Energy Aware Scheduling for DAG Structured Applications on Heterogeneous and DVS Enabled Processors

Venkateswaran Shekar and Baback Izadi

Dept. of Electrical and Computer Engineering
State University of New York at New Paltz
New Paltz, NY, 12561
bai@engr.newpaltz.edu

***Abstract*—The trend towards ever more powerful and faster processors has led to an enormous increase in power consumption. This paper focuses on scheduling tasks in a heterogeneous environment with DVS enabled processors to minimize both execution time and energy consumed. The proposed algorithm, called Energy-Dynamic Level Scheduling (EDLS), favors low-energy consuming processors by introducing a cost factor that affects scheduling decisions. Our scheme allows for trade offs between energy consumption and the desired performance. Our simulation results exhibit significant power savings at a reasonable increase in overall execution time. Moreover, our results demonstrates a high degree of correlation between the energy saving and the increase in the heterogeneity of processors.**

***Keywords*-Energy Aware; Heterogeneous Computing; DVS; Scheduling; DAG;**

I. Introduction

Today's data centers are hardly scalable. They are simply too large and consume too much power. They have huge footprint and consume several megawatts (MW) of power; One megawatt costs about one million dollars per year. Several studies have revealed that improving energy and power efficiency is the most formidable challenge facing the development of new data centers and exaflop multiprocessor systems[1]. Conceiving and building such systems requires reducing power by three order of magnitude and therefore pose technical challenges at all levels of the computing stack, including circuits, architecture, software systems, and applications. Among the schemes that researchers have examined to control power are processor throttling also known as Dynamic Voltage Scaling (DVS). A processor typically consumes from third to half of the node's total power [2]. Today's modern processors have the ability to operate at different voltages and switch between them dynamically. Consequently, Dynamic Voltage Scaling technologies such as Intel SpeedStep and AMD PowerNOW! has provided reduced power consumption and prolonged battery life for laptops and handheld devices.

In this paper, we consider schemes aim at reducing dynamic power of a processor. Total power consumed by a processor is the sum of the static and dynamic power dissipation. The processor's dynamic Power is given by[3]

$$P = C_{ef} \times V_{dd}^2 \times f \quad (1)$$

where C_{ef} is the effective switching capacitance, V_{dd} is the supply voltage and f is the processor clock frequency. The processor clock frequency is linearly related to the supply voltage $f = k \times (V_{dd} - V_t)^2/V_{dd}$, where k is a constant and V_t is the threshold voltage. Hence, the energy consumed by a processor to execute task T_i is $E_i = C_{ef} \times V_{dd}^2 \times Cy_i$, where Cy_i is the number of cycles required to execute the task. Since decreasing the processor speed correlates linearly with decreasing the voltage supply, it reduces the power consumed cubically and energy quadratically, but at the cost of linearly increasing the task's latency.

Task scheduling to meet performance parameters such as time and power is an NP-Complete problem[4]. Therefore, many heuristics have been developed for real-time scheduling algorithms [5], [6], [7], [8]. Scheduling tasks in a heterogeneous environment presents additional constraints due to different performance and energy management characteristics of different processors and cores. On the up side, a heterogeneous computing environment does provide a significant opportunity to meet today's mandated performance and power requirements. Therefore, researchers [9], [10] have explored energy-efficient scheduling for heterogeneous systems. Unfortunately, these algorithms mainly focus on minimizing the energy consumption, while the execution time becomes secondary.

Our scheme is an extension of Sih and Lee's [11] earlier scheduling algorithm called the Dynamic Level Scheduling (DLS) algorithm. The DLS algorithm is shown especially effective when selecting the task and processor at the same time [11]. A number of researchers have implemented variation of the DLS algorithm [12], [13]. Our scheme, called the Energy Dynamic Level Scheduling (EDLS), utilizes both time and energy to make scheduling decision. The resultant energy saving can have some adverse effect on the overall execution time. However, our scheme does provide a control to tradeoff between the execution time and energy saving.

The rest of the paper is organized as follows. Notation and definitions are given in the next section. In Section III,

978-1-4244-7614-5/10/$26.00

we review the DLS algorithm through an example case. Section IV and V describe the EDLS and Measured EDLS schemes, and show the relationship between energy consumption and the scheduling of tasks. In Section VI, we outline and analyze our experimental results. Finally, concluding remarks are given in Section VII.

II. Notation and Definitions

We make the following assumptions. Applications have DAG form and are periodic. Moreover, the system consists of a network of interconnected heterogeneous processors. These processors run at a single speed throughout the application. Figure 1 shows a typical DAG application $G = (T, E)$, where each node represents a task $T_i \in T$ and each weighted directed edges $E_{ij} = (T_i, T_j) \in E$ represents precedence execution and communication between tasks T_i and T_j. The computing environment consists of a pool of heterogeneous processors $P = \{P_1, P_2, P_3, ..., P_n\}$, with the ability to run at different discrete speeds. For example, processor P_i can run at speeds $S_{Pi} = \{S_1, S_2, ..., S_n\}$, where S_1 is the fastest speed and S_n is the slowest speed. We note $P_x@S_y$ as processor x running at speed y.

