Accepted to the 10th International Green and Sustainable Computing Conference, October 21 -24, 2019, Alexandria, VA, USA.

Naïve Bayesian based 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@pu.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*—Directed acyclic graph (DAG) is a popular representation of an application, which includes application characteristics such as task dependencies, task execution time and inter processor communication time. Temperature and Energy aware Dynamic Level Scheduling (TEDLS) algorithm has been shown to minimize energy consumption and processor temperature while executing DAG structured applications in a heterogeneous environment with DVS enabled processors. To improve selection process of ideal processor-speed combination, this paper provides a Naïve Bayesian Classifier (NBC) based scheme. Our result demonstrates an improved energy consumption resulting from lower computational overhead.**

***Keywords*-Heterogeneous computing; DVS; DAG; Scheduling; Temperature and Energy aware; TEDLS; Naive Bayesian Classifiers**

I. Introduction

Modern computer applications are quite complex in nature and consist of tasks that vary widely in complexity. If simple tasks are scheduled on high performance processors, they will likely get executed ahead of the deadline, while consuming excessive energy. Energy efficient cores on the other hand, can execute the same tasks by utilizing the total time available for task execution. Thus, to cater to the computational needs of modern computer applications, an increasing number of computing systems are now required to be heterogeneous in nature [1].

In spite of the heterogeneous nature of modern computing systems, individual annual electrical energy consumption of data centers in the U.S. is expected to increase to approximately 140TWh by 2020 [2]. This is equivalent to the annual output of 50 power plants with an undesirable annual carbon pollution of about 100 million metric tons. Therefore, to reduce the impact of new data centers on the environment and also to make them more economically viable, researchers [3, 4, 5, 6, 7] have proposed schemes to reduce energy consumption in a heterogeneous environment. Processors continue to consume a significant amount of system's total power [8]. Thus, by lowering processor power consumption, power required by data centers can be significantly reduced. Total electrical power dissipated in a processor is the combination of the static and dynamic power. Although, static power is dependent on process technology [9], per Equation 1, dynamic power [10] is significantly dependent on supply voltage V_{dd},

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

Dynamic Voltage Scaling (DVS) has allowed researchers to vary the supply voltage and thus enable processors to run on various power budgets. Today, most modern processor manufactures have developed DVS enabled processors [11, 12]. Since processor clock frequency and supply voltage have a linear correlation [7], power consumption is also related to execution time, as indicated by Equation 2 [7]. Lower power consumption can have a trade-off in the form of higher application execution time.

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

Thus, researchers have utilized DVS capability of these processors to reduce the energy usage at the cost of increased execution time.

Since energy efficient processors tend to be used more frequently, they have the tendency to operate at a higher temperature, which has a side effect of rise in transient and permanent faults [13, 14]. This has led a multitude of researchers [13, 14, 15, 16] to use thermal sensors to acquire processor temperatures to manage the overheated processors. However, readjusting speed of DVS enabled processors based on temperature readings can introduce delays and cause inaccurate speed throttling. Also, one major drawback in these approaches is the lack of emphasis on energy usage. In [17], a heuristic scheme was proposed to efficiently schedule tasks in DAG structured applications onto heterogeneous DVS enabled processors in order to minimize execution time, energy usage, and processors' temperature. Such scheduling has been shown to be NP-Complete [17, 18].

The scheme in [17] is rooted in Dynamic Level Scheduling (DLS) algorithm [19], which is a non-preemptive compile scheduling heuristic that uses a cost function to minimize execution time of DAG structured application by balancing the entire workload in a heterogeneous environment with DVS enabled processors. It also takes care of the inter processor communication (IPC) and has a low operational overhead. Thus, in [7] the Energy Dynamic Level Scheduling (EDLS) algorithm was developed that focused on minimizing both application execution time and energy consumed. An improvement over this algorithm was made in [17], where the Temperature and Energy aware Dynamic Level Scheduling (TEDLS) algorithm was proposed. The algorithm is focused on minimizing energy consumption, processor temperature and application execution time. The simulation results showed that, with respect to energy consumption and processor temperature, the TEDLS algorithm scaled better with increased size of the DAG in comparison to the DLS and EDLS algorithms. In TEDLS algorithm, a heat model from [20] was used to predict the final temperature of processors while running a given task. As the heat model used energy consumption for estimating the processor temperature, the temperature value was used to modify the cost function to incorporate aspects of both energy consumption and processor temperature. The TEDLS algorithm schedules tasks to the most energy efficient processor as long as it is also one of the cooler processors.

