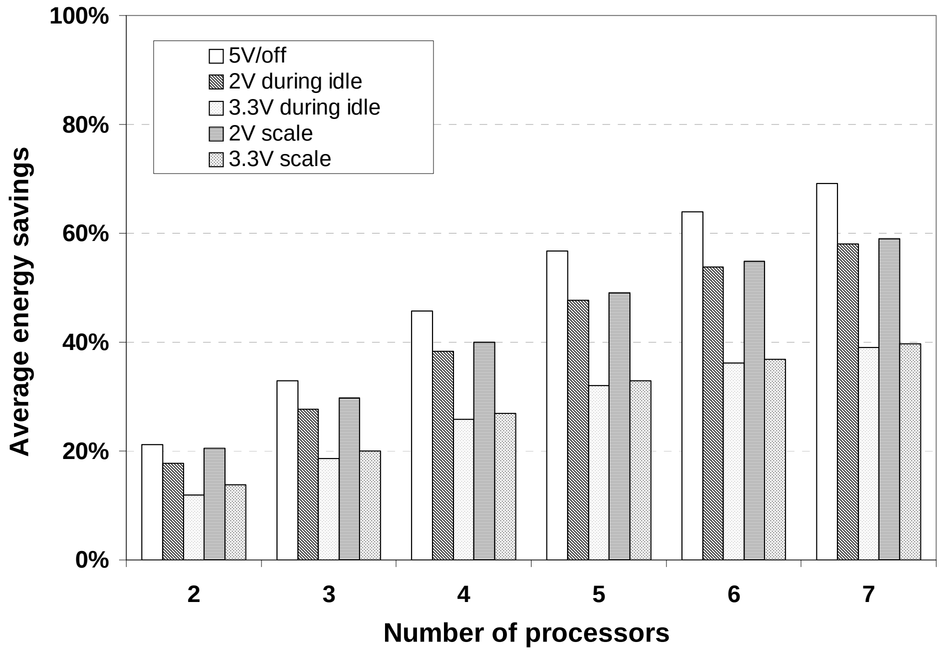


Figure 6.14. Average energy savings for Gaussian elimination algorithm with EAGS-D with different number of processors

Figure 6.15 plots the average energy savings with respect to different *CCR* values. The average savings increase with increasing *CCR*. The average energy savings over HNPD ranges from 33% for *CCR* = 0.1 to 69% when *CCR* = 10 for 5V/off technique. Savings are smaller for 2V during idle operating voltage; they range between 27% and 58%. Savings are even smaller for the 3.3V operating voltage; they are 18% for *CCR* = 0.1 and 39% for *CCR* = 10, while for 2V scale and 3.3V scale the average energy savings over HNPD ranges from 30% for *CCR* = 0.1 to 59% when *CCR* = 10 and 20% for *CCR* = 0.1 and 40% for *CCR* = 10 respectively. Table 6.12 and Table 6.13 list the *makespan* and the average energy savings for different number of processors and different *CCR* values for the EAGS-D algorithm.

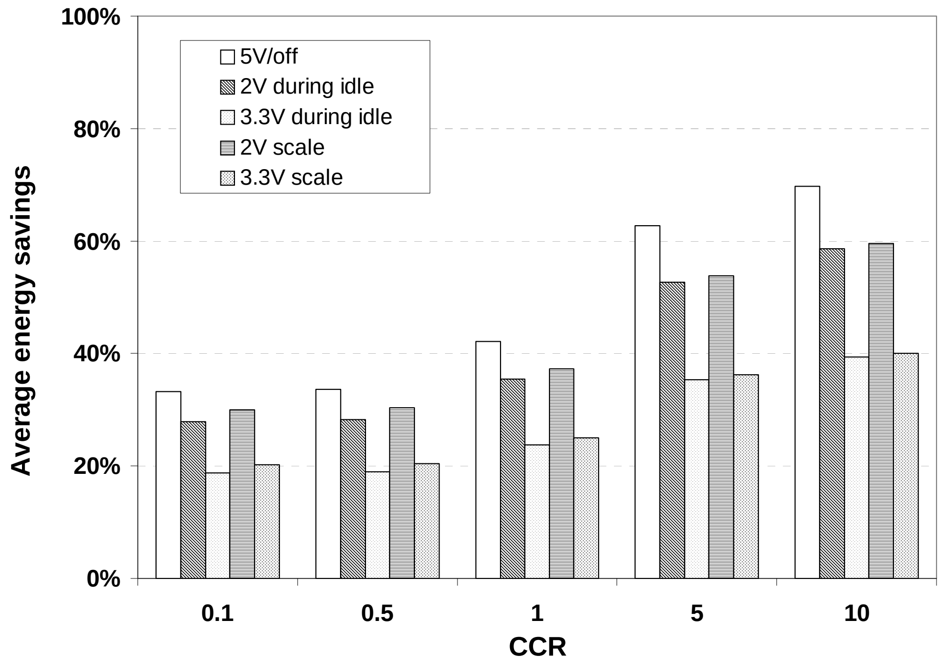


Figure 6.15. Average energy savings for Gaussian elimination algorithm with EAGS-D algorithm with different *CCR* values

Number of processors	*Makespan*	Percentage of energy savings				
		5V/off	2V during idle	3.3V during idle	2V Scale	3.3V Scale
2	484	21.15 %	17.76 %	11.94 %	20.48%	13.76%
3	442.4	32.93 %	27.66 %	18.59 %	29.79%	20.02%
4	432	45.68 %	38.37 %	25.78 %	40.04%	26.91%
5	432	56.78 %	47.69 %	32.05 %	49.03%	32.95%
6	432	63.98 %	53.74 %	36.11 %	54.86%	36.86%
7	432	69.13 %	58.07 %	39.02 %	59.02%	39.66%

Table 6.12. *Makespan* and average energy savings for different number of processors for Gaussian elimination with EAGS-D

CCR	*Makespan*	Percentage of energy savings				
		5V/off	2V during idle	3.3V during idle	2V Scale	3.3V Scale
0.1	303.67	33.21 %	27.9 %	18.74 %	30.03%	20.17%
0.5	305	33.59 %	28.22 %	18.96 %	30.33%	20.38%
1	350	42.13 %	35.39 %	23.78 %	37.27%	25.04%
5	580	62.7 5	52.67 %	35.39 %	53.82%	36.16%
10	673.33	69.73 %	58.58 %	39.36 %	59.58%	40.03%

Table 6.13. *Makespan* and average energy savings with respect to *CCR* for Gaussian elimination with EAGS-D

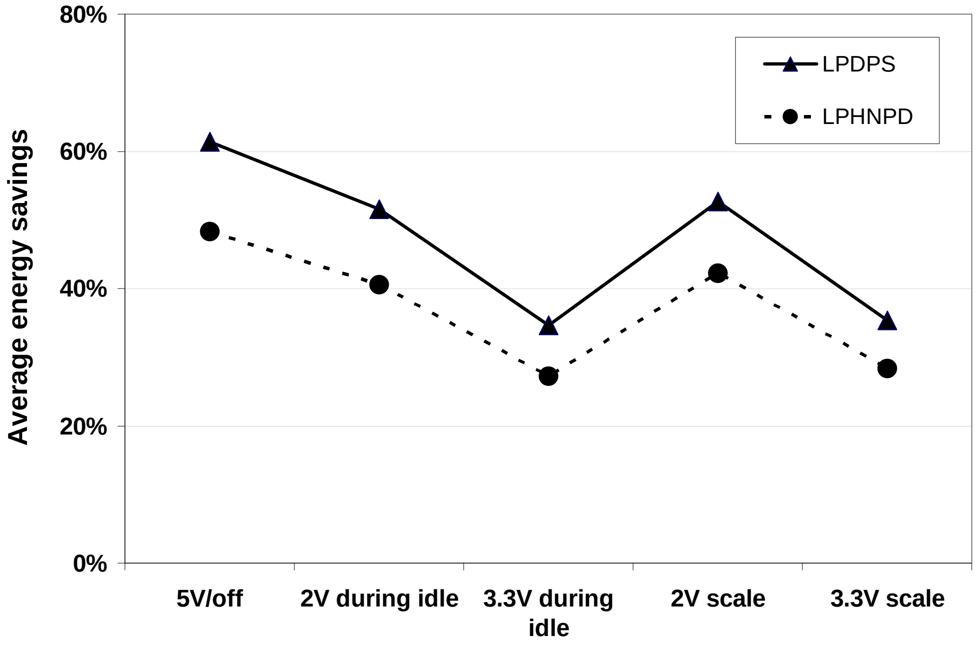


Figure 6.16. Average energy savings for Gaussian elimination algorithm with EADAGS and EAGS-D with voltage scaling levels

Figure 6.16 shows the average energy savings for both EADAGS and EAGS-D with respect to the five voltage scaling levels tested for the Gaussian elimination algorithm for a matrix of size 8. The overall average energy savings for EAGS-D is lower than the average energy savings measured for EADAGS. That is due to the nature of the EAGS-D algorithm, which uses task duplication to minimize the *makespan*. That duplication uses a big portion of the processor's idle time and that reduces energy savings.

6.2.2 Molecular dynamic code

Figure 6.17 represents the DAG of a molecular dynamics code as given in [Chun92]. We used this graph to evaluate the performance of EADAGS and EAGS-D, since the graph has a fixed structure and fixed number of nodes, the only parameters that could be varied was the number of processors and *CCR* values. Since there are at most seven tasks at any level in Figure 6.17, the number of processors were bounded to seven. The amount of energy consumed is measured for the five different voltage scaling levels explained earlier. 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.

