Temperature and Energy Aware Scheduling of Heterogeneous Processors

Rashadul Kabir
Dept. of Electrical and Computer Engineering
Presidency University
11/A, Road 92, Gulshan, Dhaka 1212, Bangladesh
Email: rashadk@mail.presidency.edu.bd

Baback Izadi
Division of Engineering Programs
State University of New York at New Paltz
New Paltz, NY, 12561, USA
Email: bai@engr.newpaltz.edu

***Abstract*—Modern computing requires faster and more powerful processing. Faster and more powerful processors have resulted in higher heat dissipation and power consumption. In this paper we present an offline algorithm called Temperature and Energy aware Dynamic Level Scheduling (TEDLS). It is able to schedule tasks in a heterogeneous environment with DVS enabled processors to minimize execution time, energy consumption and heat dissipation. We use a heat model to estimate the final temperature of a processor executing a task. This estimation of temperature is based on processor characteristics, which aids in choosing the cooler processors. Our simulation results have shown that the TEDLS algorithm not only results in processors having lower temperatures but also produces lower energy consumption as compared to the previous offline algorithms. The TEDLS algorithm also produces lower application execution time when the application size is small.**

Keywords-Heterogeneous environment; DVS; DAG; Scheduling; Temperature and Energy aware; TEDLS

I. Introduction

With widespread growth of the internet, data centers have been trying to incorporate faster and more powerful processors to meet the high data processing requirements. This has resulted in data centers consuming several MW's of power every year. Studies have shown that efforts in improving energy or power efficiency has become the biggest hurdle in developing larger data centers [1]. Processors consume third to half of system's total power [2]. Thus, by lowering processor power consumption, power required by data centers can be significantly reduced. The total power consumed by a processor is the summation of the static and dynamic power. Static power is technology dependent and dynamic power [3] can be given by (1),

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

Where, α is the switching activity, C_{ef} is the effective capacitance, V_{dd} is the supply voltage and f is the clock frequency. Per (1), a decrease in supply voltage would bring about a quadratic decrease in dynamic power. Thus, it seems changing the supply voltage can be the most effective way of reducing power consumption in processors. The mechanism that allows us to achieve this is called Dynamic Voltage Scaling (DVS), and processors with such capability are known as DVS enabled processors. Two examples of DVS enabled processors are Enhanced Intel SpeedStep [4] and AMD's Cool 'n' Quiet [5].

Aside from high power usage, processor temperatures should not be allowed to go beyond a critical temperature of operation, as such a situation might cause faults in the circuit. For example, rise in junction (device) temperature causes high leakage current [6], which can ultimately damage the processor circuitry. Spatial increase in temperature can cause Elmore delay when there is inter-processor communication [7]. Thus, when scheduling tasks onto processors, both energy consumption and processor temperature have to be taken into consideration alongside application execution time. Task scheduling to minimize execution time and energy consumption on DVS enabled processors has already been shown to be NP-Complete [8]. Thus, task scheduling to minimize execution time, energy consumed and processor temperature would also certainly be considered NP-Complete. An optimal heuristic scheduling algorithm called the Dynamic Level Scheduling (DLS) algorithm was proposed by Sih and Lee [9], which schedules tasks based on fastest execution of an application. Other researchers [10], [11], [12], [13], [14] have developed energy-efficient scheduling algorithms for heterogeneous processors. Shekar and Izadi [14] developed an algorithm called the Energy Dynamic Level Scheduling (EDLS) algorithm, that focuses on minimizing both application execution time and energy consumed. Although the EDLS algorithm focuses on minimization of both execution time and energy consumption, it does so at times by heating up the most energy efficient processor. This might give rise to transient or permanent faults, making the system less reliable. Several researchers [7], [6], [15], [16] have used thermal sensors to obtain processor temperatures, and use the readings to slow down or shut down overheated processors. One of the drawbacks in their approach is the lack of emphasis on energy usage.

Our algorithm, called the Temperature and Energy aware Dynamic Level Scheduling (TEDLS) algorithm, is an ex-

978-1-5090-3251-8/16/$31.00 ©2016 IEEE

tension of the DLS algorithm [9]. It focuses on minimizing energy consumption, application execution time and processor temperature. We use the heat model in [17] to predict final temperatures of processors while running a given task. Our algorithm schedules tasks to the most energy efficient processor as long as it is also one of the cooler processors.

The rest of the paper is organized as follows. In Section II, we discuss some definitions and notations. In Section III, we explain our scheduling algorithm, called the TEDLS algorithm. In Section IV, we illustrate scheduling of tasks onto a pool of processors using the TEDLS algorithm and compare simulation results for execution time, energy consumption and processor temperatures with those while scheduling tasks using the DLS and EDLS algorithms. Finally, concluding remarks are given in Section V.