The static implementation of the TEDLS algorithm in [17] provided many possible cases of processor speed combinations with varying execution times, energy consumption and processor temperatures. In a practical scenario, where there are thousands of DVS enabled processors, the number of cases would be significant. To deal with the complexity, in this paper, we have implemented a machine learning based approach capable of producing the best choice for processor speed combinations for a given energy consumption and processor temperature budget in a dynamic environment. Our focus has been to come up with a simple yet effective way to achieve this. One popular classification algorithm, called Naïve Bayesian Classifier, which is based on Bayesian Networks introduced by Pearl in [21], has already been used for finding the influence relation between genes [22], network traffic analysis [23], text classification [23], anti-spam techniques for email [24], intrusion protection systems [25] and even bankruptcy prediction [26]. It provides a way to come up with the right choice in the shortest amount of time with minimal input data [27]. Thus, our scheme utilizes Naïve Bayesian Classifier to select the processor speed combinations required for application execution.

The rest of the paper is organized as follows. In Section II, we discuss some definitions, notations and background regarding temperature aware task scheduling. Section III provides an overview of the TEDLS scheduling algorithm. In Section IV, we illustrate how Naïve Bayesian Classifiers can be applied to the TEDLS algorithm. We also schedule tasks onto a pool of processors using the TEDLS algorithm by applying the learning algorithm and by random selection, and compare simulation results. Finally, concluding remarks are given in Section V.

II. Background

In this paper, we assume applications are periodic in nature and can be represented in DAG structured forms. Figure 1 depicts a typical DAG application $G = (T, E)$, where each node denotes a task $T_i \in T$, and each edge implies the dependency of associated tasks. If two tasks are performed on separate processors, each weighted directed edge $E_{ij} = (T_i, T_j) \in E$ represents the communication delay associated with sending output of task T_i to the processor executing task T_j. The noted values depicted in Figure 1 are the average communication overhead among the various processors.

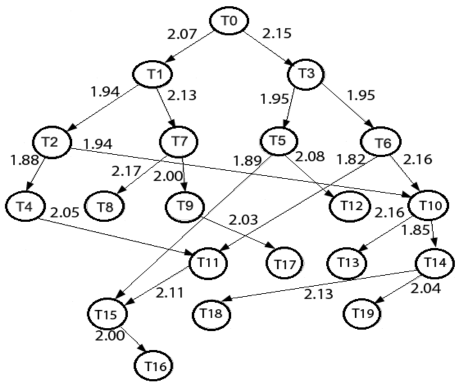
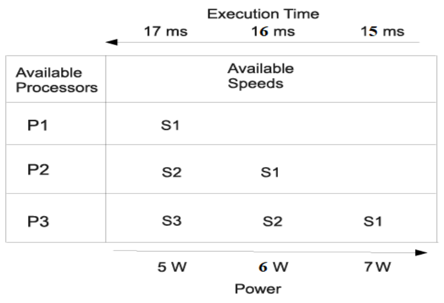


Fig. 1. Example DAG

Figure 2 illustrates a sample pool of three heterogeneous processors $P = \{P_1, P_2, P_3\}$ with different speed settings; $S_{P1} = \{S_1\}$ and $S_{P2} = \{S_1, S_2\}$ and $S_{P3} = \{S_1, S_2, S_3\}$. Each processor-speed setting has an associated power usage and corresponding average execution time.


Execution Time
17 ms
16 ms
15 ms
5 W
6 W
7 W
Power

Available Processors	Available Speeds		
P1	S1		
P2	S2	S1	
P3	S3	S2	S1

Fig. 2. Processor Pool and their associated speed and power settings

The TEDLS algorithm utilizes the processors in Figure 2 to execute tasks in Figure 1. During execution of different tasks by assigned processors, either a processor may not be ready (specified as Processor Ready Time) to receive the new task or the scheduled task may need the result of execution from a dependent task (Data Ready Time). In either case, a delay in the execution of task is needed. For example, in Figure 1, if Task 0 and Task 1 are scheduled to be executed on the same processor, Data Ready Time (DA_{1p}) and Processor Ready Time (TF_{1p}) for Task 1 will both be equal to the execution time of Task 0. However, if Tasks 0 and 1 are scheduled on different processors p' and p, respectively, then DA_{1p} for Processor p will be sum of the execution of Task 0 on Processor p' and time required in transferring the results to Processor p. Another two terminologies that will be used in our discussion of TEDLS algorithm are Static Level (SL_{np}) and Processor Speed Difference (Δ_{np}). The former is the sum of the execution times of tasks along the longest path, from the current task n to the leaf nodes. The latter is the difference in execution time of Task n on Processor p with that of a processor with median task execution times [7].