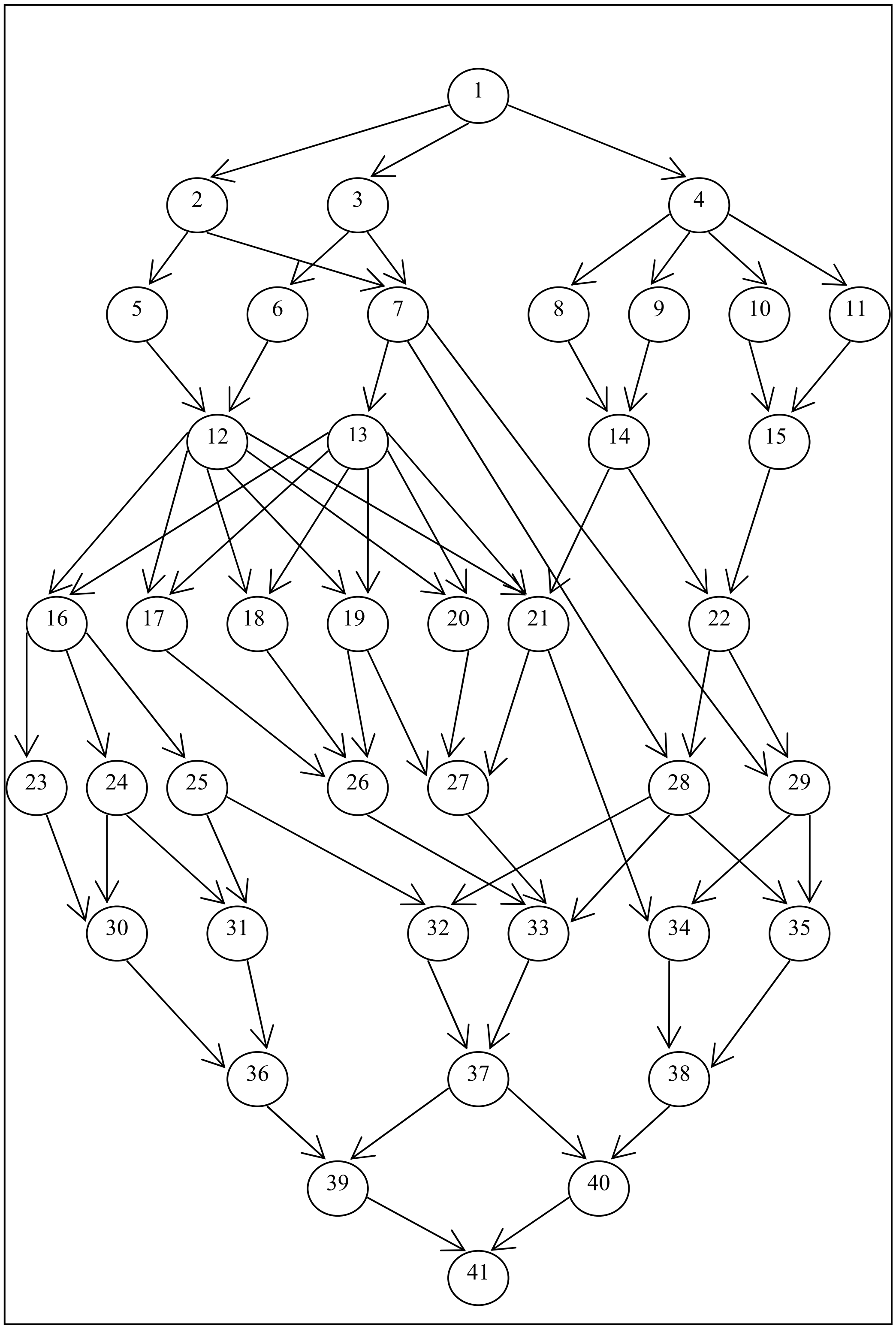


Figure 6.17. Directed a-cyclic graph (DAG) for a molecular dynamics code

6.2.2.1 Results for EADAGS

Figure 6.18 and Figure 6.19 show the average energy savings for EADAGS with respect to number of processors and *CCR* values respectively. Figure 6.18 shows an increase in the average energy savings with increasing number of processors. The average energy savings measured was 22% for 2 processors in the 5V/off technique, 18% when processors operate at 2V during idle, 12% if they operate at 3.3V during idle, 19% for the 2V scale, and 13% for the 3.3Vscale. When three processors are used, the average energy savings were 50% for the 5V/off technique, 40% when processors scaled down to 2V during idle, 29% for 3.3V during idle, 44% for 2V scale, and 30% for 3.3V scale. For four processors, the average energy savings were 52%, 43%, 29%, 44%, and 30% for the 5V/off, 2V during idle, 3.3V during idle, 2V scale, and 3.3V scale respectively. The average energy savings were 59%, 49%, 33%, 50%, and 33% when 5 processors are used for the 5V/off technique, 2V during idle, 3.3V during idle, 2V scale, and 3.3V scale respectively. For 6 processors the average energy savings were 63%, 52%, 35%, 53%, and 36% for the 5V/off, 2V during idle, 3.3V during idle, 2V scale, and 3.3V scale respectively. Finally, the average energy savings were 67%, 57%, 38%, 57%, and 38% when 7 processors are used for the 5V/off technique, 2V during idle, 3.3V during idle, 2V scale, and 3.3V scale respectively.

The *makespan* and the average energy savings for different number of processors are listed in Table 6.14.

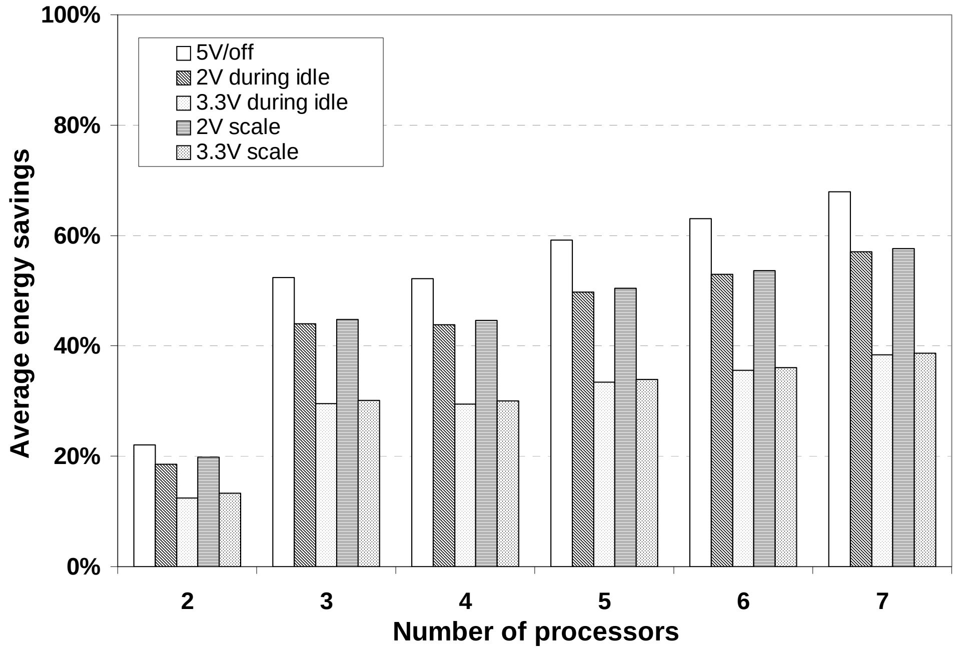


Figure 6.18, Average energy savings for molecular dynamics code with EADAGS with different number of processors

Number of processors	*Makespan*	Percentage of energy savings				
		5V/off	2V during idle	3.3V during idle	2V Scale	3.3V Scale
2	740.8	22.08 %	18.55 %	12.46 %	19.79%	13.30%
3	859.2	52.42 %	44.04 %	29.59 %	44.82%	30.11%
4	67.2	52.19 %	43.84 %	29.46 %	44.64%	30.00%
5	663.2	59.19 %	49.72 %	33.41 %	50.41%	33.87%
6	634	63.07 %	52.98 %	35.6 %	53.61%	36.02%
7	631.2	67.09 %	57.09 %	38.36 %	57.64%	38.73%

Table 6.14. *Makespan* and average energy savings for different number of processors for molecular dynamics code with EADAGS

Figure 6.19 plots the average energy savings with respect to different *CCR* values. The average savings increased with increasing *CCR*. The average energy savings over DPS ranges from 45% for *CCR* = 0.1 to 64% when *CCR* = 10 for the 5V/off technique. Savings are smaller for 2V during idle; they range between 38% and 54%. Savings are even smaller for the 3.3V during idle; they are 25% for *CCR* = 0.1 and 36% for *CCR* = 10, while for 2V scale and 3.3V scale the average energy savings over DPS ranges from 39% for *CCR* = 0.1 to 54% when *CCR* = 10 and 26% for *CCR* = 0.1 and 36% for *CCR* = 10 respectively. The *makespan* and the average energy savings for different *CCR* values are listed in Table 6.15.

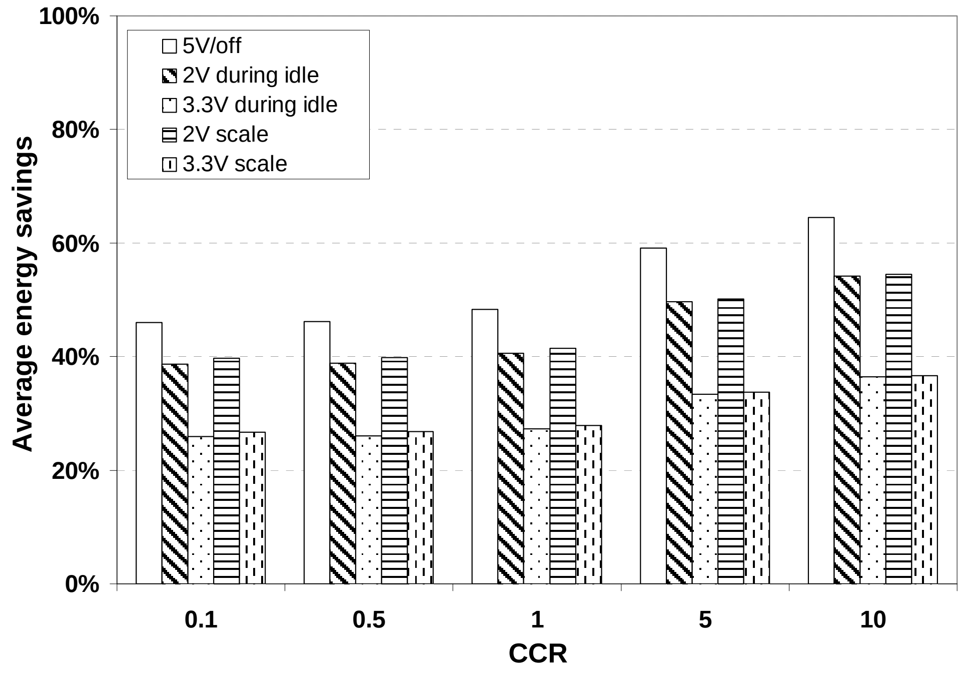


Figure 6.19. Average energy savings for molecular dynamics code with EADAGS with different *CCR* values

CCR	*Makespan*	Percentage of energy savings				
		5V/off	2V during idle	3.3V during idle	2V Scale	3.3V Scale
0.1	391.33	45.99 %	38.63 %	25.96 %	39.72%	26.69%
0.5	411.67	46.21 %	38.81 %	26.08 %	39.85%	26.78%
1	453.33	48.29 %	40.56 %	27.25 %	41.50%	27.89%
5	853.33	59.11 %	49.65 %	33.36 %	50.17%	33.71%
10	1396.67	64.5 %	54.18 %	36.41 %	54.51%	36.63%

Table 6.15. *Makespan* and average energy savings with respect to *CCR* for molecular dynamics code with EADAGS

6.2.2.2 Results for EAGS-D

Figure 6.20 and Figure 6.21 show the average energy savings for EAGS-D with respect to number of processors and *CCR* values respectively.