II. Definitions and Background

For simulation purposes, we assume that our applications are periodic in nature. We also assume that applications can be represented in a DAG structured form. Fig. 1 shows a typical DAG application $G = (T, E)$, where each node represents a task $T_i \in T$. An arrow from one task to another task indicates the dependency of tasks. If two tasks are performed in separate processors, each weighted directed edge $E_{ij} = (T_i, T_j) \in E$ represents the communication delay associated with sending of task T_i to the processor executing task T_j. For example, T_0, in Fig. 1, needs to be executed before T_1. If T_0 and T_1 are executed on two processors P_i and P_j, it takes 1.99 time units to transfer the results of T_0 from P_i to P_j.

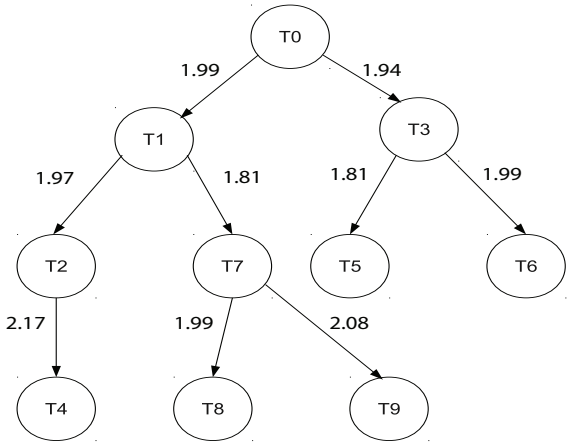


Fig. 1. Example DAG

Fig. 2 shows an example of a pool of three processors $P = \{P_1, P_2, P_3\}$, that can be used to schedule the tasks in Fig. 1. Here, the processor speeds are $S_{P1} = \{S_1\}$ and $S_{P2} = \{S_1, S_2\}$ and $S_{P3} = \{S_1, S_2, S_3\}$. Hence, P_1 can only operate at a single frequency, and P_2 and P_3 have the DVS capability of operating at two and three different speeds, respectively. Processor speed has a linear correlation with supply voltage $f \propto (V_{dd} - V_t)^2/V_{dd}$ [18]. Thus, per (1), power consumed can be found to be cubically correlated with the speed of the processor. Equation (2) shows the relationship between power consumed and its speed of operation.

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

We have intentionally chosen power and speed models for our different processors to be simple. Emphasis has been given to the relationship between execution time and power values, rather than exact values of both. Consistent with the current technology [19], the processors have been chosen to have a maximum of three power settings. It should be noted that, $P_3@S_3$, $P_2@S_2$, and $P_1@S_1$ have 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 the fastest processor-speed combination and the least power efficient processor.

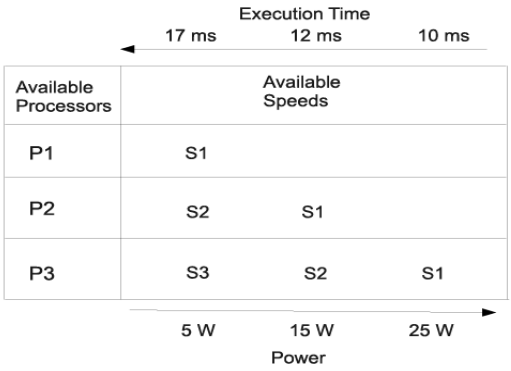


Fig. 2. Processor Pool

When different processors execute different tasks, several scenarios can occur. A processor that is assigned to perform Task 1 will have to wait for Task 0 to end, so that its results can be used to execute Task 1. The time required for data to be available to a processor is called Data Ready Time (DA_{np}). Here, n represents the task number and p the processor number. Sometimes, certain processors might be busy executing tasks and therefore unable to execute ready tasks. If P_1 is already executing T_0, then the Processor Ready Time (TF_{np}) to perform T_1 would be the execution time for T_0. Static Level, denoted by SL_{np}, is the summation of the execution times of tasks along the longest path. Processor Speed Difference (Δ_{np}) is the difference in execution time of T_n on Processor p with that of the Median Processor.