Temperature and Energy aware Dynamic Level Scheduling (TEDLS) [17] is a list-based scheduling algorithm, which focuses on minimal energy consumption and low processor temperature. The algorithm first prioritizes tasks to be executed and then assigns processors to these tasks based on a cost function specified by Equations 3 [17], 4 [19] and 5 [17].

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

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

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

In determining $TEDL_{np}$ scheduling cost function, $NormTemp$ is used to penalize the heated processors. Its value is obtained by dividing the predicted final temperature T_{np} by a maximum operating temperature $MaxTemp$ (set by the manufacturer). A cooler processor will have a lower $NormTemp$, resulting in a higher $TEDL_{np}$ and a more favorable opportunity for a task assignment. The heat model specified by Equation 6 is used to estimate the processor's temperature $T_{np}(t_2)$ for an interval $(t_2 - t_1)$ [20]. In the model, β and ρ represent the thermal resistance and the cooling constant of the processor, respectively. P_{np} is the power consumption of Task n on Processor p. The model takes into account the rise in temperature when the processor is active as well as the cooling effect, due to Newton's law of cooling [20]. The TEDLS scheduling algorithm is given by Algorithm 1.

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

III. Overview of TEDLS Algorithm [17]

In this section, through the example DAG application of Figure 1 and processor pool of Figure 2, we provide an overview of the TEDLS algorithm. We generate DAG applications using TGFF [28] software. For the sake of simulation, we randomize the execution times and consumed power for each of the 20 tasks in Figure 1 for Processors 1, 2 and 3 within $\pm 10\%$ of the nominal values in Figure 2. Based on these randomized values, we come up with the Median Execution Times, which is shown in in Table I. Also, using the Median Execution Times we calculate the Static Levels for various tasks. 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
Static Level Table

Task Number	Median Times	Static Levels
0	15.71	116.72
1	15.86	101.01
2	17.12	85.15
3	16.17	83.86
4	17.36	68.03
5	15.86	49.53
6	16.10	67.69
7	15.76	48.83
8	16.54	16.54
9	17.53	33.07
10	16.01	51.59
11	17.00	50.67
12	17.78	17.78
13	16.42	16.42
14	18.00	35.58
15	17.85	33.67
16	15.82	15.82
17	15.53	15.53
18	17.58	17.58
19	16.96	16.96

According to Algorithm 1, we first need to calculate the static level (SL) and the processor speed difference (Δ) of all tasks. SL gives priority of execution to the longest task path. Δ, which is the relative speed of a processor to the median processor ($P_2@S_1$), gives priority to the faster processor for task execution.