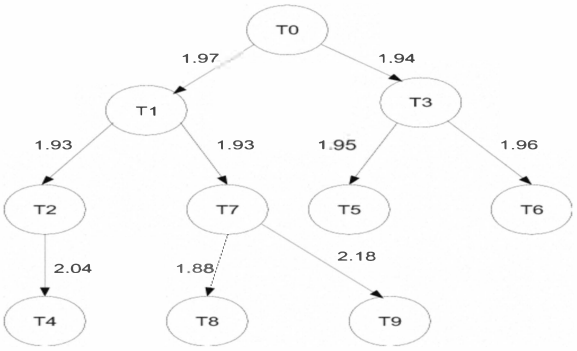


Figure 1. Directed acyclic graph of test case

III. DLS Algorithm

The Dynamic Level Scheduling Algorithm (DLS) is developed by Sih and Lee [11]. For the sake of completion, in this section, we briefly describe it through an example case. Figure 1 illustrates a DAG with 10 tasks labeled T_0 to T_9 with their dependencies. Let's consider a pool of 3 processors $P = \{P_1, P_2, P_3\}$, where the speeds are $S_{P1} = \{S_1\}$ and $S_{P2} = \{S_1, S_2\}$ and $S_{P3} = \{S_1, S_2, S_3\}$, respectively. The relative speed-power characteristics of these processors are as depicted in Figure 2. Our power and speed models for different processors is rather simple, but effective. In fact, we are not interested in accurate simulations of the real consumed power or execution time, but rather, in a comparative analysis among different algorithms. Since decreasing the processor speed correlates linearly with decreasing the voltage supply, from Equation 1, it follows that power consumed is cubically correlated with the speed of the processor. Hence, the values of the execution time and the consumed power of tasks on each processor can be represented using Equation 2.

$$P \propto \frac{1}{t^3} \tag{2}$$

Subsequently, for our example in Figure 2, we have chosen a family of processors with three power settings, consistent with the existing technology [14]. Moreover, we have chosen the typical execution of tasks within the fastest processor to be about 10 ms. Using Equation 2, we can estimate the execution of the processors at different power setting by noting that:

$$\frac{P_1}{P_2} \propto \frac{{t_2}^3}{{t_1}^3} \tag{3}$$

Note that, $P_3@S_3$, $P_2@S_2$, and $P_1@S_1$ are given similar execution time and power characteristics. Likewise, $P_3@S_2$ and $P_2@S_1$ have similar characteristics, but faster execution time (lower power consumption) than the first group. Finally, $P_3@S_1$ is chosen as the fastest and therefore least power efficient processor.

The execution time and consumed power for each of the 10 tasks of Figure 1 for Processors 1, 2 and 3 are based on the nominal values of Figure 2. Individual values of the tasks have been randomized within $\pm 10\%$ of the nominal values, and are specified in Tables I and II, respectively.

Figure 2. Processor Pool

Table I
Details of Processor 1 and Processor 2

	Processor 1		Processor 2			
Task Number	Exec Time $@S_1$(ms)	Power $@S_1$(W)	Exec Time $@S_1$(ms)	Power $@S_1$(W)	Exec Time $@S_2$(ms)	Power $@S_2$(W)
0	15.74	4.63	11.11	13.89	15.59	4.58
1	15.95	4.69	11.25	14.06	15.63	4.59
2	16.89	4.96	12.10	15.13	17.06	5.02
3	16.2	4.76	11.44	14.31	16.01	4.71
4	17.53	5.15	12.43	15.54	17.56	5.16
5	15.76	4.63	11.03	13.78	15.90	4.67
6	16.29	4.79	11.41	14.26	16.20	4.76
7	15.71	4.62	11.21	14.02	15.80	4.64
8	16.68	4.90	11.58	14.47	16.75	4.92
9	17.4	5.11	12.19	15.24	17.20	5.05

The DLS algorithm is designed to schedule a DAG onto a set of heterogeneous processors in order to minimize the execution time of the application. The algorithm considers

Table II
DETAILS OF PROCESSOR 3

Task Number	Exec Time @S_1(ms)	Power @S_1(W)	Exec Time @S_2(ms)	Power @S_2(W)	Exec Time @S_3(ms)	Power @S_3(W)
0	9.26	24.26	11.01	13.76	15.79	4.64
1	9.2	24.2	11.01	13.76	15.9	4.68
2	10.05	25.05	12.12	15.15	16.97	4.99
3	9.54	24.54	11.41	14.27	15.97	4.7
4	10.33	25.33	12.4	15.5	17.5	5.15
5	9.36	24.36	11.22	14.03	15.81	4.65
6	9.39	24.39	11.51	14.39	16.08	4.73
7	9.2	24.2	11.1	13.88	15.81	4.65
8	9.78	24.78	11.73	14.66	16.75	4.93
9	10.2	25.2	12.39	15.49	17.53	5.16

the execution time of the tasks as well as the interprocessor communication overhead, while mapping the tasks onto the processors. The algorithm determines when it is appropriate to make matching and scheduling decisions and not when to schedule a particular task. There are many processor-speed combinations that one can choose from. In this example, let's consider $P_1@S_1$, $P_2@S_1$, $P_3@S_1$. In other words all three processors are run at maximum speed throughout the application.