III. TEDLS Algorithm

In this section, we present our energy-efficient scheduling algorithm, Temperature and Energy aware Dynamic Level Scheduling (TEDLS), that also focuses on minimal processor temperature. Given that there is a pool of heterogeneous processors to perform an application, this offline

algorithm can schedule application tasks to the cooler and more energy efficient processors with the help of the cost function given by (3). This cost function is a modification of the cost function for the DLS algorithm [9], as shown in (4).

$$TEDL_{np} = DL_{np} + DL_{np} \times (1 - NormTemp) \quad (3)$$

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

$NormTemp$ in (3) is used to penalize the heated processors, and is given by (5). Its value is obtained by dividing the predicted final temperature T_{np} by a maximum operating temperature $MaxTemp$ (set by the manufacturer). Cooler processors will have lower values of $NormTemp$, while processors that have heated up from performing tasks will have a higher value of $NormTemp$.

$$NormTemp = \frac{T_{np}(t_2)}{MaxTemp} \quad (5)$$

We estimate the value of final temperature, $T_{np}(t_2)$ of Processor p after execution of task n, using the following heat model [17] in (6). This heat model is particularly useful because it not only reflects the increase in processor temperature as the processor is working, but it also reflects the cooling effect on the processor due to Newton's law of cooling.

$$T_{np}(t_2) = \frac{\beta P_{np}}{\rho} + (T_{np}(t_1) - \frac{\beta P_{np}}{\rho}) \exp^{-\rho(t_2 - t_1)} \quad (6)$$

Algorithm 1 (TEDLS)

```
Calculate Static Level and Δ for every task
while ∃ unscheduled task do
    Make list of Ready Tasks
    Estimate T_np(t_2) for Ready Tasks on Processor p using (6)
    Calculate NormTemp using (5)
    Calculate TEDL for Ready Tasks using (3)
    if T_np(t_2) of processor-task pair with highest value of TEDL != Highest T_np(t_2) then
        Schedule task-processor pair with the highest TEDL
    else
        Schedule available task-processor pair with the next highest TEDL
    end if
    Mark assigned task as scheduled
    Calculate DA and TF for next Ready Tasks
end while
```

In this heat model, β represents the thermal resistance of the processor; i.e. the amount of heat needed to be supplied to raise the processor temperature by 1 K. P_{np} represents power dissipation of Processor p while executing Task n, ρ represents the cooling constant of the processor, $T_{np}(t_1)$ represents the initial temperature of the processor and $(t_2 - t_1)$ is the execution time of Task n on Processor p. Our TEDLS scheduling algorithm is given by Algorithm 1.

IV. Simulation and Comparative Analysis

Let's consider the example DAG in Fig. 1 and see how tasks in it can be scheduled on to a pool of processors P in Fig. 2 using the TEDLS algorithm. For the sake of simulation, we randomized the execution times and consumed power for each of the 10 tasks in Fig. 1 for Processors 1, 2 and 3 within $\pm 10\%$ of the nominal values in Fig. 2. These values are provided in Tables I, II and III. In this example, we first consider the case where processors are set to be running at their maximum speeds, i.e. $P_1@S_1$, $P_2@S_1$ and $P_3@S_1$.

Table I
Execution of tasks in Example DAG on Processor 1

Task Number	Exec Time $@S_1$(ms)	Power $@S_1$(W)
0	15.74	4.63
1	15.95	4.69
2	16.89	4.97
3	16.2	4.76
4	17.53	5.16
5	15.76	4.64
6	16.29	4.79
7	15.71	4.62
8	16.68	4.91
9	17.4	5.12

Table II
Execution of tasks in Example DAG on Processor 2

Task Number	Exec Time $@S_1$(ms)	Power $@S_1$(W)	Exec Time $@S_2$(ms)	Power $@S_2$(W)
0	11.11	13.89	15.60	4.59
1	11.25	14.07	15.63	4.60
2	12.11	15.13	17.07	5.02
3	11.45	14.31	16.02	4.71
4	12.44	15.55	17.57	5.17
5	11.03	13.79	15.91	4.68
6	11.41	14.26	16.20	4.77
7	11.22	14.02	15.81	4.65
8	11.58	14.48	16.75	4.93
9	12.2	15.24	17.20	5.06

Table III
Execution of tasks in Example DAG on 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	23.15	11.01	13.76	15.79	4.64
1	9.2	22.99	11.01	13.76	15.9	4.68
2	10.05	25.13	12.12	15.15	16.97	4.99
3	9.54	23.85	11.41	14.27	15.97	4.7
4	10.33	25.83	12.4	15.5	17.5	5.15
5	9.36	23.41	11.22	14.03	15.81	4.65
6	9.39	23.48	11.51	14.39	16.08	4.73
7	9.2	23.0	11.1	13.88	15.81	4.65
8	9.78	24.45	11.73	14.66	16.75	4.93
9	10.2	25.50	12.39	15.49	17.53	5.16

Per Algorithm 1, we first need to calculate the static level (SL) and the processor speed difference (Δ) of all tasks. SL gives priority to longest task path in

the DAG, while Δ indicates the relative speed of a processor to the median processor. Here, $P_2@S_1$ has the median speed. In Fig. 1, Task 0 has 5 end tasks. However, the longest path to an end task can only be out of the paths containing four tasks. Hence, to figure out the SL for Task 0 we need to find out the sum of median execution times of the following three paths.