Figure 6.20 shows an increase in the average energy savings with increasing number of processors. The average energy savings measured were 18% for 2 processors using the 5V/off technique, 15% when processors operate at 2V during idle, 10% if they operate at 3.3V during idle, 17% for 2V scale, and 11% for 3.3V scale. The average energy savings increases to 61%, 51%, 34%, 52%, and 35% when 7 processors are used for the 5V/off technique, 2V during idle, 3.3V during idle, 2V scale, and 3.3V scale respectively. The *makespan* and the average energy savings for different number of processors are listed in Table 6.16.

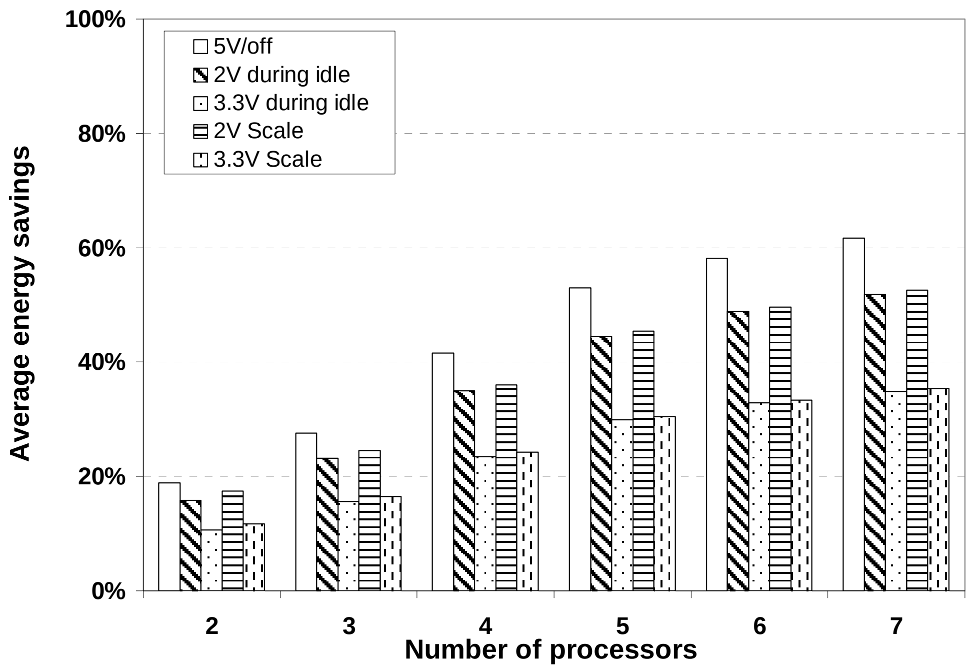


Figure 6.20. Average energy savings for molecular dynamics code with EAGS-D with different number of processors

Number of processors	*Makespan*	Percentage of energy savings				
		5V/off	2V during idle	3.3V during idle	2V Scale	3.3V Scale
2	566.4	18.87 %	15.85 %	10.65 %	17.44%	11.71%
3	488.8	27.58 %	23.17 %	15.57 %	24.48%	16.45%
4	467.2	41.6 %	34.95 %	23.48 %	36.02%	24.20%
5	459.2	52.96 %	44.49 %	29.89 %	45.38%	30.49%
6	445.2	58.15 %	48.85 %	32.82 %	49.64%	33.35%
7	438.4	61.72 %	51.85 %	34.84 %	52.56%	35.31%

Table 6.16. *Makespan* and average energy savings for different number of processors for molecular dynamics code with EAGS-D

Figure 6.21 plots the average energy savings with respect to different *CCR* values. The average savings increased with increasing *CCR*. The average energy savings over HNPD ranges from 24% for *CCR* = 0.1 to 74% when *CCR* = 10 for the 5V/off technique. Savings are smaller for 2V during idle; they range between 20% and 62%. Savings are even smaller for the 3.3V during idle; they are 13% for *CCR* = 0.1 and 41% for *CCR* = 10, while for 2V scale and 3.3V scale the average energy savings over HNPD ranges from 220% for *CCR* = 0.1 to 62% when *CCR* = 10 and 15% for *CCR* = 0.1 and 42% for *CCR* = 10 respectively. The *makespan* and the average energy savings for different *CCR* values are listed in Table 6.17.

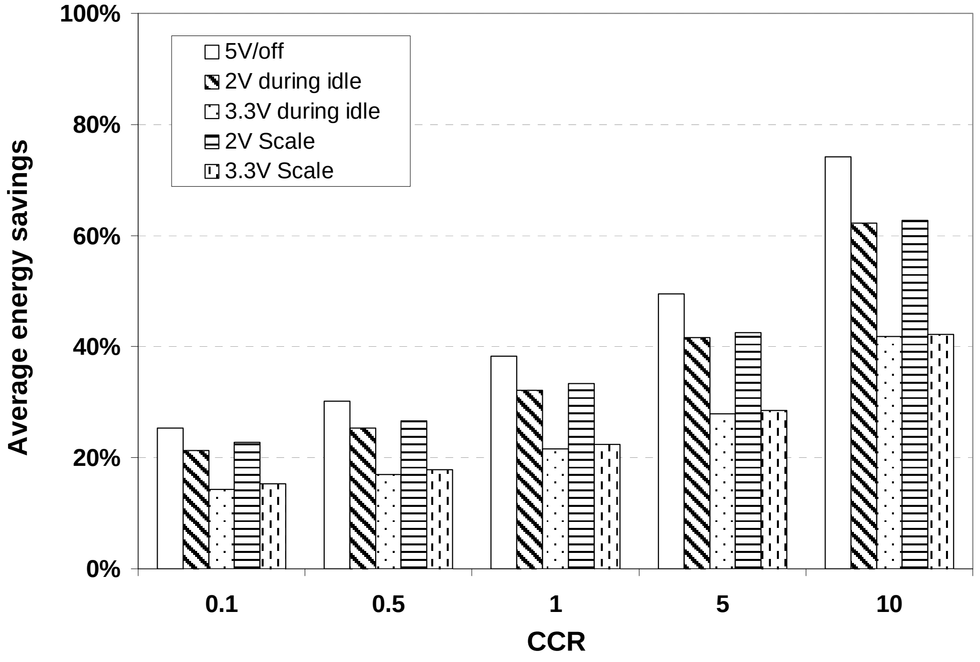


Figure 6.21. Average energy savings for molecular dynamics code with EAGS-D with different *CCR* values

CCR	*Makespan*	Percentage of energy savings				
		5V/off	2V during idle	3.3V during idle	2V Scale	3.3V Scale
0.1	299.33	25.37 %	21.31 %	14.32 %	22.73%	15.27%
0.5	321.67	30.13 %	25.31 %	17.01 %	26.64%	17.90%
1	360	38.28 %	32.16 %	21.61 %	33.35%	22.41%
5	506.67	49.51 %	41.59 %	27.94 %	42.46%	28.53%
10	900	74.12 %	62.26 %	41.83 %	62.75%	42.16%

Table 6.17. *Makespan* and average energy savings with respect to *CCR* for molecular dynamics code with EAGS-D

Figure 6.22 shows the average energy savings for both EADAGS and EAGS-D with respect to the five voltage scaling strategies tested for the molecular dynamic code algorithm. The overall average energy savings for EAGS-D is lower than the average energy savings measured for EADAGS. That is due to the nature of the EAGS-D algorithm which uses task duplication to minimize the *makespan*. That duplication uses a big portion of the processor's idle time and that reduces energy savings.

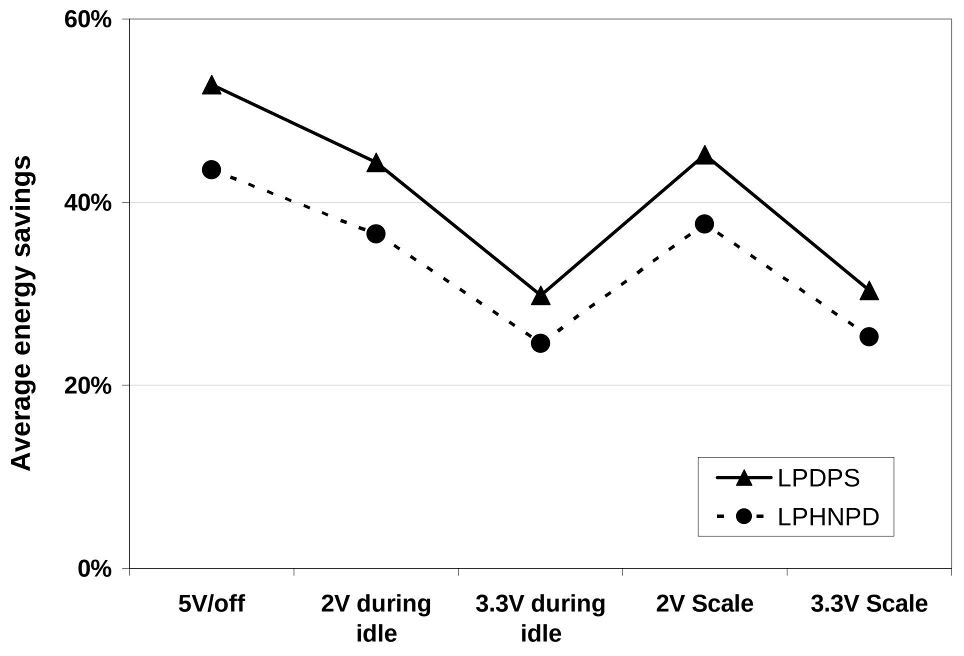


Figure 6.22. Average energy savings for molecular dynamics code algorithm with EADAGS and EAGS-D with voltage scaling levels

6.2.3 Fast Fourier Transform FFT

FFT is an efficient algorithm to compute the discrete Fourier transform (DFT) and its inverse. FFTs are of great importance to a wide variety of applications, from digital signal processing to solving partial differential equations to algorithms for multiplying large integers. DFT and FFT are used to generate frequency analysis of a discrete non-periodic signal. The computation of DFT is complicated; it involves many additions and multiplications involving complex numbers. Even a simple eight sample signal would require 49 complex multiplications and 56 complex additions to work out the DFT. At this level it is still manageable; however a realistic signal could have 1024 samples which requires over 20,000,000 complex multiplications and additions. FFT is a simpler method of laying out the computation and much faster for larger number of samples.

The idea behind the FFT is the *divide and conquer* approach, by breaking up the original *N* point sample into two *(N/2)* sequences. This is because a series of smaller problems is easier to solve than one large one. The DFT requires $(N\text{-}1)^2$ complex multiplications and *N(N-1)* complex additions as opposed to the FFT's approach, which only requires 1 multiplication and 2 additions and the recombination of the points which is minimal [Corm90].