At each scheduling step, the DLS algorithm chooses the next task to schedule and the processor on which the task is to be executed. This is done by finding the Ready Task and processor pair that have the highest cost function, called Dynamic Level, and specified by Equation 4.

$$DL_{np} = SL_{np} - max(DA_{np}, TF_{np}) + \Delta \qquad (4)$$

DL_{np} = Dynamic Level of Task n on Processor p
SL_{np} = Static Level of a task n on Processor p
DA_{np} = Data Ready time n on Processor p
TF_{np} = Processor Ready time n on Processor p
Δ = Processor speed difference

All the terms in Equation 4 are expressed in time units. DL represents how well the task and processor are matched. SL is the largest sum of the execution times along a directed path for a task T_i to an end task over all end tasks. SL gives priority to tasks that are farther away from the end task(s). DA is the earliest time that all the data required by a task is available at the processor and TF represents the time that the last task assigned to the processor finishes execution. The maximum term between DA and TF is chosen so the task which takes longer time to be ready is penalized. Finally, Δ accounts for the speed difference between the processors, allowing the processors with higher Δ to process the task faster.

As the first step of the DLS Algorithm, the Static Level is calculated based on the median execution time of tasks among different processors. For example, from Tables I and II, the median execution time of $T_1 = Median(15.95, 11.25, 9.2) = 11.25$, as specified in Table III. Now, let's consider Task 1 in Figure 1. The three possible paths to the end tasks and the sum of their median execution time are

$T_1 \rightarrow T_2 \rightarrow T_4 \Rightarrow 11.25 + 12.1 + 12.44 = 35.79ms$
$T_1 \rightarrow T_7 \rightarrow T_8 \Rightarrow 11.25 + 11.22 + 11.58 = 34.05ms$
$T_1 \rightarrow T_7 \rightarrow T_9 \Rightarrow 11.25 + 11.22 + 12.195 = 34.665ms$

The DLS picks the path with the largest value (35.79ms) and assigns it as the Static Level of Task 1. Similarly, Static Level of other tasks are calculated and the result is tabulated in Table III.

Table III
STATIC LEVEL TABLE

Task Number	Median Times	Static Level
0	11.11	46.90
1	11.25	35.79
2	12.1	24.537
3	11.44	22.85
4	12.43	12.43
5	11.03	11.03
6	11.41	11.41
7	11.21	23.41
8	11.58	11.58
9	12.19	12.19

In the beginning, the only Ready Task is Task 0 and all three processors are available for execution. This implies that the Processor Ready Time (TF) and Data Ready Time (DA) are zero. The DLS picks the task-processor pair with the highest Dynamic Level. The Dynamic Levels for Task 0 and three processors are calculated using Equation 4 and the results are shown in Table IV. From the table, Task 0

Table IV
STEP 1 OF DLS ALGORITHM

Task	SL	DA	TF	Δ	DL
Processor 1					
0	46.90	0.0	0.0	-4.63	42.27
Processor 2					
0	46.90	0.0	0.0	0.0	46.90
Processor 3					
0	46.90	0.0	0.0	1.85	48.75 $\Leftarrow$

has the highest Dynamic Level value for Processor 3, and therefore, Task 0 is assigned to Processor 3.

Following Task 0, per Figure 1, Task 1 and Task 3 are ready to be scheduled. Processor Ready Time (TF) for Processors 1 and 2 are 0 since no task was assigned to them in the previous step. The Data Ready Time (DA) for Task 1 on either Processors 1 and 2 is 9.26 + 1.97 = 11.23 ms, where 9.26 ms is the execution time for Task 0 on Processor 3, from the previous scheduling step, and 1.97 ms is the communication overhead between Tasks 0 and 1. Similarly, DA for Task 3 for either Processors 1 and 2 is 9.54 + 1.94 = 11.48 ms. For Processor 3, both TF and DA are 9.26 ms since there is no communication overhead if tasks remain in the same processor. The result of Step 2 is shown in Table V. Hence, Task 1 is assigned to Processor 3.

By repeating the process, the rest of the tasks are assigned to the processors, as shown in Figure 3. The total energy

Table V
STEP 2 OF DLS ALGORITHM

Task	SL	DA	TF	Δ	DL
Processor 1					
1	35.79	11.23	0.0	-4.69	19.86
3	22.85	11.48	0.0	-4.75	6.90
Processor 2					
1	35.79	11.23	0.0	0.0	24.56
3	22.85	11.48	0.0	0.0	11.65
Processor 3					
1	35.79	9.26	9.26	2.06	28.59 ⇐
3	22.85	9.26	9.26	1.90	15.50

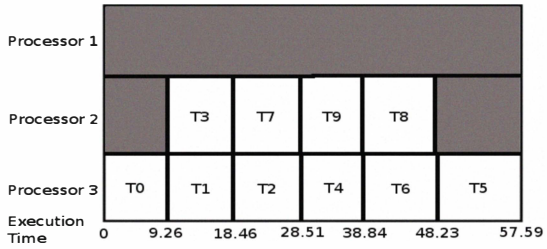


Figure 3. Scheduling using DLS algorithm

consumed is 2.092 Joules, which is determined by adding the consumed energy of individual scheduled task-processor pairs, as specified by Tables I and II. From Figure 3, all tasks are assigned to processors 2 and 3. This is because the DLS algorithm favors the task-processor pairs with shortest execution time and Processor 1 is a slower (albeit more energy efficient) processor per Figure 2.