$T_0 \rightarrow T_1 \rightarrow T_2 \rightarrow T_4 \Rightarrow 11.11 + 11.25 + 12.11 + 12.44 = 46.91ms \Leftarrow$

$T_0 \rightarrow T_1 \rightarrow T_7 \rightarrow T_8 \Rightarrow 11.11 + 11.25 + 11.22 + 11.58 = 45.16ms$

$T_0 \rightarrow T_1 \rightarrow T_7 \rightarrow T_9 \Rightarrow 11.11 + 11.25 + 11.22 + 12.20 = 45.78ms$

Thus, the SL for Task 0 is calculated to be $46.91ms$. SL of all the tasks in Figure 1 are shown in Table IV.

Table IV
STATIC LEVEL TABLE

Task Number	Median Times	Static Levels
0	11.11	46.91
1	11.25	35.80
2	12.11	24.54
3	11.45	22.86
4	12.44	12.44
5	11.03	11.03
6	11.41	11.41
7	11.22	23.41
8	11.58	11.58
9	12.2	12.2

Initially, per Fig. 1, the task ready for execution is Task 0. After Task 0 is scheduled, the following tasks that are tagged ready have to be scheduled until there is no more task left. The predicted final temperatures of different processors executing Task 0 can be calculated using the heat model provided in (6) and the power consumption and execution time values given in Tables I, II and III. Table V shows the final temperature values for Task 0. Here, we use a typical value of β (thermal resistance) = 1 J/K and of ρ (cooling constant) = 0.003 K/J per [20]. Also, we use an arbitrary value of $T_{0p}(0)$ (initial temperature) = 313.15 K (40 °C).

Table V
STEP 1A OF THE TEDLS ALGORITHM

Task	β	P_{np}	ρ	$T_{np}(t_1)$	$(t_2 - t_1)$	$T_{np}(t_2)$
Processor 1						
0	1	4.63	0.003	313.15	15.74	369.91
Processor 2						
0	1	13.89	0.003	313.15	11.11	454.71
Processor 3						
0	1	23.15	0.003	313.15	9.26	516.02

Once final temperatures are determined, the $NormTemp$ values can be calculated using (5). These values are shown in Table VI. In accordance to [7], we chose the $MaxTemp$ to be 358.15 K (85 °C).

The next step requires calculation of $TEDL_{np}$ using (3). The results are shown in Table VII. Since Task 0 does not depend on the result of any previous task, DA of Task 0 for all processors is 0 ms. Also, since all processors are at their idle states, before the execution of Task 0, TF for all processors is also 0 ms.

Table VI
STEP 1B OF THE TEDLS ALGORITHM

Task	$T_{np}(t_2)$	$MaxTemp$	$NormTemp$
Processor 1			
0	369.91	358.15	1.03
Processor 2			
0	454.71	358.15	1.27
Processor 3			
0	516.02	358.15	1.44

Table VII
STEP 1C OF THE TEDLS ALGORITHM

Task	SL	DA	TF	Δ	DL	$NormTemp$	TEDL
Processor 1							
0	46.91	0.0	0.0	-4.63	42.28	1.03	40.89 ⇐
Processor 2							
0	46.91	0.0	0.0	0.0	46.91	1.27	34.26
Processor 3							
0	46.91	0.0	0.0	1.85	48.76	1.44	27.27

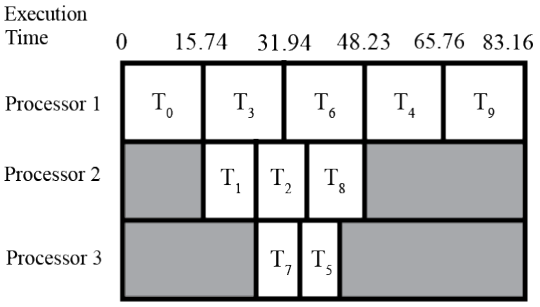
Table VII indicates that Task 0 has the highest value of $TEDL$ for Processor 1. High $TEDL_{np}$ generally indicates a low expected final temperature. However, a large Δ may cause a processor with a high $T_{np}(t_2)$ to have a high $TEDL_{np}$. Thus, according to Algorithm 1, we need to check if $T_{01}(t_2)$ is the highest $T_{np}(t_2)$. Per Table V, $T_{01}(t_2)$ is 369.91 K, which also happens to be the lowest $T_{np}(t_2)$. Therefore Task 0 is scheduled to Processor 1. Per (6), expected final temperature value ($T_{np}(t_2)$) is proportional to the power consumed (P_{np}). Hence, a processor with the lowest $T_{np}(t_2)$ will likely be the most energy efficient processor as well.