Algorithm 1 (TEDLS)

```
Calculate Static Level and $\Delta$ for every task
while $\exists$ unscheduled task do
    Make list of Ready Tasks
    Estimate $T_{np}(t_2)$ for Ready Tasks on Processor $p$ using Equation (6)
    Calculate $NormTemp$ using Equation (5)
    Calculate $TEDL$ for Ready Tasks using Equation (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 Figure 1, Task 0 has 7 end tasks (i.e. leaf nodes in the sub-tree rooted in Task 0 node). SL for Task 0 can be obtained by taking the sum of the median execution times of the tasks along the longest execution path, namely, $T_0 \rightarrow T_1 \rightarrow T_2 \rightarrow T_4 \rightarrow T_{11} \rightarrow T_{15} \rightarrow T_{16} \Rightarrow 15.71 + 15.86 + 17.12 + 17.36 + 17.00 + 17.85 + 15.82 = 116.72\ ms$. Initially, the task ready for execution in Figure 1 is T_0. After this task is scheduled, T_1 and T_3 get tagged as ready tasks, and are subsequently scheduled. This process is repeated until there is no additional task left to be scheduled.

The scheduling of tasks starts with the prediction of final temperatures of different processors after executing the first ready task. For example, the predicted final temperature for T_0 is calculated using the heat model provided by Equation 6 and randomized execution times of tasks for respective processor-speed combinations. For the heat model, we use a typical value of β = 1 J/K and ρ = 0.009999 K/J as per [29]. The initial temperature is chosen to be 313.15 K (40 °C). Table II shows the final temperature values for Task 0. Once final temperatures are determined, the $NormTemp$ values can be calculated using Equation 5. These values are shown in Table III. We set $MaxTemp$ = 358.15 K (85 °C), per [14].

Table II
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	4.57	0.0099	313.15	15.55	333.88
Processor 2						
0	1	5.49	0.0099	313.15	14.64	345.27
Processor 3						
0	1	6.40	0.0099	313.15	13.72	355.07

Table III
STEP 1B OF THE TEDLS ALGORITHM

Task	$T_{np}(t_2)$	$MaxTemp$	$NormTemp$
Processor 1			
0	333.88	358.15	0.93
Processor 2			
0	345.27	358.15	0.96
Processor 3			
0	355.07	358.15	0.99

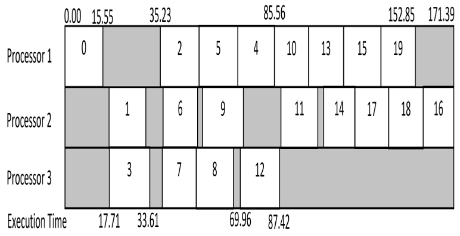
Next, $TEDL_{np}$ is calculated using Equation 3 and the result is tabulated in Table IV.

Table IV
STEP 1C OF THE TEDLS ALGORITHM

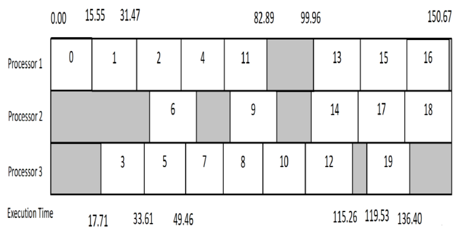
Task	SL	DA	TF	Δ	DL	$NormTemp$	TEDL
Processor 1							
0	116.72	0.0	0.0	-0.91	115.80	0.93	123.65 $\Leftarrow$
Processor 2							
0	116.72	0.0	0.0	0.00	116.72	0.96	120.91
Processor 3							
0	116.72	0.0	0.0	0.91	116.66	0.99	118.64

Given no task has been assigned to any processor yet, DA_{0p} = TF_{0p} = 0 ms. Per Table IV, Task 0 has the highest value of $TEDL$ and yields the lowest value of $NormTemp$ for Processor 1. Hence, T_0 is assigned to Processor 1. Per Figure 1, the next ready tasks are Task 1 and Task 3. Similar tables can be easily generated for these and subsequent ready tasks.

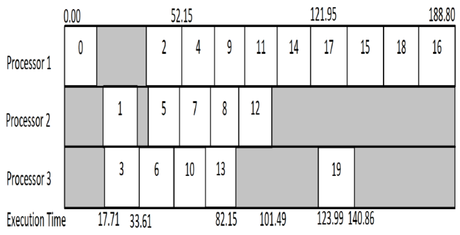
Figure 3(a) shows the resulting scheduling diagram for the example DAG.



(a) TEDLS algorithm



(b) DLS algorithm



(c) EDLS algorithm

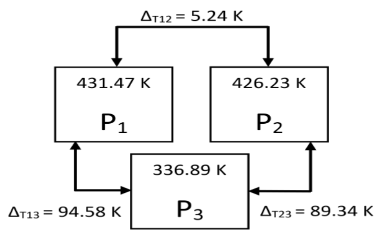
Fig. 3. Scheduling of Processors under the TEDLS, DLS and EDLS algorithms

For comparative analysis, we applied both the DLS and EDLS algorithms to the DAG of Figure 1. Figures 3(b) and 3(c) show the resulting scheduling of tasks under these algorithms, respectively.

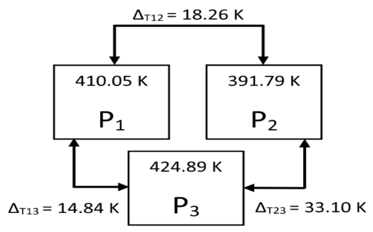
$$E_n = \sum Power_{np} \times ExecTime_{np} \tag{7}$$