IV. EDLS ALGORITHM

In this section, we present an energy-efficient DLS algorithm (*EDLS*). This is done by modifying the DL cost function to favor processors with low-power capability. Hence, we introduce a new Energy Dynamic Level (EDL) for Task n on Processor p.

$$EDL_{np} = DL_{np} + DL_{np} \times (1 - \alpha_{np}) \quad (5)$$

$$DL_{np} = SL_{np} - max(DA_{np}, TF_{np}) + \Delta_{np} \quad (6)$$

The second term in Equation 5 is added to favor scheduling tasks on processors with lower energy consumption. Specifically,

$$\alpha_{np} = \frac{Task\ n\ Energy\ on\ Processor\ p}{Max\ Energy\ by\ task\ n\ over\ all\ processors}$$

Note that α_{np}, for the task-processor pair with the highest consumed energy, would be 1, resulting in $EDL = DL$. Other task-processor pairs, with lower consumed energy, result in $\alpha_{np} < 1$. Subsequently, the lower value of α would correspond to a proportional higher value of EDL than DL.

The EDLS scheduling algorithm is specified as follows.

Algorithm 1 (EDLS)

Calculate Static Level and Δ for every task
while ∃ unscheduled task **do**
  Make list of Ready Tasks
  Calculate α for these tasks
  Calculate EDL value for Ready Tasks using Equation 5
  Schedule task-processor pair with the highest EDL
  Mark assigned task as scheduled
  Calculate DA and TF for next Ready Tasks
end while

Example: Let's reconsider the example in prior section. The task that is initially ready for execution is still Task 0. If Task 0 is to be executed by Processor 1, per Table I, the resulting energy consumption is 15.74 ms × 4.63 W = 72.87 mJ. Similarly, the energy consumed for Task 0 by Processors 2 and 3 are 154.32 mJ and 224.65 mJ, respectively. Hence, α for processor 1 is $\frac{72.87}{224.65} = 0.32$. Similarly, α for Processors 2, and 3 become 0.68 and 1.0, respectively. Consequently, the values of the second term in Equation 5 ($DL_{np} \times (1 - \alpha_{np})$) become 45.81, 0 and 5.38 for Processors 1, 2, and 3, respectively. Moreover, the EDL values for Task 0 and Processors 1, 2, and 3 become 70.81, 61.58 and 48.75, respectively. The results for Step 1 of the EDLS algorithm are given in Table VI. Accordingly, Task 0 has the highest EDL for processor 1, and therefore is assign to it.

Table VI
STEP 1 OF EDLS ALGORITHM

Task	SL	DA	TF	Δ	α	EDL
Processor 1						
0	46.90	0.0	0.0	-4.63	0.32	70.82 ⇐
Processor 2						
0	46.90	0.0	0.0	0.0	0.69	61.58
Processor 3						
0	46.90	0.0	0.0	1.85	1.0	48.75

During the second step, Tasks 1 and 3 are ready for execution. The resulting values for elements of Equations 5 is tabulated in Table VII. Accordingly, EDL_{11} is the maximum and therefore Task 1 is assigned to Processor 1.

Table VII
STEP 2 OF EDLS ALGORITHM

Task	SL	DA	TF	Δ	α	EDL
Processor 1						
1	35.79	15.74	15.74	-4.69	0.33	25.53 ⇐
3	22.86	15.74	15.74	-4.75	0.33	3.95
Processor 1						
1	35.79	17.71	0.0	0.0	0.71	23.28
3	22.86	17.68	0.0	0.0	0.70	6.73
Processor 2						
1	35.79	17.71	0.0	2.06	1.0	20.14
3	22.86	17.68	0.0	1.9	1.0	7.08

Similar tables are easily generated for the subsequent steps to determine processor-task pair combinations. Figure 4 shows the resulting scheduling diagram for our example. The total energy consumed is the sum of consumed energy

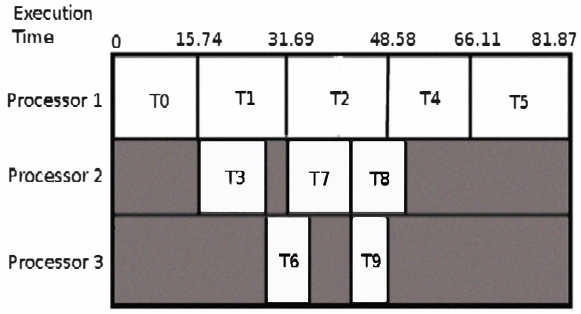


Figure 4. Scheduling using EDLS

of individual scheduled task-processor pairs by the EDLS algorithm, which would be 1.3 J. This amounts to about 34.51% energy saving compared to the DLS algorithm. The tradeoff comes in the increased execution time. In this case, the overall execution time is increased by about 29.67%, which is due to the assignment of tasks onto slower, but more energy efficient Processor 1.