The recursive, one-dimensional FFT task graph for 4 data points is shown in Figure 6.23 [Chun92]. The FFT algorithm consists of two parts: recursive calls and the butterfly operations. The task graph in Figure 6.23 can be divided into two parts; the tasks above the dashed line are the recursive call tasks while the tasks below the dashed line are the butterfly operation tasks.

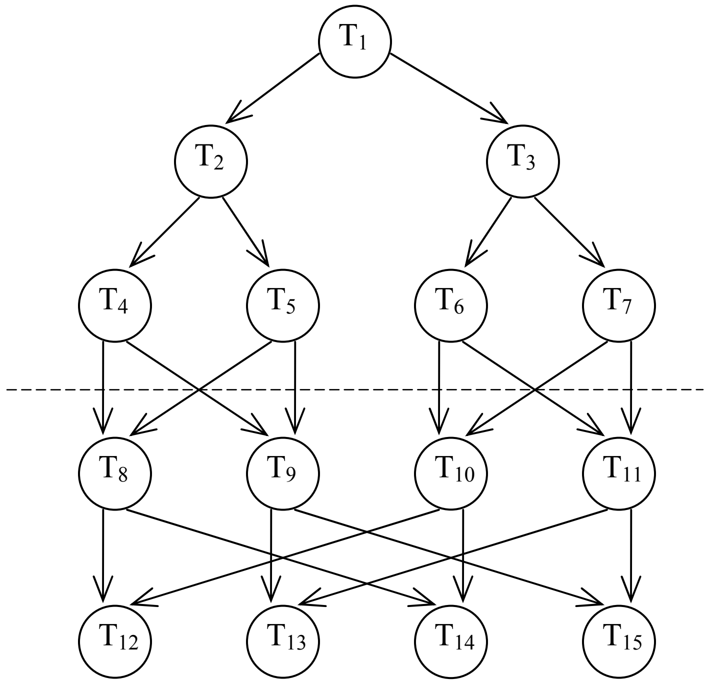


Figure 6.23. The generated DAG for FFT with four points

We used this task graph to evaluate the performance of EADAGS and EAGS-D. Since the graph has a fixed structure and fixed number of nodes, the only parameters we changed were the number of processors and *CCR* values. Since there are at most four tasks at any level in Figure 6.23, the number of processors were bounded to four processors starting with only 2 processors in the system and up to 4 processors incrementing by 1. Each path from the entry node to an exit node is a critical path since the computation cost of tasks in any level are equal and the communication costs of all edges between two consecutive levels are equal. The amount of energy consumed was measured for the five different strategies listed before.

6.2.3.1 Results for EADAGS

Figure 6.24 shows the average energy savings for EADAGS with respect to number of processors. Figure 6.24 shows a decrease in the average energy savings with increasing number of processors.

This is because increasing number of processors allows several parallel task executions, thus minimizing the wait times which were used by our algorithm to save energy. The average energy savings measured were 34% for 2 processors using the 5V/off technique, 29% when processors operate at 2V during idle, 19% if they operate at 3.3V during idle, 30% for 2V scale, and 20% for 3.3V scale. When three processors are used, the average energy were 51% for the 5V/off technique, 42% for 2V during idle, 27% for 3.3V during idle, 44% for 2V scale, and 29% savings for 3.3V scale. For four processors, the average energy savings are 53%, 45%, 31%, 47%, and 33% for the 5V/off, 2V during idle, 3.3V during idle, 2V scale, and 3.3V scale respectively. Table 6.18 lists the average energy savings and the *makespan* for EADAGS over DPS for different number of processors.

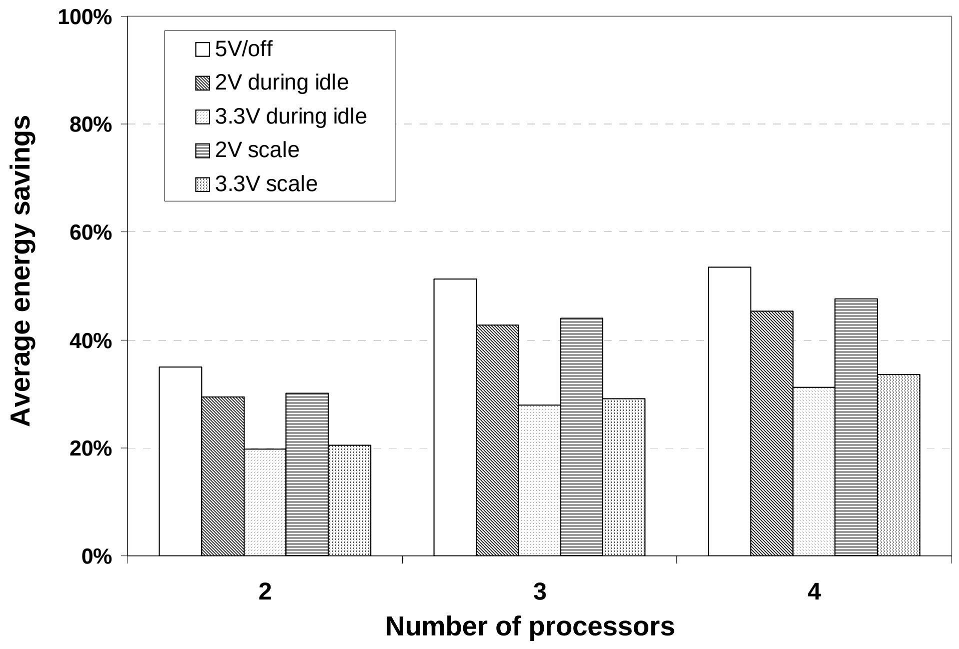


Figure 6.24. Average energy savings for FFT with EADAGS with different number of processors

Number of processors	*Makespan*	Percentage of energy savings				
		5V/off	2V during idle	3.3V during idle	2V Scale	3.3V Scale
2	233	34.97%	29.37%	19.74%	30.08%	20.43%
3	296	55.89%	46.95%	31.54%	44.02%	29.11%
4	232.8	51.32%	43.11%	28.96%	47.66%	33.59%

Table 6.18. *Makespan* and average energy savings with respect to number of processors for FFT with EADAGS

Figure 6.25 plots the average energy savings with respect to different *CCR* values. The average savings increased with increasing *CCR*. When *CCR* increases, processors incur longer idle times due to communication between tasks. Our algorithm was able to use such idle times to achieve energy savings.

The average energy savings over DPS ranges from 23% for *CCR* = 0.1 to 77% when *CCR* = 10 for 5V/off technique. Savings are smaller for 2V during idle; they range between 19% and 64%. Savings are even smaller for the 3.3V during idle; they are 12% for *CCR* = 0.1 and 43% for *CCR* = 10, while for 2V scale and 3.3V scale the average energy savings over DPS ranges from 23% for *CCR* = 0.1 to 64% when *CCR* = 10 and 16% for *CCR* = 0.1 and 43% for *CCR* = 10 respectively.

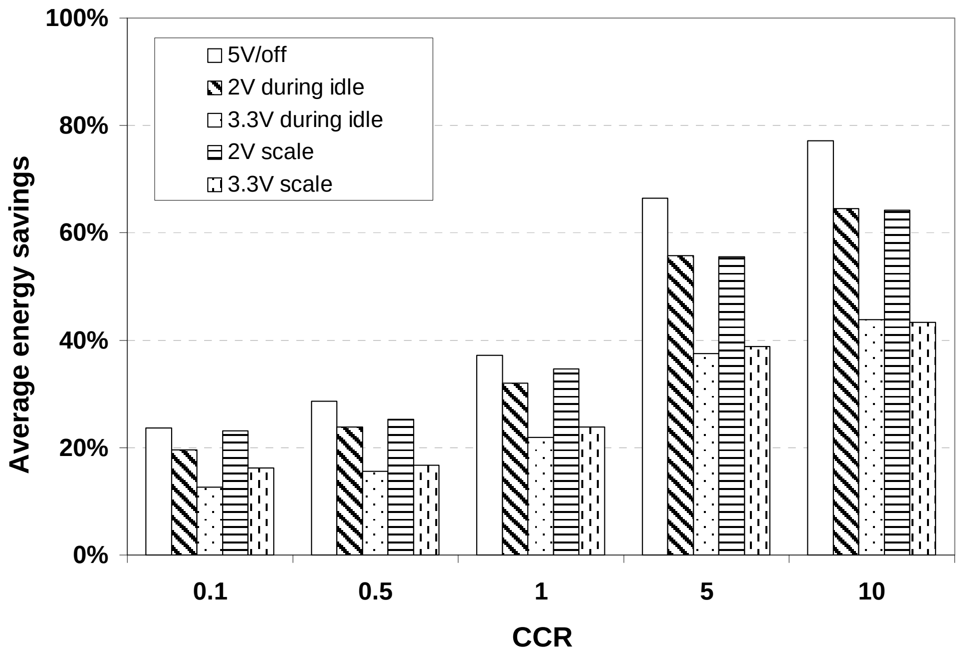


Figure 6.25. Average energy savings for FFT with EADAGS with different *CCR* values

Table 6.19 lists the average energy savings and the *makespan* for EADAGS over DPS for different *CCR* values.

CCR	*Makespan*	Percentage of energy savings				
		5V/off	2V during idle	3.3V during idle	2V Scale	3.3V Scale
0.1	155.33	25.30 %	21.25 %	14.28 %	23.13%	16.23%
0.5	163.33	29.95 %	25.16 %	16.9 %	25.32%	16.67%
1	180	36.5 %	30.67 %	20.61 %	34.67%	23.89%
5	340	66.42 %	55.79 %	37.48 %	55.58%	38.81%
10	540	78.79 %	66.19 %	44.47 %	64.22%	43.32%

Table 6.19. *Makespan* and average energy savings with respect to *CCR* for FFT with EADAGS

6.2.3.2 Results for EAGS-D

Figure 6.26 and Figure 6.27 show the average energy savings for EAGS-D with respect to number of processors and *CCR* values respectively.

Figure 6.26 shows an increase in the average energy savings with increasing number of processors. The average energy savings measured were 18% for 2 processors for the 5V/off technique, 14% for 2V during idle, 9% for 3.3V during idle time, 15% for 2V scale, and 10% for 3.3V scale. When three processors are used, the average energy were 39% for the 5V/off technique, 33% for 2V during idle, 21% for 3.3V during idle, 34% for 2V scale, and 22% for 3.3V scale. For four processors, the average energy savings were 49%, 41%, 27%, 42%, and 29% for the 5V/off, 2V during idle, 3.3V during idle, 2V scale, and 3.3V scale respectively.