We compared the consumed energy under the three algorithms, per Equation 7, where, $Power_{np}$ and $ExecTime_{np}$ are the power consumption and execution time for Task n on Processor p, respectively. Energy consumption of processors under the TEDLS algorithm is calculated to be 1.62 J. This amounts to energy saving of about 45.69 % and 32.08 % in processors compared to when task scheduling is done using the DLS and EDLS algorithms, respectively. The execution time of the application under the TEDLS algorithms is 171.39 ms, versus 150.67 ms (-12.09%) and 188.80 ms ($+10.16\%$), under the DLS and EDLS algorithms, respectively.

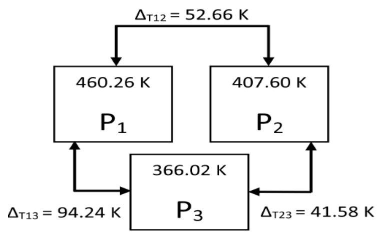
We next used the heat model in Equation 6 to predict the final processor temperatures. Figure 4 shows the predicted final temperatures for processors scheduled with the TEDLS, DLS and EDLS algorithms. The goal of this prediction is not to find the exact value of processor temperatures, but rather to compare how processor temperatures vary when they are scheduled with different scheduling algorithms. It can be seen from Figure 4 that the final temperature of processors scheduled using the TEDLS algorithm is 5.71% lower than the processors that are scheduled using the DLS algorithm and 3.21% lower than the processors that are scheduled using the EDLS algorithm. Thus, the TEDLS algorithm results in lower temperature distribution among the processors, making them less susceptible to hardware faults.



(a) TEDLS algorithm



(b) DLS algorithm



(c) EDLS algorithm

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

IV. Application of Naïve Bayesian Classifier (NBC) to TEDLS and Comparative Analysis

The example in Section III illustrated only processor speeds $P_1@S_1$, $P_2@S_1$ and $P_3@S_1$. However, the pool of processors in Figure 2 can have sixteen other processor-speed combinations. Thus, for a particular instance of application execution, there needs to be a single processor-speed combination that meets the energy, temperature and time budgets. Making such a decision becomes even more difficult in the presence of large number of processors with variable speed settings. In this section, we use the Naïve Bayesian Classifier (NBC) twice to speed up the process as directed below. First, NBC is used to obtain a set of processor-speed combinations that result in low energy consumption. Next, NBC is used again to obtain a processor-speed combination that produce the lowest processor temperatures from the set of combinations with low energy consumption.

Naïve Bayesian Classifier was chosen for its lack of complexity, shorter training time, less sensitivity to inaccurate data and for its applicability in multi-class classification problems. The algorithm is based on the Bayes' theorem, which is given by Equation 8, where $P(A|B)$ is posterior probability, $P(A)$ is prior probability, $P(B|A)$ is likelihood probability and $P(B)$ is probability of evidence.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{8}$$

One generic example of a Naïve Bayesian Classifier is given in Figure 5. Here, the node labeled "Label" is the object that has to be classified based on the features given in the nodes f_1, f_2 ... f_n.

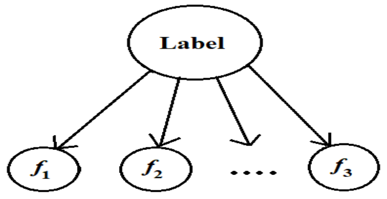


Fig. 5. Typical Naïve Bayesian Network

According to Bayes theorem the posterior probability of "Label" belonging to the class "X" is given by Equation 9.

$$P(X|f_1, f_2 ... f_n) = \frac{P(X) \times P(f_1, f_2 ... f_n)|X)}{P(f_1, f_2 ... f_n)} \tag{9}$$

Since the features are assumed to be independent in NBC, this equation can be rewritten as Equation 10, where Z is the probability of evidence and is constant.

$$P(X|f_1, f_2 ... f_n) = \frac{1}{Z} \times P(X) \times \prod_{i=1}^{n} P(f_i|X) \tag{10}$$