So far, we have only considered $P_1@S_1$, $P_2@S_1$ and $P_3@S_1$. However, in our example $S_{P1} = \{S_1\}$ and $S_{P2} = \{S_1, S_2\}$ and $S_{P3} = \{S_1, S_2, S_3\}$. Hence, sixteen other processor-speed combinations exists. To get a general sense of energy versus execution delay characteristics, we repeatedly applied the EDLS algorithm to other processor-speed combinations. Table VIII compares the consumed energy for the DLS and EDLS algorithms for all seventeen processor-speed combinations. This includes cases where one processor is shut down (noted as speed 0 in the table). The table does not include the cases where only one processor is active, as this would lead to the same scheduling for both the DLS and the EDLS algorithms.

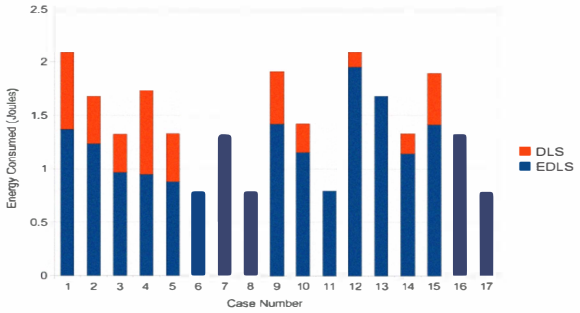
The right two columns of Table VIII indicate the resulting percent energy saving and percent slowdown in execution time, when scheduling Figure 1 tasks using the EDLS algorithm versus the DLS algorithm for each combination of processor and speed. For better illustration, the tabulated result is also depicted in Figure 5.

Clearly, in most cases, the extra energy saving is accompanied with added execution time. However, the amount of extra execution time varies and depends on the combination of processors and speed. For example, by simply switching $P_2@S_1$ to $P_2@S_2$, the energy saving is increased to about 45% (Case 4) for about the similar execution slow down as before (Case 1). On the otherhand, switching to $P_3@S_2$ (Case 2) can reduce the execution time penalty by half at the cost of modestly reducing the energy saving. Hence, the right combination of processors and speed is important in meeting the budgeted load and performance demands.

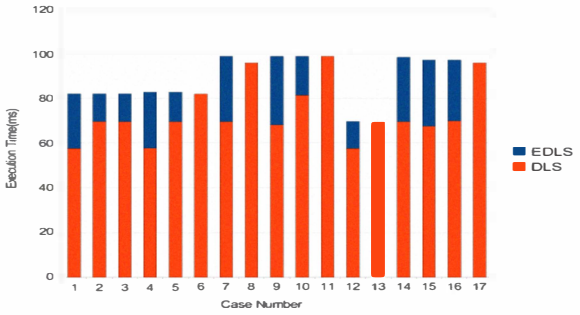
Our results indicate that higher heterogeneity of processors correlates with higher energy saving. For combinations which are slightly homogeneous, the EDLS algorithm still outperforms the DLS algorithm; the EDLS algorithm saves some energy and finishes the application slightly faster than the DLS algorithm. Case 6 is such an example, where all three processors are running at their most energy efficient mode. In this case, the EDLS algorithm attains nearly 1.5% energy saving while at the same time reducing the overall execution time by 3.1%. Hence, the EDLS algorithm is very effective and outperforms the DLS algorithm in both the homogenous and heterogeneous environments.

Table VIII
COMPARISON OF ENERGY CONSUMED BY DLS AND EDLS ALGORITHMS

Case Number	Proc1 Speed	Proc2 Speed	Proc3 Speed	DLS(J)	EDLS(J)	% Energy Saving	% Slowdown
1	1	1	1	2.09	1.37	34.53	29.67
2	1	1	2	1.68	1.23	26.46	15.05
3	1	1	3	1.32	0.96	27.16	15.1
4	1	2	1	1.74	0.95	45.5	30.24
5	1	2	2	1.33	0.88	33.96	15.85
6	1	2	3	0.79	0.78	1.57	-3.1
7	1	1	0	1.33	1.15	13.51	29.46
8	1	2	0	0.78	0.78	0	0
9	1	0	1	1.91	1.42	25.7	30.99
10	1	0	2	1.42	1.15	18.9	17.49
11	1	0	3	0.79	0.79	0	0
12	0	1	1	2.09	1.95	6.72	17.16
13	0	1	2	1.68	1.68	0	0
14	0	1	3	1.32	1.14	14.1	29.2
15	0	2	1	1.89	1.41	25.41	30.31
16	0	2	2	1.33	1.14	14.36	27.91
17	0	2	3	0.78	0.78	0	0



(a) Energy consumed



(b) Execution time

Figure 5. Comparison of DLS and EDLS

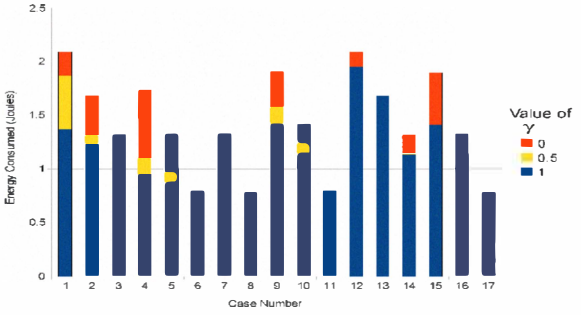
V. Measured EDLS

As the result of our last section demonstrated, often there is a tradeoff between saving energy and execution delay of tasks. Moreover, during certain computing periods, environmental condition or computing demand may change and adjustment may be required to accept less energy saving in favor of a faster execution delay or vice versa. To accommodate this, we introduce an operator controlled variable $0 \leq \gamma \leq 1$ and modify Equation 5,