Per Fig. 1, the next ready tasks are Task 1 and Task 3. At Step 2, the above process is repeated for Task 1 and Task 3. Table VIII shows the calculation for TEDL values of both Tasks 1 and 3 on different processors. DA for both Processor 2 and 3 is the summation of the execution time of Task 0 on Processor 1 and the communication delay for the results of Task 0 to travel to either Processor 2 or 3, i.e. DA_{12} = DA_{32} = 15.74 ms + 1.99 ms = 17.73 ms. Since both of the processor are idle, TF for both Processor 2 and 3 is zero. DA and TF for Processor 1 are both 15.74 ms, since both previous and present task would reside in the same processor.

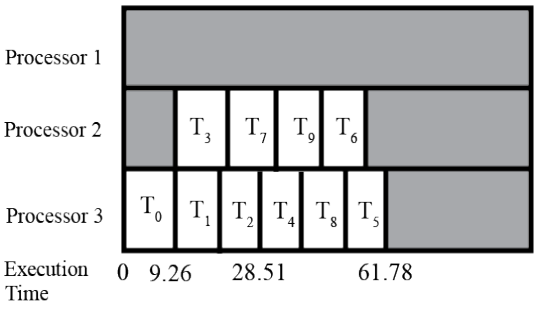
Table VIII
STEP 2 OF THE TEDLS ALGORITHM

Task	SL	DA	TF	Δ	T_{np}	$NormTemp$	TEDL
Processor 1							
1	35.80	15.74	15.74	-4.70	426.20	1.19	12.46
3	22.86	15.74	15.74	-4.75	416.20	1.19	1.92
Processor 2							
1	35.80	17.73	0.0	0.0	458.43	1.28	13.01 ⇐
3	22.86	17.68	0.0	0.0	462.01	1.29	3.68
Processor 3							
1	35.80	17.73	0.0	2.06	512.15	1.43	11.47
3	22.86	17.68	0.0	1.91	530.06	1.48	3.69 ⇐

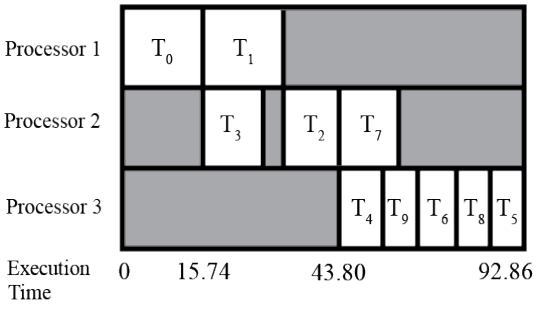
Per Table VIII, the TEDL value for Task 1 is the highest for Processor 2. Since the final temperature of Processor 2 executing Task 1 is also one of the lower values, Task 1 can be scheduled to Processor 2. The TEDL value for Task 3 is the highest for Processor 3. However, the final temperature of Processor 3 performing Task 3 is 530.06 K, which is the highest expected final temperature value for Task 3. Thus the available Processor with the next highest TEDL value should be assigned to perform the task. Since Processor 2 has already been assigned to Task 1, Task 3 has to be scheduled to Processor 1. Similar tables can be easily generated for the subsequent ready tasks. Fig. 3(a) shows the resulting scheduling diagram for the example DAG.