In our case, we initially choose processor combinations which consume the least amount of energy, with the minimum number of iterations. Here, the object to be classified is Energy and the influencing factors are the different processor speed combinations (C_1, C_2 ... C_l). As the DAG application remains constant, it is not thought of as an influencing factor. Thus, using NBC, if we want to find $P(E|C_i)$ (i.e. Energy class for a given set of processor speed combinations), based on Equation 10, we come up with Equation 11. Here, we have ignored probability of evidence, as it is only a scaling factor. In Equation 11, C_i denotes a particular processor-speed combination; e.g. $P_1@S_1$, $P_2@S_1$ and $P_3@S_1$. l is the total number of processor-speed combinations. For simplicity and accuracy, our classifier has three distinct classes. Based on Equation 11, the probabilities of different Energy (E) classes given a certain combination C_i is calculated. C_i is linked to the Energy (E) class that results in the highest value of probability.

$$P(E|C_i) = P(E) \prod_{k=1}^{l} P(C_k|E) \tag{11}$$

Once, the lowest energy consuming combinations are identified, a second NBC, described using Equation 12, is applied on these combinations. It classifies these combinations based on processor temperature to obtain the combination with the lowest processor temperature.

$$P(T|C_i) = P(T) \prod_{k=1}^{l} P(C_k|T) \tag{12}$$

To implement either of the learning schemes described in Equation 11 or 12, we need to create two tables, "TestSet" and "DataSet". Figure 6 shows the block diagram of our scheme, which utilizes tables "TestSet" and "DataSet".

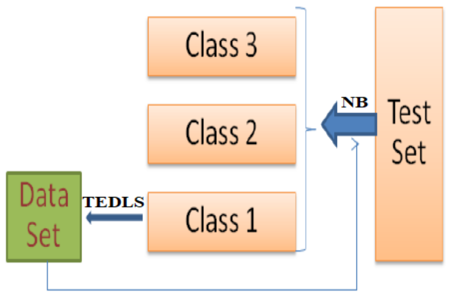


Fig. 6. Block diagram of our learning scheme

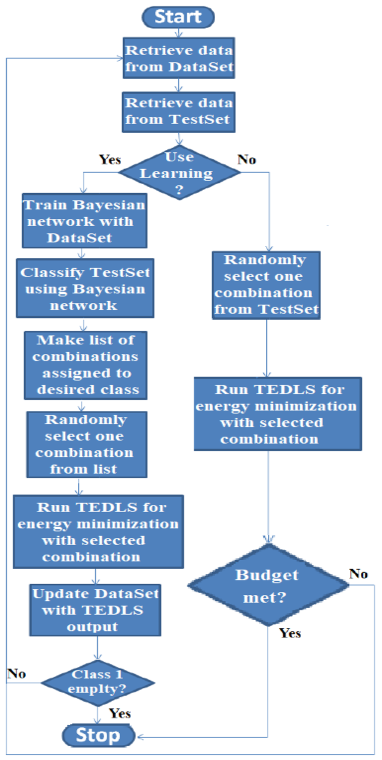
"TestSet" contains all combinations for which the resultant energy consumption or processor temperature is not known. The "DataSet" contains processor combinations, which have been examined by the TEDLS algorithm as well as their associated consumed energy or processor temperature. DataSet is used to train the Bayesian Network. Three distinct classes or bins are defined to classify each combination. Each energy or temperature class represents a range of energy or temperature values. Steps within the loop are executed at least once before the evaluation of the while statement. In the beginning of the do-while loop, the classifier trains itself using the processor data available on the DataSet. Next, the classifier classifies all available entries in the TestSet. In the following step, a random combination is selected from the lowest class (Class 1) combinations. Next, the TEDLS algorithm is applied on the selected combination. In Step 6, the results are stored in the DataSet. In the final step, the algorithm checks for the presence of any combination in Class 1. As long as there is a combination assigned to Class 1, the loop repeats. Otherwise, the loop terminates. At this point, the DataSet can be checked to obtain a group of combinations with the lowest energy consumption. The effect of this learning algorithm can be compared with random selection to find the same group of processor-speed combinations. In random selection, the TEDLS algorithm is applied to all combinations at random and their energy values are obtained until a certain energy budget is met. Figure 7 shows the flowchart of the simulation. Algorithm 2 gives an overview of the learning algorithm for obtaining lowest energy consuming combinations.

Algorithm 2 (Learning Algorithm for obtaining low energy consuming processor-speed combinations)