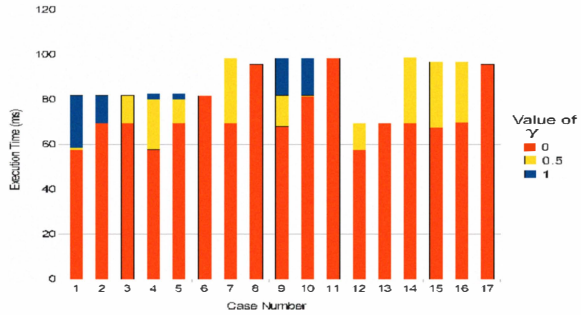
$$EDL_{np} = DL_{np} + \gamma \times (DL_{np} \times (1 - \alpha_{np})) \quad (7)$$

Subsequently, we call the resulting scheme the Measured EDLS, where the same EDLS Algorithm is used, except instead of Equation 5, Equation 7 is used to calculate EDL. Note that $\gamma = 0$ results in the DLS algorithm, and $\gamma = 1$ would implement the EDLS algorithm. Hence, for a higher value of γ, the algorithm favors more energy efficient processors at the likely expense of higher computation time and vice versa.

To compare the performance of the Measured EDLS algorithm under varied γ, we re-examined our prior example and compared the cases under γ = 0, 0.5, and 1. The result, as depicted in Figure 6, shows that although the effectiveness of γ varies from case to case, in about 65% of cases, the energy saving under $\gamma = .5$ is comparable to the EDLS ($\gamma = 1$). Moreover, in these cases, there is no additional execution time penalty beyond $\gamma = 0.5$.



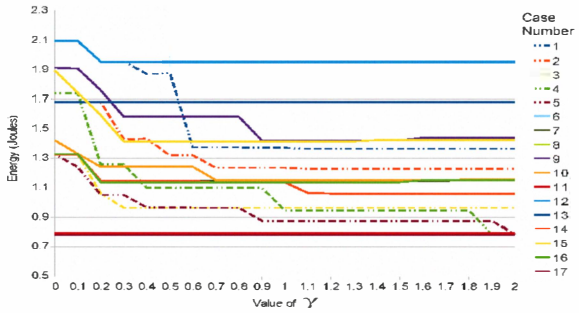
(a) Energy Consumed



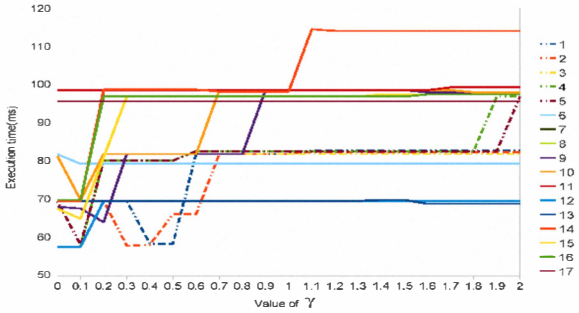
(b) Execution Time

Figure 6. Performance of EDLS under different γ

So far, our results demonstrate that both proper processors - speed combination and appropriate value for γ are needed to create an appropriate computing environment, which would match the budgeted energy and performance demands. For a better insight, we next examine the effectiveness of the Measured EDLS example case under a wide range of γ ($0 \leq \gamma \leq 2$). Note that cases with $\gamma > 1$ are added to examine the capacity of the Measured EDLS in attaining even higher energy saving. The results are depicted in Figure 7. Accordingly, our result demonstrate that the effectiveness of γ varies from case to case. However, based on Figure 7(a), about 40% of the cases, the energy saving attained by $\gamma = 0.4$ is comparable to the $\gamma = 1$. For $0.5 \leq \gamma \leq 1.1$, to a lesser degree, additional energy saving is achieved. Further changes in γ has little effect in the overall energy saving. As for the execution time, our result indicates that, in general, there is initially either no additional execution penalty or a reduction in the overall execution time. Moreover, in about 70% of the cases, there is no additional execution delay for $\gamma > 0.4$ cases. Finally, in all cases, there is no additional execution penalty for $\gamma \geq 1$ cases. Hence, the controlled operator γ, when used appropriately with processors - speed combination is beneficial in optimizing the energy usage while minimizing the execution penalty.



(a) Energy Consumed



(b) Execution Time

Figure 7. Performance of EDLS by varying γ

VI. Simulation Results

The result of our prior examples, while promising, represent scheduling a small task graph of Figure 1 on three processors. In this section, our JAVA-based simulator utilizes Task Graphs For Free (TGFF) [15] to examine the performance of the EDLS algorithm under large randomly generated task graphs with varying execution time and power

consumption. Our first case involves randomly generated 100 task DAG's that are to be scheduled onto a pool of five processors $P = \{P_1, P_2, P_3, P_4, P_5\}$, where $S_{P1} = \{S_1\}$, $S_{P2} = \{S_1, S_2\}$, $S_{P3} = \{S_1, S_2, S_3\}$, $S_{P4} = \{S_1, S_2, S_3\}$ and $S_{P5} = \{S_1, S_2, S_3\}$. The relative speed-power characteristics of P_1,P_2, and P_3 are kept as before (Figure 2), while P_4 and P_5 are designated as more high performance processors, as depicted in Figure 8. Therefore, we used our