(a) TEDLS algorithm



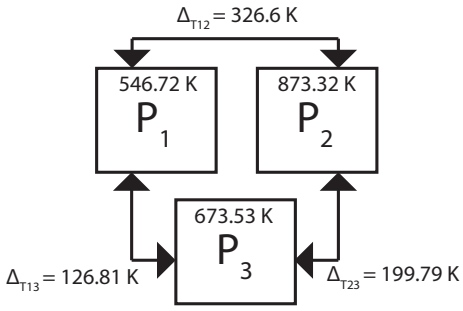
(b) DLS algorithm



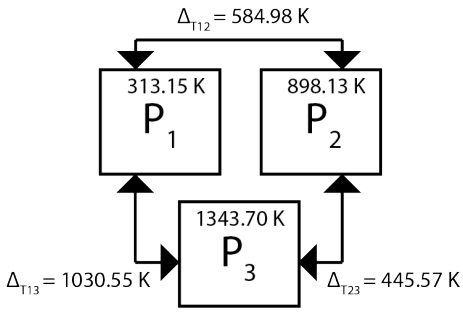
(c) EDLS algorithm

Fig. 3. Scheduling of Processors under the TEDLS, DLS and EDLS algorithms

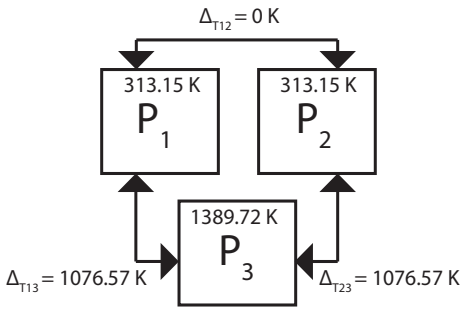
For comparative purposes, we applied both the DLS and EDLS algorithms to the DAG of Fig. 1. Fig. 3(b) and Fig. 3(c) show the resulting scheduling of tasks under the two algorithms, respectively. The total energy consumed under an algorithm, $E_n = \sum Power_{np} \times ExecTime_{np}$, where $Power_{np}$ and $ExecTime_{np}$ are the power consumption and execution time for for Task n on Processor p, respectively. Therefore, energy consumption of processors under the DLS, EDLS and TEDLS algorithms, calculated using Tables I, II and III, are 2.32 J, 1.86 J and 1.26 J, respectively. Thus, scheduling using the TEDLS algorithm has resulted in 45.69% and 32.08% energy savings in processors over the DLS and EDLS algorithms, respectively. The execution time of application when processors are scheduled with the DLS, EDLS and TEDLS algorithms are 61.78 ms, 92.86 ms 83.16 ms. The results show that despite the significant energy saving, the TEDLS algorithm also has improved execution time over the EDLS algorithm.



(a) TEDLS algorithm



(b) DLS algorithm



(c) EDLS algorithm

Fig. 4. Processor temperatures under the TEDLS, DLS and EDLS algorithms

We next used the heat model of (6) to predict the final processor temperatures. Fig. 4 shows the predicted final temperatures for processors scheduled with the TEDLS, DLS and EDLS algorithms. The goal of this prediction is not to find the exact value of processor temperatures, but

rather to compare how processor temperatures vary when they are scheduled with different scheduling algorithms.
Fig. 4(a) indicates that the final temperature difference between Processors 1 and 2, δ_{12} = 326.6 K, is the highest δ. In contrast, Fig. 4(b) and Fig. 4(c) illustrate the greatest δ of 1030.55 K and 1076.57 K for the DLS and EDLS algorithms, respectively. Thus, the TEDLS algorithm resulted in a more uniform temperature distribution among the processors, which makes them less susceptible to transient and permanent hardware faults, as well as lower chances of Elmore Delay [7] during inter processor communication; Elmore delay can cause a processor circuit to malfunction and even stall applications.

So far, in this example, only processor speeds $P_1@S_1$, $P_2@S_1$ and $P_3@S_1$ have been considered. However, the pool of processors in Fig. 2 can have sixteen other processor-speed combinations. Table IX shows the energy saving and speedups of the TEDLS algorithm with respect to the DLS and EDLS algorithms for all speed combinations for the example DAG in Fig. 1. In Table IX, Processor Speed 0 indicates that the processor is not being utilized. The highest Processor Speed is denoted by 1, which according to Processor Pool in Fig. 2 is around 17 ms for P_1, 12 ms for P_2 and 10 ms for P_3. Processor Speed 2 and 3 denote the lower processor speeds. It can be seen in Table IX that almost consistently, the energy saving under the TEDLS algorithm is greater than those for the DLS and EDLS algorithms. Energy saving can be as high as 59.82 % and 41.03 % with respect to the DLS and EDLS algorithms, respectively. Energy saving under the TEDLS algorithm is greater than that under the DLS algorithm, because the TEDLS algorithm focuses on the power efficient and cooler processors. It has been observed that unlike the EDLS algorithm, the TEDLS algorithm does not have the tendency to schedule inter-dependent tasks on the same processor. So high power consuming tasks are scheduled to cooler and more energy efficient processors. Also, a faster processor is able to execute a low energy consuming task. Thus, for the example DAG in Fig. 1, the difference in processor speed compensates for the inter-processor communication, causing processors assigned tasks using the TEDLS algorithm to execute the application faster than the DLS or EDLS algorithm.