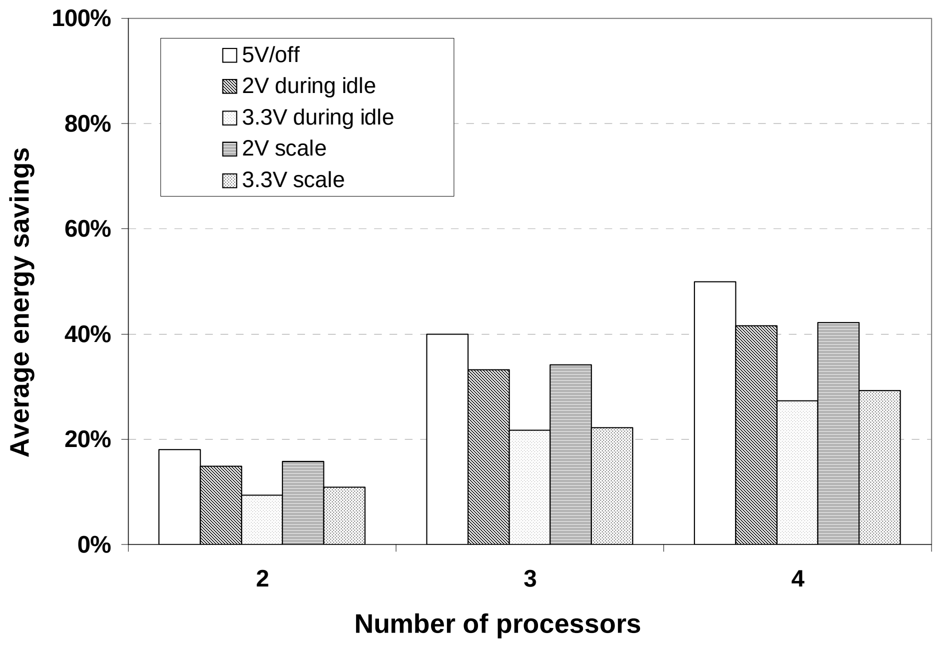


Figure 6.26. Average energy savings for FFT with EAGS-D with different number of processors

Figure 6.27 plots the average energy savings with respect to different *CCR* values. The average savings increased with increasing *CCR*. When *CCR* increases, processors incur longer idle times due to communication between tasks. Our algorithm was able to use such idle times to achieve energy savings. The average energy savings over HNPD ranges from 16% for *CCR* = 0.1 to 61% when *CCR* = 10 for 5V/off technique. Savings are smaller for 2V during idle; they range between 13% and 51%. Savings are even smaller for the 3.3V during idle; they are 9% for *CCR* = 0.1 and 34% for *CCR* = 10, while for 2V scale and 3.3V scale the average energy savings over HNPD ranges from 15% for *CCR* = 0.1 to 51% when *CCR* = 10 and 11% for *CCR* = 0.1 and 35% for *CCR* = 10 respectively. Tables 6.20 and 6.21 lists the average energy savings for EAGS-D over

HNPD for different processors number and for different *CCR* values respectively

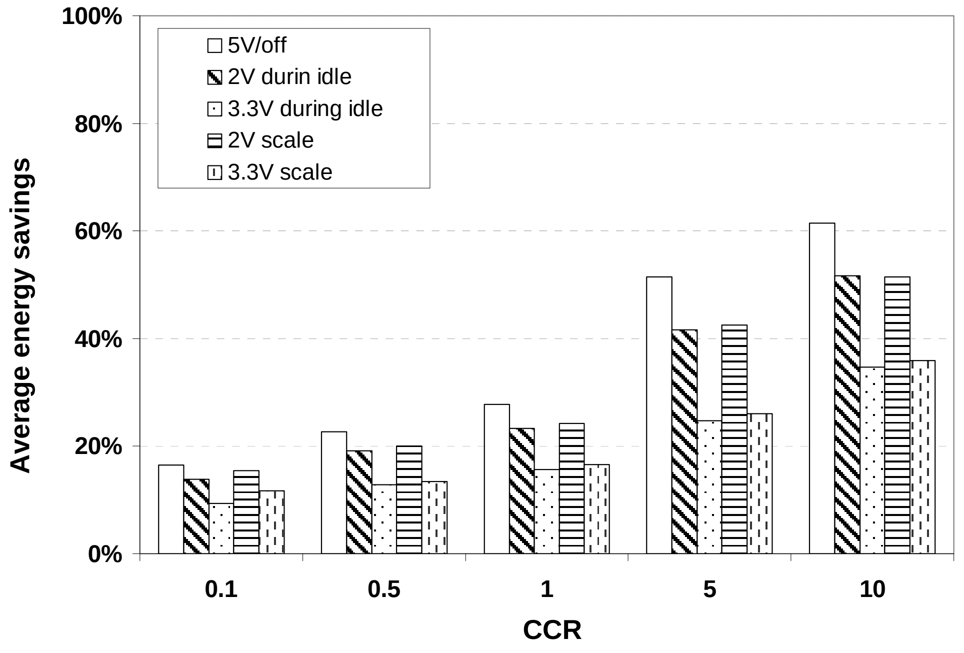


Figure 6.27. Average energy savings for FFT with EAGS-D with different *CCR* values

Number of processors	*Makespan*	Percentage of energy savings				
		5V/off	2V during idle	3.3V during idle	2V Scale	3.3V Scale
2	218.4	18.09 %	14.88 %	9.34%	15.82%	10.90%
3	208	39.96 %	33.25 %	21.68 %	34.18%	22.21%
4	199.2	49.93 %	41.62 %	27.31 %	42.15%	29.23%

Table 6.20. *Makespan* and average energy savings for different number of processors for FFT with EAGS-D

CCR	*Makespan*	Percentage of energy savings				
		5V/off	2V during idle	3.3V during idle	2V Scale	3.3V Scale
0.1	136	16.52 %	13.88 %	9.33 %	15.45%	11.74%
0.5	146.67	22.71 %	19.08 %	12.82 %	20.05%	13.46%
1	160	27.78 %	23.33 %	15.67 %	24.21%	16.56%
5	300	51.48 %	41.64 %	24.7 %	42.53%	26.09%
10	300	61.48 %	51.64 %	34.7 %	51.48%	35.91%

Table 6.21. *Makespan* and average energy savings with respect to *CCR* for FFT with EAGS-D

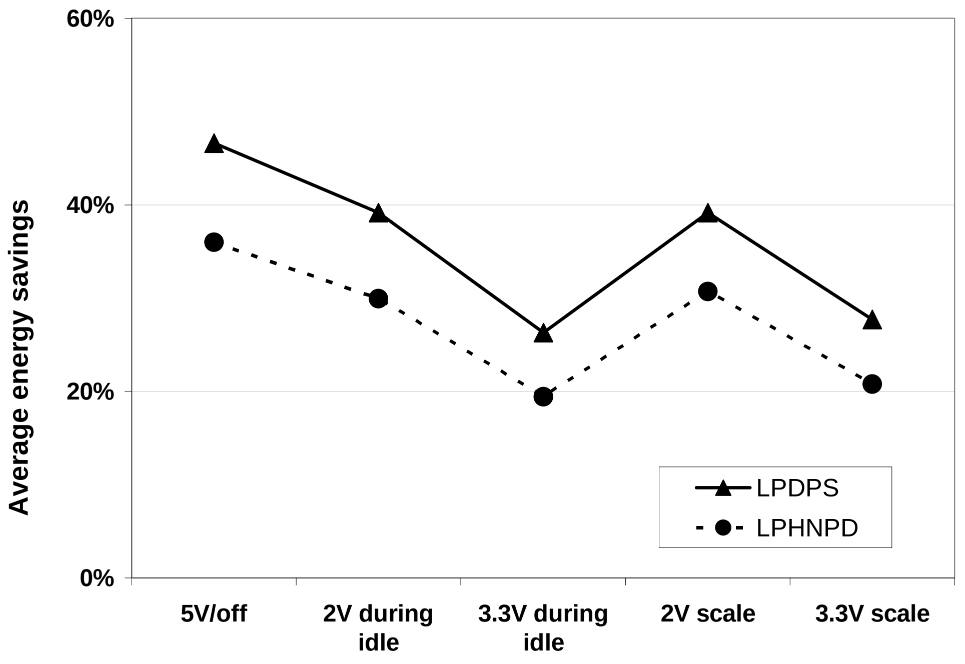


Figure 6.28. Average energy savings for FFT with EADAGS and EAGS-D with voltage scaling levels

Figure 6.28 shows the average energy savings for both EADAGS and EAGS-D with respect to the five voltage scaling strategies tested for a four point FFT DAG. The overall average energy savings for EAGS-D is lower than the average energy savings measured for EADAGS. That is due to the nature of the EAGS-D algorithm, which uses task duplication to minimize the *makespan*. That duplication uses a big portion of the processor's idle time and that reduces the amount of energy savings.

6.2.4. Sieve of Eratosthenes

Sieve Eratosthenes is a method of identifying all prime numbers in a sequence of numbers up to a certain *N*. A prime number is a natural number greater than 1 that can be divided only by itself and by 1, while a composite number *n* is a natural number that can be divided by a number less than *n* and greater that 1. The Sieve of Eratosthenes identifies all prime numbers up to a given number *N* as follows [Corm90]:

1. Write down all numbers *1, 2, 3,…, N*. We will eliminate composites by marking them. Initially all numbers are unmarked.
2. Mark number 1 as special (it is neither prime nor composite).
3. Set *k = 1*. While *k* is less than the square root of *N,* do this:
    a. Find the first number in the list greater than *k* that has not been identified as composite (the first number found is 2) and call it *m*. Mark the numbers *2m, 3m, 4m,……* as composites. (Thus in the first run we mark all even numbers greater than 2. In the second run we mark all multiples of 3 greater than 3.)

b. *m* is a prime number; put it on your list.

c. Set *k* = *m* and repeat.