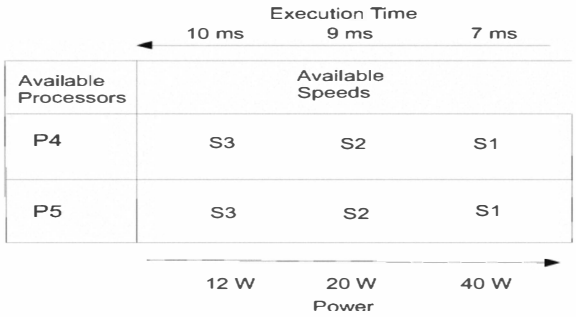
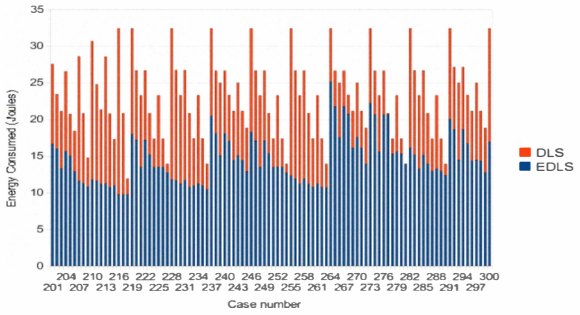


Figure 8. Processor Pool

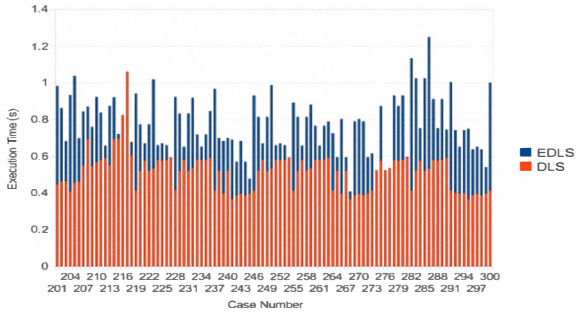
prior procedure to establish power and speed parameters for Processors 4 and 5 as well. Moreover, we have randomized the execution time and consumed power of each of the 100 tasks for a given speed of a processor using the base values from Figures 2 and 8 with $\pm 10\%$ variation. For example, the execution times for all the tasks on Processor 4 at speed 3 is randomized within the range of $10 \pm 1.1\ ms$ and the corresponding consumed power is randomized within the range of $12 \pm 1.2\ W$.

For the given processor pool, there are 371 possible speed-processor combinations, ranging from 2 to 5 processors and running at different speeds. For each case, our simulation algorithm picks a random DAG of 100 tasks and apply them to the DLS and the EDLS algorithms. To reduce the clutter, we have randomly selected the simulation results of a block of 100 (out of the 371 combinations), and displayed them in Figure 9. The result replicates our findings in our earlier example. Namely, different processors - speed combinations exhibit different power saving and execution penalty characteristic. However, the larger pool of processors and a larger number of tasks seem to result in a higher percentage of energy saving. In fact, in some cases, up to 70% energy saving is attained. Compared to the example case, the percent execution time overhead also seem to have increased, however, to a lesser degree. As before, large energy saving is predominate in cases where the speed difference between the processors is more prominent.

We next repeated our simulations by varying the value of $0 \leq \gamma \leq 2$. Five randomly selected cases were picked, as shown in Figure 10. The results demonstrate that γ is more effective with larger DAG's and additional processors. Moreover, the majority of the energy saving can be attained with $\gamma < .4$. Within this limit, the execution penalty will be less than 50%. Our results also reveal that $\gamma > 1$ has no