Using the heat model in (6), we determined the final temperatures of the processors under the DLS, EDLS and TEDLS algorithms. The results are shown in Table X. Emphasis is given on the comparative analysis of temperature of processors under different scheduling algorithms, and not on the exact estimation of the temperatures. Also, only cases where all the processors were utilized are shown. Table X illustrates that P_1 exhibits higher temperature when tasks are scheduled onto it using the TEDLS algorithm. On the other hand, P_2 and P_3 exhibit higher processor temperatures under the DLS and EDLS algorithms. This is because the TEDLS algorithm schedules tasks onto the more energy efficient and slower processors. P_1 being the most energy efficient, gets scheduled with more tasks when scheduled with the TEDLS algorithm. This causes temperature of P_1 to increase. P_2 and P_3, being less energy efficient, get scheduled with fewer number of tasks, and thus their temperatures would not be affected much.

Table IX
COMPARISON OF ENERGY CONSUMPTION AND EXECUTION TIMES FOR DLS AND EDLS ALGORITHMS WITH THOSE FOR TEDLS ALGORITHM

Case Number	P_1 Speed	P_2 Speed	P_3 Speed	% Energy saving wrt DLS	% Energy saving wrt EDLS	% Speedup wrt DLS	% Speedup wrt EDLS
1	1	1	0	37.81	16.01	0.73	1.04
2	1	2	0	-0.26	-0.26	13.44	13.45
3	1	0	1	53.07	36.83	-33.30	-25.93
4	1	0	2	38.42	17.79	-1.48	-0.66
5	1	0	3	-0.39	-0.39	8.26	8.26
6	0	1	1	16.76	4.39	16.62	11.31
7	0	1	2	-1.27	-1.27	16.05	16.05
8	0	1	3	42.77	22.68	-11.31	2.54
9	0	2	1	53.21	31.36	-32.99	-16.69
10	0	2	2	43.06	23.79	-16.20	-15.46
11	0	2	3	-0.24	-0.24	16.31	16.31
12	1	1	1	45.69	32.08	-0.18	7.32
13	1	1	2	27.58	14.47	1.92	1.39
14	1	1	3	41.80	21.27	34.44	42.51
15	1	2	1	59.82	41.03	3.05	11.79
16	1	2	2	43.16	23.87	35.22	34.73
17	1	2	3	-0.71	-0.71	14.30	14.30

Table X
COMPARISON OF TEMPERATURE REDUCTION FOR TEDLS ALGORITHM AS COMPARED WITH DLS AND EDLS ALGORITHMS

Case Number	P_1 Temperature (K) under DLS	P_1 Temperature (K) under EDLS	P_1 Temperature (K) under TEDLS	% Reduction in Temperature wrt DLS for P1	% Reduction in Temperature wrt EDLS for P1
12	313.15	313.15	659.66	-110.65	-110.65
13	313.15	414.55	603.78	-92.81	-45.65
14	313.15	471.47	550.87	-75.91	-16.84
15	313.15	313.15	554.07	-76.93	-83.05
16	313.15	313.15	554.07	-76.93	-76.93
17	561.80	561.80	496.02	11.71	11.71
Case Number	P_2 Temperature (K) under DLS	P_2 Temperature (K) under EDLS	P_2 Temperature (K) under TEDLS	% Reduction in Temperature wrt DLS for P2	% Reduction in Temperature wrt EDLS for P2
12	313.15	1230.05	629.51	-101.02	48.82
13	1607.41	1592.50	623.62	61.20	60.84
14	3113.34	2122.24	649.57	79.14	69.39
15	313.15	501.60	594.33	-89.79	-18.49
16	313.15	501.60	547.29	-74.77	-9.11
17	523.08	523.08	497.97	4.80	4.80
Case Number	P_3 Temperature (K) under DLS	P_3 Temperature (K) under EDLS	P_3 Temperature (K) under TEDLS	% Reduction in Temperature wrt DLS for P3	% Reduction in Temperature wrt EDLS for P3
12	4621.34	3038.37	753.94	83.69	75.19
13	1915.68	1330.32	765.70	60.03	42.44
14	313.15	345.06	555.05	-77.25	-60.86
15	4608.55	3023.74	559.80	87.85	81.49
16	3141.51	2160.06	626.33	80.06	71.00
17	699.84	699.84	555.04	20.69	20.69

To get a better sense of our algorithm's capability, we implemented the DLS, EDLS and TEDLS algorithms on larger DAG's, which have been randomly generated using TGFF [21]. Fig. 5 shows the variation in energy consumption for 10 Task, 20 Task, 30 Task and 40 Task DAGs using the DLS, the EDLS and the TEDLS algorithm. It can be observed in Fig. 5 that as the DAG size starts to increase, energy saving under the TEDLS algorithm with respect to the DLS or EDLS algorithm also increases. Thus, in a practical scenario where DAG sizes are large, the TEDLS algorithm will give rise to a substantial energy saving. Fig. 6 shows variation of execution times of applications when scheduled using different scheduling algorithms. It is evident from Fig. 6 that as the DAG size increases, the execution time under the TEDLS algorithm is negatively impacted.