```
1: do :
2: Train Bayesian Network with DataSet
3: Apply Bayesian Learning on TestSet
4: Select 1 processor combination from lowest Energy class
5: Run TEDLS with the selected combination
6: Update DataSet with new information $TEDL$
7: while a combination is assigned to Class 1 (Lowest class)
```

For simulation purposes, we have applied the learning algorithm to schedule the DAG application in Figure 1 onto the processor pool P in Figure 2. A group of processor speed combinations with low energy consumption is generated using Algorithm 2. The energy budget is set to $< 0.4J$, which is lower than the upper limit of the lowest class (i.e. Class1). Simulation with our learning scheme and random selection were run 100 times each. Figure 9 shows the simulation results, where we compared the number of iterations required to come up with processor-speed combination for minimal energy consumption for a 20 task DAG. Our simulation result demonstrates that initially both the learning algorithm and random selection have the same performance. As the NBC network gets trained, the learning scheme arrives at the processor-speed combination with the lowest energy consumption in 9 iterations, compared with 15 iterations for the random selection.

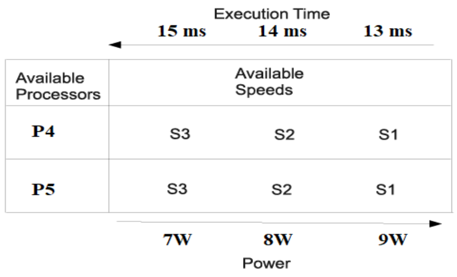
Next, the second NBC is applied to the group of processor-speed combinations with low energy consumption. The objective is to obtain a particular combination that generates the lowest processor temperatures for application execution. For achieving this, NBC is applied in a similar fashion as in Algorithm 2. However, for Step 4, one processor combination from lowest Temperature class is selected. Once, classification terminates, combinations that produces temperatures closest to $75\,°C$ [14] are chosen for application execution. The flowchart of the simulation of this algorithm is similar to that in Figure 7. The average processor temperature budget is set to $< 75\,°C$ [14], which is lower than the upper limit of the lowest class (i.e. Class1). Once again, simulation with our learning scheme and random selection



Start
Retrieve data from DataSet
Retrieve data from TestSet
Use Learning ?
Yes
No
Train Bayesian network with DataSet
Classify TestSet using Bayesian network
Make list of combinations assigned to desired class
Randomly select one combination from list
Run TEDLS for energy minimization with selected combination
Update DataSet with TEDLS output
Class 1 empty?
No
Yes
Stop
Randomly select one combination from TestSet
Run TEDLS for energy minimization with selected combination
Budget met?
No
Yes

Fig. 7. Flowchart of Simulation for arriving at minimal energy consumption combinations

were run 100 times each. Figure 10 shows a comparison of number of iterations to come up with processor-speed combination for minimal processor temperature for a 20 task DAG. Our simulation result demonstrates that the learning algorithm arrives at the processor-speed combination with the lowest processor temperature in 10 iterations, compared with 14 iterations for the random selection.



	Execution Time		
	15 ms	14 ms	13 ms
Available Processors	Available Speeds		
P4	S3	S2	S1
P5	S3	S2	S1
Power	7W	8W	9W

Fig. 8. Processor Pool and their associated speed and power settings

Both Figures 9 and 10 show reduction in computational steps to arrive at the ideal processor-speed combination. However, to illustrate the scalability of our learning scheme, we next applied randomly generated 100 tasks and 200 tasks DAGs to a processor pool of $P\prime = \{P_1, P_2, P_3, P_4, P_5\}$, where P_4 and P_5 are high performance processors, as depicted in Figure 8. Once again, for the sake of simulation, we randomized the execution times and consumed power for each of the 100 tasks and 200 tasks in larger DAGs for Processors 1, 2, 3, 4 and 5 within $\pm 10\%$ of the nominal values in Figures 2 and 8. Figures 11 and 12 show comparisons of number of iterations to come up with processor-speed combination from $P\prime$ for minimal processor temperature for 100 task and 200 task DAGs respectively. The average processor temperature budget is set to $=< 80\,^{\circ}\mathrm{C}$ [14]. The figures show that initially both the learning algorithm and the random selection has low performance. This is because the Bayesian network is yet to be trained adequately. As the number of steps increases and the classifier trains itself, the learning algorithm arrives at the ideal processor-speed combination for the 100 task DAG in an average of 243.6 steps, compared with 369 steps for the random selection. This implies that our learning scheme is approximately 33.98% faster than random selection in this case.