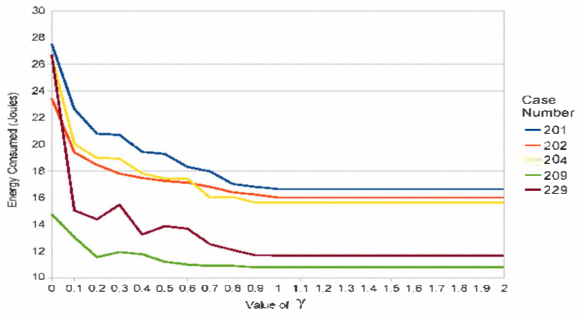


(a) Energy Consumed

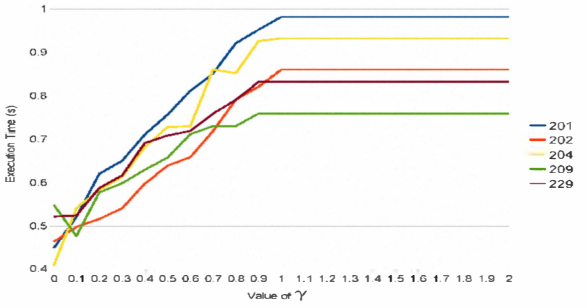


(b) Execution Time

Figure 9. Comparison of DLS and EDLS for 100 task DAG



(a) Energy Consumed

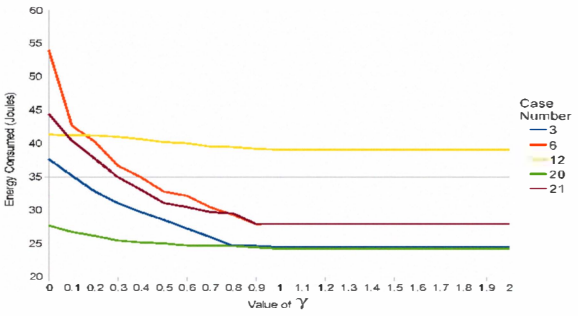


(b) Execution Time

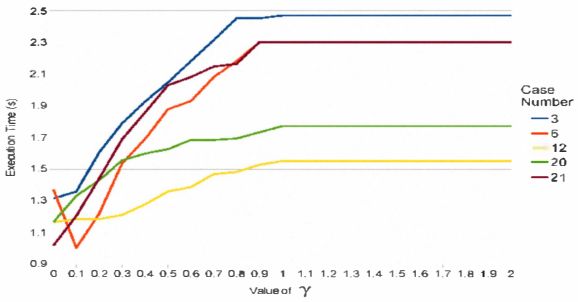
Figure 10. simulation results with variation in γ for 100 task DAG

effect on the performance of the EDLS algorithm.

To examine the scalability of the EDLS algorithm, we repeated our simulations with randomly selected 200 tasks onto the same pool of 5 processors. The 200 task DAG's were assigned similar properties as before. The result of five cases were randomly picked and shown in Figure 11.



(a) Energy Consumed



(b) Execution Time

Figure 11. Simulation results with variation in γ for 200 task DAG

The results show consistency with our prior simulations and indicate that the Modified EDLS is scalable as γ is effective in controlling execution overhead penalty while allowing to control the consumed energy.

VII. Conclusion

In this paper, we have have presented a scheme, called the Energy Dynamic Level Scheduling (EDLS). The scheme utilizes both time and energy to schedule tasks. The algorithm attains a higher energy saving by rewarding task processor pairs which are more energy efficient. Our results demonstrate that the EDLS algorithm can significantly improve the energy efficiency of a heterogeneous computing system. Moverover, with an appropriate processors - speed combination, the execution time penalty can be modest. In general, we have shown that a higher heterogeneity results in a higher energy saving, though, our EDLS algorithm outperforms the DLS algorithm even in a homogeneous computing environment. Our simulation results have revealed that the EDLS algorithm is scalable and therefore can be effective in data centers. To control the execution penalty that we may incur, our Modified EDLS scheme utilizes an operator controlled variable γ, which adjusts the scheduling cost function. Our results have shown that the scheme is especially useful with larger task graphs.

References

[1] P. M. Kogge, "The challenges of petascale architectures," *Computing in Science and Engineering*, vol. 11, pp. 10–16, Sep./Oct. 2009.

[2] M. Y. Lim, V. W. Freeh, and D. K. Lowenthal, "Adaptive, transparent frequency and voltage scaling of communication phases in mpi programs," *Proc. of the 2006 ACM/IEEE Conf. on Supercomputing*, 2006.

[3] A. Chandrakasan, S. Sheng, and W. Brodersen, "Low-power cmos digital design," *IEEE Journal of Solid-State Circuits*, pp. 473–484, April 1992.

[4] H. Topcuouglu, S. Hariri, and M. you Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, p. 260, 2002.

[5] D. Zhu, R. Melhem, and B. Childers, "Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems," *Proc. of the 22nd IEEE Real-Time Syst. Symp. (RTSS)*, p. 84, 2001.

[6] B. Rountree, D. K. Lowenthal, S. Funk, V. W. Freeh, and M. Schulz, "Bounding energy consumption in large-scale mpi programs," *Proc. of the 2007 ACM/IEEE conf. on Supercomputing*, p. 1, Nov. 2007.

[7] S. Ranka and J. Kang, "Dynamic algorithms for energy minimization on parallel machines," *16th Euromicro Conference on Parallel, Distributed and Network-Based Processing*, p. 399, Feb. 2008.

[8] K. H. Kim, R. Buyya, and J. Kim, "Power aware scheduling of bag-of-tasks applications with deadline constraints on dvs-enabled clusters," *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*, pp. 541–548, 2007.

[9] C.M.Hung, J. Chen, and T.W.Kuo, "Energy-efficient real time task scheduling for a dvs system with non-dvs processing element," *ACM/IEEE Conference of Design, AUtomation and Test in Europe*, 2006.

[10] J. Luo and N. Jha, "Static and dynamic variable voltage scaling algorithms for real time heterogeneous distributed embedded systems," *15th International Conference on VLSI Design*, pp. 719–726, 2002.

[11] G. Sih and E. Lee, "A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures," *IEEE Transactions on Parallel and Distributed Systems*, pp. 175–187, 1993.

[12] M. Iverson and F. Ozguner, "Dynamic, competitive scheduling of multiple dags in a distributed heterogeneous environment," *Seventh Heterogeneous Computing Workshop*, p. 70, Mar. 1998.

[13] A. Dogan and F. Ozguner, "Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, March 2002.

[14] List of cpu power dissipation. [Online]. Available: http://en.wikipedia.org/wiki/ListofCPUpowerdissipation

[15] Task graphs for free. [Online]. Available: http://ziyang.eecs.umich.edu/ dickrp/tgff/