This occurs as the TEDLS algorithm, in trying to manage and provide a uniform temperature among processors, shifts some of the tasks from faster to slower and more energy efficient processors.

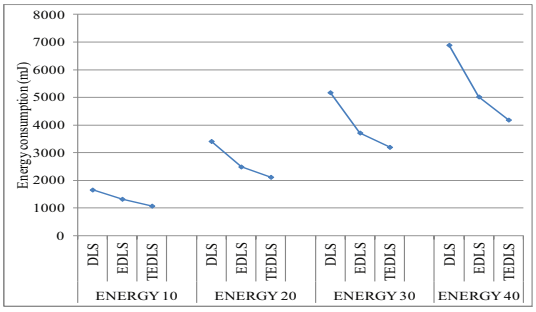


Fig. 5. Variation of Energy consumption for 10 Task, 20 Task, 30 Task and 40 Task DAGs

Fig. 6. Variation of Execution time for 10 Task, 20 Task, 30 Task and 40 Task DAGs

V. Conclusion

In this paper we presented an offline algorithm that not only helps in achieving low and uniform processor temperature, but also produces lower energy consumption than the energy efficient EDLS algorithm. Also, for smaller DAG sizes, the TEDLS algorithm is shown to be competitive with the DLS algorithm that focuses on fastest execution of tasks. However, as the DAG size increases in a processor constrained environment, the TEDLS algorithm schedules more tasks to slower and energy efficient processors to manage processor temperature and energy consumption.

References

[1] P. Kogge and et al., "Exascale computing study: Technology challenges in achieving exascale systems," *DARPA Information Processing Techniques Office (IPTO) sponsored study, http://www.cse.nd.edu/Reports/2008/TR-2008-13.pdf*, 2008.

[2] V. W. F. M. Y. Lim and D. K. Lowenthal, "Adaptive, transparent frequency and voltage scaling of communication phases in MPI programs," *Proceedings of the 2006 ACM/IEEE Conf. on Supercomputing 2006*, p. 14, November 2006.

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

[4] Enhanced Intel SpeedStep Technology. http://www.intel.com/.

[5] Cool'n'quiet technology. http://www.amd.com/.

[6] M. S. A. K. O. Semenov, A. Vassighi and C. Hawkins, "Burn-in temperature projections for deep sub-micron technologies," *Proceedings of the International Test Conference 2003*, pp. 95–104, September 2003.

[7] T. S. R. A. K. Coskun and K. Whisnant, "Temperature aware task scheduling in MPSoCs," *Design, Automation Test in Europe Conference Exhibition*, pp. 1–6, April 2007.

[8] M. Y. W. H. Topcuoglu, S. Hariri, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transaction Parallel Distributed Systems*, p. 260, March 2002.

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

[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] C. M. Hung, J. 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*, pp. 303–312, December 2006.

[12] Z. Zong, X. Qin, X. Ruan, K. Bellam, M. Nijim, and M. Alghamdi, "Energy-efficient scheduling for parallel applications running on heterogeneous clusters," *2007 International Conference on Parallel Processing (ICPP 2007)*, September 2007.

[13] R. Xu, R. Melhem, and D. Moss, "Energy-aware scheduling for streaming applications on chip multiprocessors," *28th IEEE International Real-Time Systems Symposium*, December 2007.

[14] V. Shekar and B. Izadi, "Energy aware scheduling for dag structured applications on heterogeneous and DVS enabled processors," *Proceedings International Conference on Green Computing*, pp. 495–502, August 2010.

[15] S. Sharifi and T. S. Rosing, "Accurate direct and indirect on-chip temperature sensing for efficient dynamic thermal management," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, pp. 1586–1599, October 2010.

[16] R. T. S. Lu and W. Burleson, "Collaborative calibration of on-chip thermal sensors using performance counters," *IEEE International Conference on Computer-Aided Design (ICCAD)*, pp. 15–22, November 2012.

[17] D. Rajan and P. Yu, "Temperature-aware scheduling: When is system throttling good enough?" *The Ninth International Conference on Web-Age Information Management*, pp. 397–404, July 2008.

[18] M. I. D. Suleiman and I. Hamarash, "Dynamic voltage and frequency scaling (DVFS) for microprocessor power and energy reduction," *4th International Conference on Electrical and Electronics Engineering*, December 2005.

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

[20] T. K. N. Bansal and K. Pruhs, "Speed scaling to manage energy and temperature," *Journal of the ACM (JACM)*, vol. 54, March 2007.

[21] Task Graphs For Free. [Online]. Available: http://ziyang.eecs.umich.edu/ dickrp/tgff/