4. Put the remaining unmarked numbers in the list of prime numbers.

The Sieve of Eratosthenes algorithm can be presented by a task graph DAG. The DAG for the Sieve of Eratosthenes for *N* = *32* is shown in Figure 6.29. [Bask00]

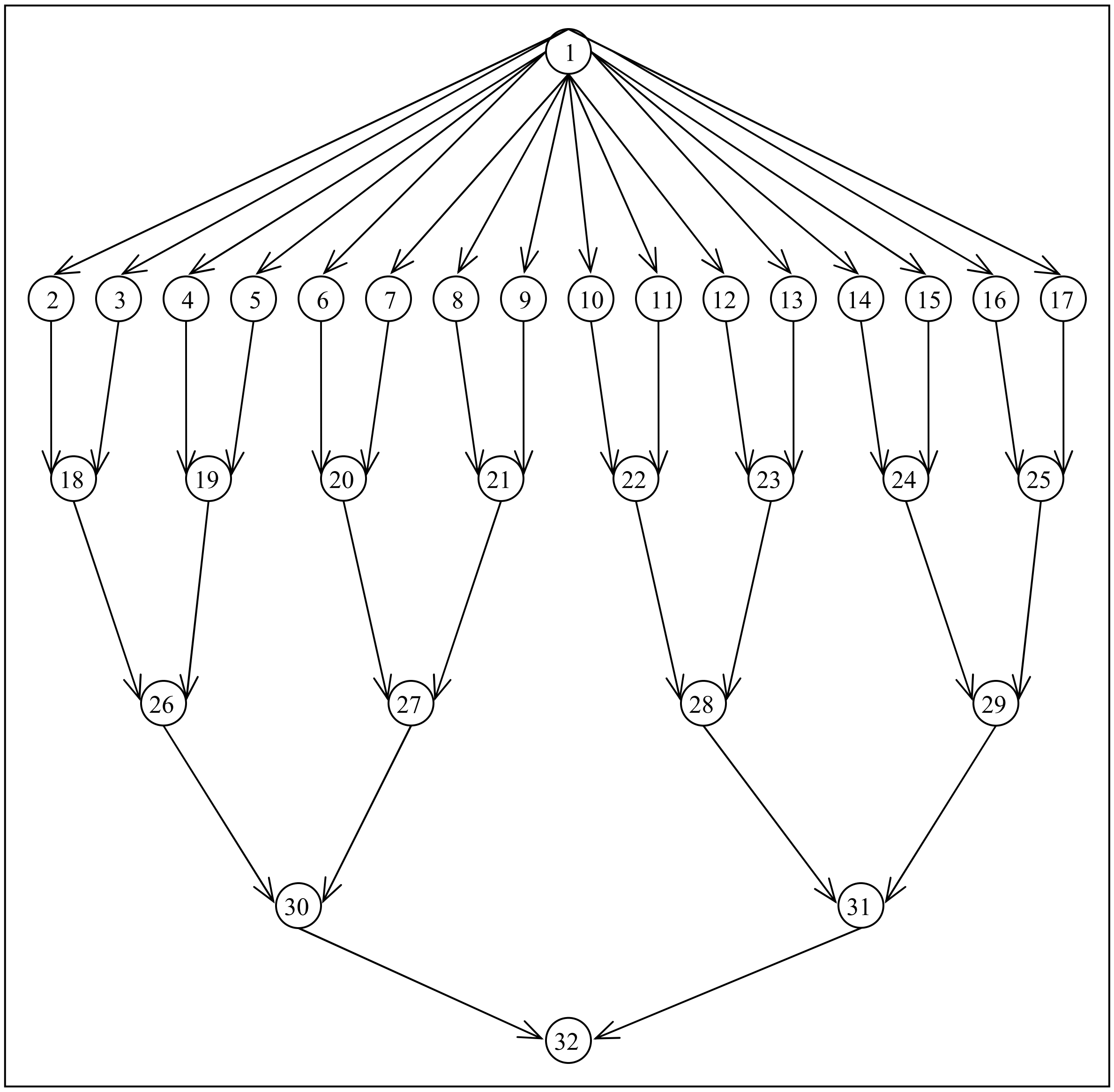


Figure 6.29. Sieve of Eratosthenes task graph for *N* = *32*

In the simulation, we used the graph of Sieve of Eratosthenes for a sequence of 32 numbers (*N = 32)* to test both EADAGS and EAGS-D. Since the structure of the graph is fixed, only the number of processors and the *CCR* values were changed. For a sequence of 32 numbers, the total number of tasks in the graph is 32 nodes and the largest number of tasks at a single level is 16 tasks, so the number of processors was bounded to 8 processors. *CCR* had five different values: 0.1, 0.5, 1.0, 5.0, and 10. The amount of energy consumed was measured for the five different operating strategies explained earlier in Chapter 5.

6.2.4.1 Results for EADAGS

Figure 6.30 shows a decrease in the average energy savings with increasing number of processors for Sieve of Eratosthenes with EADAGS over DPS. This is because increasing number of processor allows several parallel task executions, thus minimizing the wait times, which are used by our algorithm to save energy.

The average energy savings measured were 23% for 2 processors in the 5V/off technique, 19% for 2V during idle, 13% for 3.3V during idle, 21% for 2V scale, and 14% for 3.3V scale. When three processors are used, the average energy was 41% for the 5V/off technique, 33% for 2V during idle, 33% for 3.3V during idle, 35% for 2V scale, and 25% for 3.3V scale. For eight processors, the average energy savings are 65%, 54%, 36%, 56%,a and 7% for the 5V/off, 2V during idle, 3.3V during idle, 2V scale, and 3.3V scale respectively. Table 6.22 lists the *makespan* and the average energy savings for each different number of processors tested.

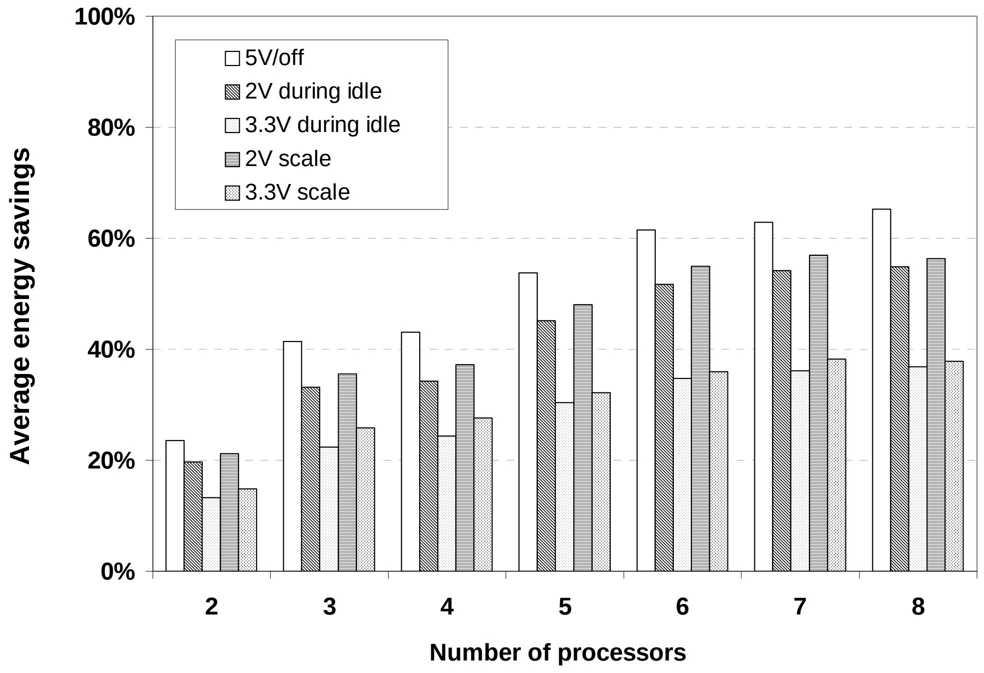


Figure 6.30. Average energy savings for different number of processors for Sieve of Eratosthenes with EADAGS

Number of processors	*Makespan*	Percentage of energy savings				
		5V/off	2V during idle	3.3V during idle	2V Scale	3.3V Scale
2	502.4	23.57 %	19.73 %	13.26 %	21.19%	14.89%
3	472	41.35 %	33.14 %	22.37 %	35.58%	25.82%
4	384.8	43.12 %	34.22 %	24.34 %	37.18%	27.62%
5	382.4	53.78 %	45.18 %	30.36 %	47.97%	32.20%
6	382.4	61.49 %	51.65 %	34.7 %	54.98%	35.91%
7	382	62.89 %	54.19 %	36.16 %	56.91%	38.22%
8	391.2	65.26 %	54.82 %	36.83 %	56.33%	37.83%

Table 6.22. *Makespan* and average energy savings for different number of processors for Sieve of Eratosthenes with EADAGS

Figure 6.31 plots the average energy savings with respect to *CCR*. The average energy savings increased with increasing *CCR*. When *CCR* increases, processors incur longer idle times due to communication between tasks. Our algorithm is able to use such idle times to achieve energy savings. The average energy savings over DPS ranges from 29% for *CCR* = 0.1 to 77% when *CCR* = 10 for 5V/off technique. Savings are smaller for 2V during idle; they range between 24% and 65%. Savings are even smaller for the 3.3V during idle; they are 16% for *CCR* = 0.1 and 43% for *CCR* = 10, while for 2V scale and 3.3V scale the average energy savings over DPS ranges from 31% for *CCR* = 0.1 to 65% when *CCR* = 10 and 20% for *CCR* = 0.1 and 45% for *CCR* = 10 respectively. Table 6.23 lists the average energy savings and the *makespan* for EADAGS over DPS for different *CCR* values.

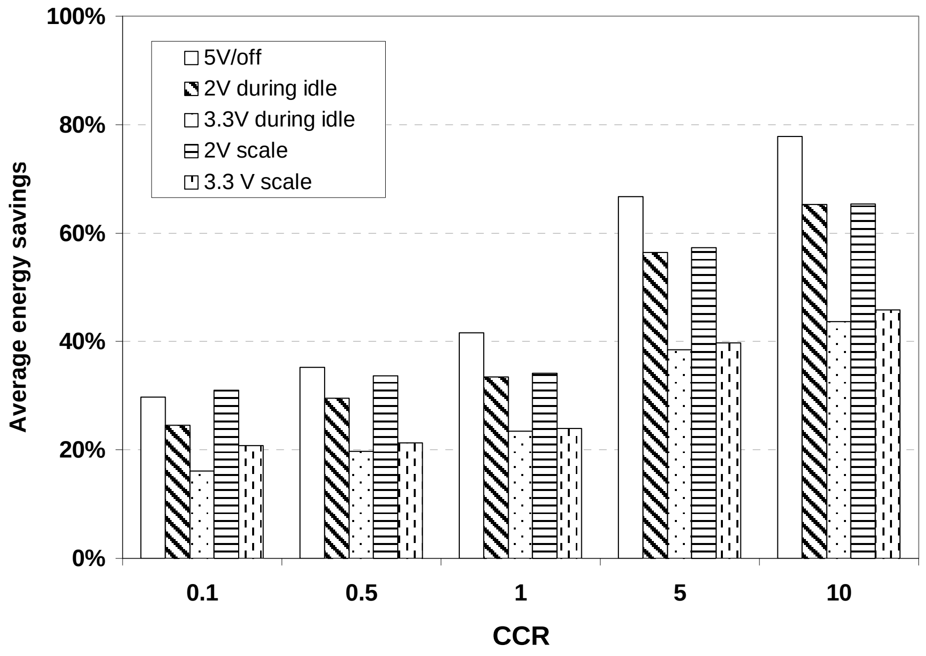


Figure 6.31. Average energy savings for different *CCR* values for Sieve of Eratosthenes with EADAGS

CCR	*Makespan*	Percentage of energy savings				
		5V/off	2V during idle	3.3V during idle	2V Scale	3.3V Scale
0.1	233.71	29.69 %	24.53 %	16.05 %	31.01%	20.84%
0.5	244.29	35.23 %	29.5 %	19.75 %	33.69%	21.29%
1	268.57	41.57 %	33.49 %	23.46 %	34.19%	23.97%
5	505.71	66.74 %	56.4 %	38.47 %	57.28%	39.77%
10	817.14	77.81 %	65.3 %	43.7 %	65.34%	45.84%

Table 6.23. *Makespan* and average energy savings for different *CCR* values for Sieve of Eratosthenes with EADAGS

6.2.4.2 Results for EAGS-D

Figure 6.32 and Figure 6.33 show the average energy savings for EAGS-D with respect to the number of processors and *CCR* values respectively for the DAG for Sieve of Eratosthenes.