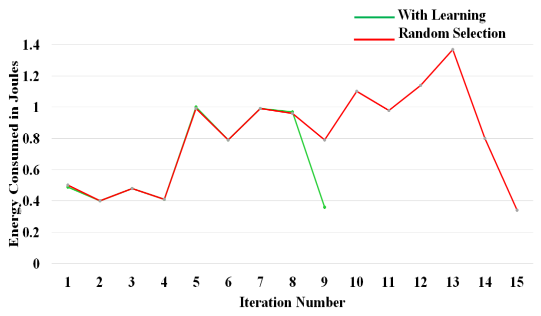


Fig. 9. Single case results of learning algorithm for obtaining low energy consuming processor-speed combinations

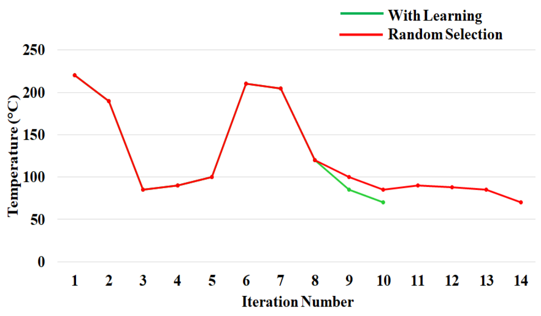


Fig. 10. Single case results of learning algorithm for obtaining processor-speed combinations with low processor temperature

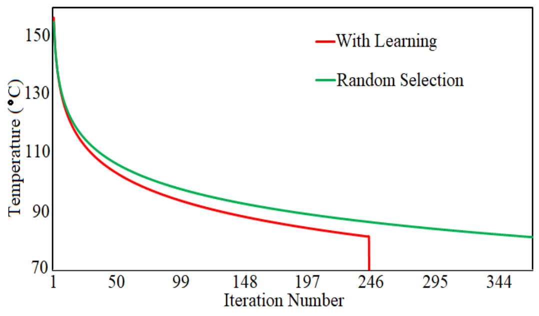


Fig. 11. Processor speed combination selection for 100 task DAG with learning and random selection

Similarly, for 200 task DAG, our learning scheme arrives at the ideal combination at an average of 220.7 steps, which when compared to random selection is 40.18% faster. Reduction in steps to arrive at the ideal processor-speed combination implies lower computational overhead. Thus, this results in overall reduction of energy consumption during compile time.

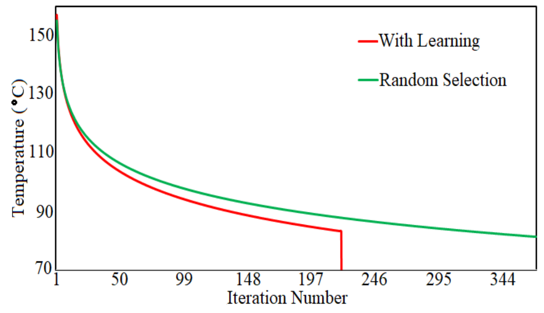


Fig. 12. Processor speed combination selection for 200 task DAG with learning and random selection

V. Conclusion

In this paper we implemented Naïve Bayesian Classifiers (NBCs) to an offline algorithm that not only helps in achieving low processor temperatures, but also produces lower energy consumption compared to the energy efficient EDLS algorithm. We implemented NBCs to arrive at an ideal processor-speed combination that produces the least amount of processor temperature and energy consumption. Our results show that for 100 task and 200 task DAGs our learning based approach arrives at the ideal processor-speed combination more than 30% faster compared to when random selection is used. As the computational overhead decreases, this results in overall energy minimization during program compilation.

References

[1] M. Zahran, "Heterogeneous computing: Here to stay," *ACMQUEUE*, Volume 14 Issue 6, November-December 2016.

[2] D. A. M. M. A. Mohammed and J. M. Abdullah, "Green computing beyond the traditional ways," *International Journal of Multidisciplinary and Current Research*, Vol.3 (July/Aug 2015 issue) 2015.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] J. Butts and G. Sohi, "A static power model for architects," *33rd Annual IEEE/ACM International Symposium on Microarchitecture*, 2000.

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

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

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

[13] 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.

[14] 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.

[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] R. Kabir and B. Izadi, "Temperature and energy aware scheduling of heterogeneous processors," *2016 Ninth International Conference on Contemporary Computing (IC3)*, pp. 48 – 54, August 2016.

[18] 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.

[19] 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.

[20] 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.

[21] J. Pearl, *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufmann, 1988.

[22] M. Mascherini, "Learning the structure of bayesian networks representing influence relations among genes," *2008 International Conference on Computational Intelligence for Modelling Control Automation*, pp. 1023 – 1028, December 2008.

[23] M. Z. D. Liu and T. Li, "Network traffic analysis using refined bayesian reasoning to detect flooding and port scan attacks," *2008 International Conference on Advanced Computer Theory and Engineering*, pp. 1023 – 1028, December 2008.

[24] L. G. Y. Li, B. Fang and S. Wang, "Research of a novel anti-spam technique based on users' feedback and improved naive bayesian approach," *International conference on Networking and Services (ICNS'06)*, July 2006.

[25] J. L. K. Cheng and C. Zhang, "Rough set weighted nave bayesian classifier in intrusion prevention system," *2009 International Conference on Networks Security, Wireless Communications and Trusted Computing*, April 2009.

[26] A. Aghaie and A. Saeedi, "Using bayesian networks for bankruptcy prediction: Empirical evidence from iranian companies," *2009 International Conference on Information Management and Engineering*, April 2009.

[27] S. Taheri and M. Mammadov, "Learning the naive bayes classifier with optimization models," *International Journal of Applied Mathematics and Computer Science*, December 2013.

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

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