Figure 6.32 shows an increase in the average energy savings with increasing number of processors. The average energy savings measured were 12% for 2 processors using the 5V/off technique, 10% for 2V during idle, 6% for 3.3V during idle, 13% for the 2V scale, and 8% for the 3.3V scale. The average energy savings increases to 63%, 53%, 35%, 54%, and 37% when 8 processors are used for the 5V/off technique, 2V during idle, 3.3V during idle, 2V scale, and 3.3V scale respectively. The *makespan* and the average energy savings for different numbers of processors are listed in Table 6.24.

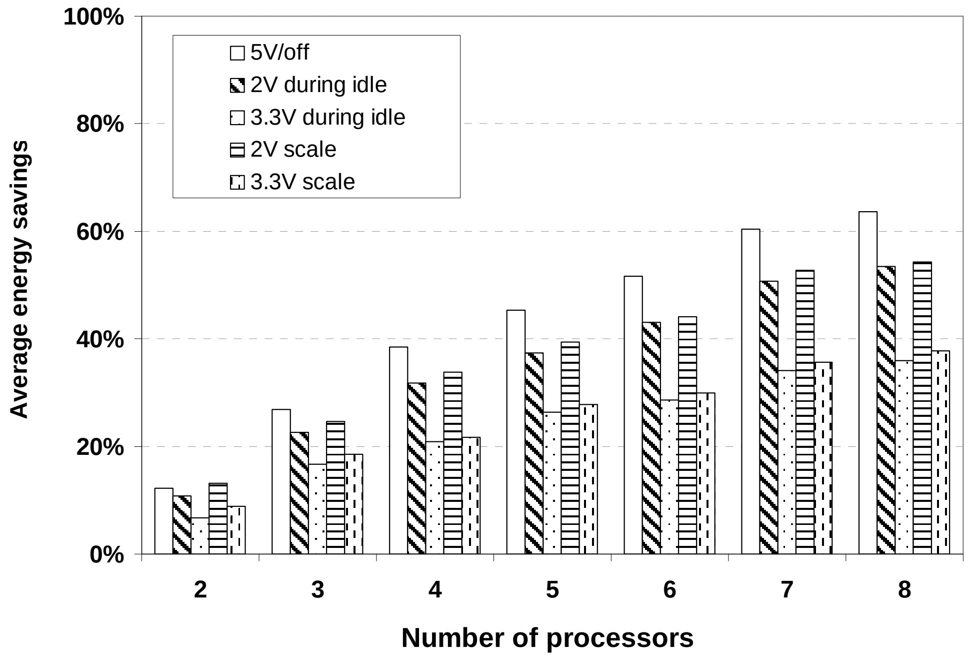


Figure 6.32. Average energy savings for Sieve of Eratosthenes with EAGS-D with different numbers of processors

Number of processors	*Makespan*	Percentage of energy savings				
		5V/off	2V during idle	3.3V during idle	2V Scale	3.3V Scale
2	416	12.27 %	10.75 %	6.71 %	13.16%	8.81%
3	354.4	26.93 %	22.56 %	16.72 %	24.63%	18.49%
4	314.4	38.49 %	31.78 %	20.84 %	33.84%	21.67%
5	304	45.34 %	37.39 %	26.35 %	39.45%	27.76%
6	288.8	51.67 %	43.03 %	28.59 %	44.10%	29.99%
7	284	60.37 5	50.71 %	34.07 %	52.74%	35.60%
8	276.8	63.65 %	53.47 %	35.92 %	54.30%	37.79%

Table 6.24. *Makespan* and average energy savings for different numbers of processors for Sieve of Eratosthenes with EAGS-D

Figure 6.33 plots the average energy savings with respect to different *CCR* values. The average savings increased with increasing *CCR*. The average energy savings over HNPD ranges from 29% for *CCR* = 0.1 to 62% when *CCR* = 10 for 5V/off technique. Savings are smaller for 2V during idle; they range between 25% and 51%. Savings are even smaller for the 3.3V during idle; they are 16% for *CCR* = 0.1 and 35% for *CCR* = 10, while for 2V scale and 3.3V scale the average energy savings over HNPD ranges from 30% for *CCR* = 0.1 to 51% when *CCR* = 10 and 21% for *CCR* = 0.1 and 35% for *CCR* = 10 respectively. The *makespan* and the average energy savings for different *CCR* values are listed in Table 6.25.

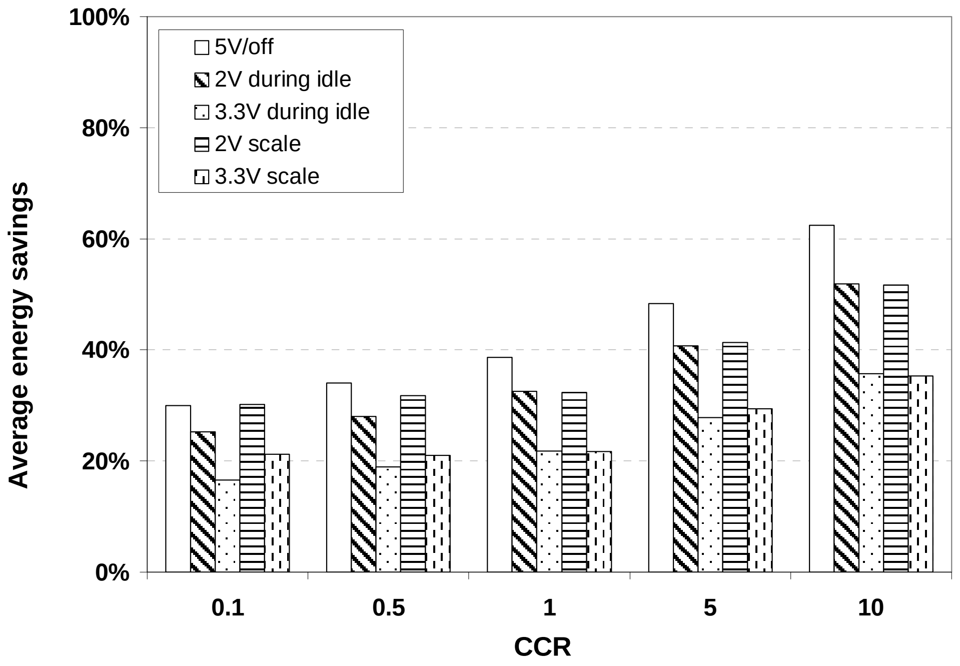


Figure 6.33. Average energy savings for Sieve of Eratosthenes with EAGS-D with different *CCR* values

CCR	*Makespan*	Percentage of energy savings				
		5V/off	2V during idle	3.3V during idle	2V Scale	3.3V Scale
0.1	236	29.94 5	25.23 %	16.59 %	30.15%	21.21%
0.5	242.86	34.05 %	28.02 %	18.98 %	31.77%	21.03%
1	257.14	38.66 %	32.5 %	21.8 %	32.32%	21.71%
5	342.86	48.32 %	40.72 %	27.77 %	41.28%	29.37%
10	520	62.38 %	51.87 %	35.74 %	51.71%	35.34%

Table 6.25. *Makespan* and average energy savings for different *CCR* values for Sieve of Eratosthenes with EAGS-D

Figure 6.34 shows the average energy savings for both EADAGS and EAGS-D with respect to the five voltage scaling strategies tested for Sieve of Eratosthenes. The overall average energy savings for EAGS-D is lower than the average energy savings measured for EADAGS. That is due to the nature of the EAGS-D algorithm, which uses task duplication to minimize the *makespan*. That duplication uses a big portion of the processor's idle time and that reduces the amount of energy savings.

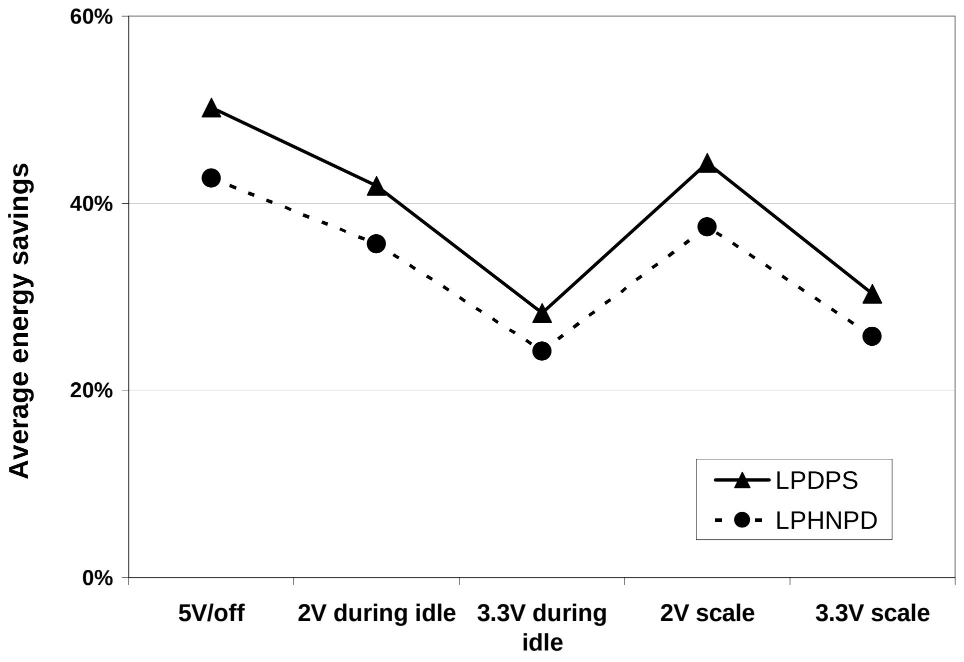


Figure 6.34. Average energy savings for Sieve of Eratosthenes with EADAGS and EAGS-D with voltage scaling levels

CHAPTER 7

CONCLUSIONS

We have proposed two new scheduling algorithms, EADAGS and EAGS-D, which try to minimize finish time as well as energy consumption by the use of dynamic voltage scaling.

The results were based on a two part software simulation study. The first part consists of a large set of randomly generated DAGs with various characteristics such as number of nodes, *CCR*, shape parameter, processor node ratio, and out degree. For each parameter the results were averaged across all other variables. The total number of random DAG generated for evaluating each algorithm were 10,800 graphs for each.

The second part of the simulation contained DAGs for real world problems namely; Gaussian elimination, molecular dynamic code, Sieve of Eratosthenes, and Fast Fourier transform. These DAGs has a specific structure so numbers of nodes, shape parameter, and out degree are fixed. We tested both EADAGS and EAGS-D with different number of available processors and *CCR*.

The results from the randomly generated DAGs showed that EADAGS algorithm resulted in an average energy saving of 40% over simple DPS, while EAGS-D algorithm estimated a reduction in energy by 28% which is less than that for EADAGS due to the nature of EAGS-D algorithm which involves task duplication.

For the second test set, first for Gaussian elimination an average of 44% of energy savings were achieved by EADAGS and an average of 37% by EAGS-D. For the molecular dynamic code problem the average energy savings with EADAGS and EAGS-D were 46% and 38% respectively. For FFT the average energy savings measured were 42% and 33% for EADAGS and EAGS-D respectively. While for the Sieve of Eratosthenes average energy savings for EADAGS and EAGS-D were 40% and 36% respectively.

The effect of different DAG characteristic on the amount of energy savings is discussed next. For both EADAGS and EAGS-D the amount of energy savings increased by increasing the number of nodes due to the increase in idle time due to task dependency. The rate of the increase is lower in EAGS-D than EADAGS due to task duplication. The increase of CCR resulted in an increase in the average energy savings for both EADAGS and EAGS-D. When CCR increases, processors incur longer idle times due to communication between tasks. Both algorithms were able to use such idle times to achieve energy savings. The results showed that the overall energy savings marginally increased with increasing the shape parameter. Increasing shape parameter increases parallelism in the DAG resulting in more idle time for the processors due to the task dependency. This time was used by both EADAGS and EAGS-D to reduce the consumed energy. The last parameter tested was out degree. 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 time for all processors and so less energy to save.

The future work can involve applying the voltage scaling technique to other scheduling algorithm. Aiming to find the optimal solution especially to the real world problem and then physically implementing them to save energy.

BIBLIOGRAPHY

[Bask00] S. Baskiyar, “Scheduling DAGs on message passing m-processor systems,” *IEICE Transactions on Information and Systems*, vol. E-38-D, July 2000.

[Bask03] S. Baskiyar, and C. Dickinson, “Scheduling Directed A-cyclic Graphs on a Bounded Set of Heterogeneous Processors Using Task Duplication,” *LNCS*, vol. 2913, pp. 259-267, Springer-Verlag, 2003.

[Burd96] T. Burd and R. Brodersen, “Processor Design for Portable Systems,” *Journal of VLSI Signal Processing*, 13(2-3), pp.203-222, 1996.

[Chae04] Chaeseok Im and Soonhoi Ha, “Dynamic Voltage Scaling for Real-time Multi-task Scheduling using Buffers,” *Proc. ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*, pp.88-94, 2004.

[Chan96] A. Chandrakasan, V. Gutnik and T. Xanthopoulos, “Data Driven Signal Processing: An Approach for Energy Efficient Computing,” *International Symposium on Low Power Electronics and Design*, pp. 347-352, Aug. 1996.

[Choi04] K. Choi, R. Soma, and M. Pedram, “Dynamic Voltage and Frequency Scaling based on Workload Decomposition,” *Proceedings of the 2004 International Symposium on Low Power Electronics and Design*, August 2004.

[Chun92] Y. Chung and S. Ranka, “Applications and Performance Analysis of a Compile-Time Optimaization Approach for List Scheduling Algorithms on Distributed Memory Multiprocessors,” *Proc. Supercomputing*, pp. 512-521, Nov 1992.

[Corm01] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, “Introduction to Algorithms,” *MIT Press*, 2001.

[Dong01] J. J. Dongarra and D. W. Walker, “The Quest for Petascale Computing,” *In IEEE Transactions on Computing in Science and Engineering*, pp. 32-39, May 2001.

[Gebo96] C. H. Gebotys and R. J. Gebotys, “Power-Minimization in Heterogeneous Processing,” *In Proceedings of 29th Hawaii International Conference on System Sciences (HICSS’96)*, vol. 1, pp. 330-337, January 1996.

[ImHa04] C. Im and S. Ha, "Dynamic Voltage Scaling for Real-time Multi-task Scheduling using Buffers," *Proc. ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*, pp.88-94, 2004.

[Iver98] M. A. Iverson and F. Ozguner, "Dynamic Competitive Scheduling of Multiple DAGs in a Distributed Heterogeneous Environment," *In Proceedings of the 1998 Workshop on Heterogeneous Processing*, pp. 70-78, March 1998.

[Khok93] A. Khokhar, V. K. Prasanna, M. E. Shaaban and C. Wang, "Heterogeneous Computing: Challenges and Opportunities," *IEEE Transactions on Computers*, vol. 26, pp. 18- 27, June 1993.

[Kiro97] D. Kirovski and M. Potkonjak, "System-level Synthesis of Low-power Hard Real-time Systems," In *Proceedings of the 34th Annual Conference on Design Automation*, 1997.

[Kump94] K. Li, R. Kumpf, P. Horton, and T. Anderson, "A Quantitative Analysis of Disk Drive Power Management in Portable Computers," *PTUC. Winter 1994 USENIX Conference*, pp. 279-292, Jan. 1994.

[Kwok99] Y. K. Kwok and A. Ishfaq, "Link Contention-Constrained Scheduling and Mapping of Tasks and Messages to a Network of Heterogeneous Processors," *In Proceedings of 1999 International Conference on Parallel Processing*, pp. 551-558, September 1999.

[Mart02] S. Martin, K. Flautner, T. Mudge, and D. Blaauw, "Combined Dynamic Voltage Scaling and Adaptive Body Biasing for Lower Power Microprocessors under Dynamic Workloads," *Proceeding of the 2002IEEE/ACM International Conference on Computer-aided Design*, November 2002.

[Lu00] Y. H. Lu, L. Benini and G. Di Micheli, "Low-Power Task Scheduling for Multiple Devices," *International Workshop on Hardware/Software Codesign*, pp. 39- 43, May 2000.

[Micr04] "OnNow Power Management Architecture for Applications" *at* *http://www.microsoft.com/hwdev/desinit/onnowapp.HTM*

[Mish03] R. Mishra, N. Rastogi, D. Zhu, D. Mossé, and R. Melhem, "Energy Aware Scheduling for Distributed Real-Time Systems," *Proc. Int'l Parallel and Distributed Processing Symposium*, pp. 9-16, April 2003.

[Much97] S. Muchnick, "Advanced Compiler Design and Implementation," *Morgan Kaufmann Publishers*, Inc., 1997.

[Pouw01] J. Pouwelse, K. Langendoen, and H. Sips, “Dynamic Voltage Scaling on a Low-Power Microprocessor,” *Proc. of the 7th Annual International Conference on Mobile Computing and Networking*, pp. 251-259, July 2001.

[Radu00] A. Radulescu and A. J. C. Van Gemund, “Fast and Effective Task Scheduling in Heterogeneous Systems,” *In 9th Heterogeneous Computing Workshop*, pp. 229-239, May 2000.

[Reut97] C. Reuter, M. Schwiegershausen and P. Pirsch, “Heterogeneous Multiprocessor Scheduling and Allocation using Evolutionary Algorithms,” *In Proceedings of the IEEE International Conference on Application-Specific Systems*, Architecture and Processors, pp. 294-303, July 1997.

[Seig97] M. Tin, H. J. Seigel, J. K. Antonio and Y. A. Li, “Minimizing the Application Execution Time Through Scheduling of Subtasks and Communication Traffic in a Heterogeneous Computing System,” *In IEEE Transactions on Parallel and Distributed Systems*, vol. 8, no. 8, pp. 857-870, August 1997.

[Shan03] L. Shang, Li-Shiuan Peh, and N. K. Jha, “Dynamic Voltage Scaling with Links for Power Optimization of Interconnection Networks,” *Proc. of the 9th International Symposium on High-Performance Computer Architecture*, pp. 91-102, February 2003.

[Shin01] D. Shin, S. Lee, and J. Kim, “Intra-Task Voltage Scheduling for Low-Energy Hard Real-Time Applications,” *IEEE Design and Test of Computers*, vol. 18, pp. 20-30, March 2001.

[Shiu00] W. T. Shiue and C. Chakrabarti. “Low-Power Scheduling with Resources Operating at Multiple Voltages,” *In IEEE Transactions on Circuits and Systems-2: Analog and Digital Signal Processing*, vol. 47, no. 6, pp. 536- 543, June 2000.

[Smar06]http://www.bsac.eecs.berkeley.edu/archive/users/warneke-brett/SmartDust/, accessed on February 2006.

[Sriv96] M. Srivastava, A. Chandrakasan, and R. Brodersen, “Predictive System Shutdown and Other Architectural Techniques for Energy Efficient Programmable Computation,” *IEEE Transactions on VLSI Systems*, vol. 4, pp. 42-55, March 1996.

[Tiwa94] V. Tiwari, S. Malik, and A. Wolfe, “Power Analysis of Embedded Software: A First Step toward Software Power Minimization,” *Proceedings of the 1994 IEEE/ACM International Conference on Computer-aided Design*, November 1994.

[Tiwa96] V. Tiwari, S. Malik, A. Wolfe, and M. Lee, “Instruction Level Power Analysis and Optimization of Software,” *Journal of VLSI Signal Processing*, 13(2/3), pp. 1-18, 1996.

[Topc99] H. Topcuoglu, S. Hariri and M. Y. Wu, “Task Scheduling Algorithms for Heterogeneous Processors,” *In Proceedings of the 8th Heterogeneous Computing Workshop***,** pp. 3-14, April 1999.

[Topc02] H. Topcuoglu, S. Hariri and M. Y. Wu, “Performance-effective and Low Complexity Task Scheduling for Heterogonous Computing Parallel and Distributed Systems,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, March 2002.

[Wang97] L. Wang, H. J. Siegel, V. P. Rowchowdhury and A. A. Maciejewski, “Task Matching and Scheduling in Heterogeneous Computing Environments Using a Genetic- Algorithm-Based Approach,” *Journal of Parallel and Distributed Computing*, vol. 47, pp. 8-22, November 1997.

[Weis94] M. Weiser, B. Demers, and Shenker, “Scheduling for reduced CPU energy,” *Proc 1st USENIX Symposium On operating Systems Design and Implementation*, pp. 13-23, Nov 1994.

[Yang01] P. Yang, C. Wong, P. Marchal, F. Catthoor, D. Desmet, D. Verkest, and R. Lauwereins, “Energy-Aware Runtime Scheduling for Embedded-Multiprocessor SOCs,” *IEEE Design and Test of Computers*, vol. 18, no. 5, pp. 46-58, Sept./Oct. 2001.

[Zhan02] Y. Zhang, D. Chen, “Task Scheduling and Voltage Selection for Energy Minimization,” *Design Automation Conference*, pp.183-188, June 2002.