Energy-Efficient Scheduling Algorithms

Undergraduate Thesis

Submitted in partial fulfillment of the requirements of
BITS F422T Thesis

By

Jaskamal Kainth
ID No. 2013B4A70586P

Under the supervision of:

Dr. Abhishek Mishra

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI, PILANI CAMPUS
May 2018

Declaration of Authorship

I, Jaskamal KAINTH, declare that this Undergraduate Thesis titled, 'Energy-Efficient Scheduling Algorithms' and the work presented in it are my own. I confirm that:

- This work was exclusively done during candidature for a research degree at this University.
- Where any part of this thesis has formerly been submitted for a degree or this has been clearly stated that any part of this thesis has formerly been submitted.
- Where I have referred to the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always provided. exclusive of these quotations, this thesis is entirely my own work.
- I have acknowledged all the major sources of help.
- Where the thesis is the result of work done by myself, I have made clear exactly what was done by others and what I have done myself.

Signed: ______________________________

Date: ______________________________

Certificate

This is to certify that the thesis entitled, "*Energy-Efficient Scheduling Algorithms*" and submitted by Jaskamal KAINTH ID No. 2013B4A70586P in partial fulfillment of the requirements of BITS F422T Thesis embodies the work done by him under my supervision.

Supervisor
Dr. Abhishek MISHRA
Asst. Professor,
BITS-Pilani Pilani Campus
Date:

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI, PILANI CAMPUS

Abstract

M.Sc. (Hons.) Mathematics and B.E (Hons) Computer science

Energy-Efficient Scheduling Algorithms

by Jaskamal KAINTH

In this report, we discuss various energy efficient scheduling algorithms for real time parallel applications on a heterogeneous distributed embedded system. We start with the basic introduction of the parallel application which is represented as a Directed acyclic graph. Then various commonly used terms like WCET, WCRT are explained and introduction ends with a mathematical problem statement which is to minimize the energy consumption by the tasks on the heterogeneous processors.
After the basic introduction, various non-DVFS scheduling algorithms like HEFT, Deadline slack , NDES algorithms are discussed alongwith the result on a sample test task graph. then a DVFS-enabled scheduling algorithm ,GDES(Global DVFS-enable scheduling algorithm) is discussed and shown that when combined with a non-DVFS algorithm, the energy consumption is minimized.
After the discussion of all the algorithms and their results on a test task graph which represents a parallel application, analysis is done on various other task graphs like FFT, gaussian, divide and conquer and systolic array task graph. In the end it is shown that GDES&NDES produces the best results i.e the energy consumption is the least in this case as compared to all other scheduling algorithms.

Acknowledgements

I would like to express my sincere gratitude to my advisor Prof. Abhishek Mishra for his regular support of my thesis study and related research, for his time, guidance and immense knowledge in this area. I could not have imagined having a better advisor and mentor for my thesis study. Beside my advisor, i would also thank my B.E(Hons) Computer science batch mates who helped me in correcting, implementing my code and analyzing the results which i got. Finally, I would like to thank Computer science department at BITS Pilani, Pilani campus for allowing me to do this thesis study and evaluating this project report.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

The problem of minimizing the energy consumption of a real-time application with precedence constrained tasks mostly focus on homogeneous systems, whereas heterogeneous multiprocessors or multi-cores continue to scale up and increasingly become the key components of distributed platforms.

Our main aim is to develop an algorithm which minimizes the energy consumption of a real time parallel application (constrained by precedence tasks) on **heterogeneous distributed systems**.

Deadline is an important design constraint for real-time applications in embedded systems, missing the hard deadlines of these applications is not functional correctly and will cause catastrophic consequences. Therefore, the deadline must be satisfied for a hard real-time application. Parallel applications with precedence constrained tasks, such as fast Fourier transform and Gaussian elimination applications, are increasing in number in heterogeneous distributed systems. There are some typical models describing a parallel application with precedence constrained tasks, such as **directed acyclic graph (DAG)**, hybrid DAG (HDAG), and task interaction graph (TIG). In this study, a parallel application is represented by a DAG, in which the nodes represent the tasks and the edges represent the communication messages between tasks

The first algorithm which study is **HEFT** (heterogenous earliest finish time). The lower bound or the minimum schedule length of the application graph when all the tasks are executed on the heterogeneous distributed system with the maximum frequency and this can be compute using the HEFT algorithm.

The second algorithm with which we achieve the above is the **deadline slack algorithm**. In this, we introduce the concept of deadline slack with which we assign the tasks to the processors by minimizing the total dynamic energy consumption and satisfying the deadline constraint as much as possible.

The third energy efficient scheduling algorithm is the **NDES algorithm** (non-DVFS). In this we introduce a new concept of variable deadline slack with which we try to minimize the total energy consumption by iteratively calling the deadline slack algorithm such that the deadline of the application graph is satisfied as in the previous algorithm it may happen that the schedule length of the application graph gets a higher value than the given deadling of the application.

All the above mentioned algorithms were non-DVFS(dynamic voltage frequency scaling) scheduling algorithms. To get better results that means to reduce the energy consumption , there is **GDES** which is global DVFS enabled scheduling algorithm. In this algorithm we scale down the frequencies and assign to the processors with more running time so as to reduce the slack between the tasks on the same processor. In the end we will show that the combination of **GDES&NDES** algorithm gives the least energy consumption among all the other scheduling algorithms.

1.1 System architecture and Application models

We will consider a heterogeneous distributed embedded architecture in which multiple processors are mounted on the common CAN (control area network) bus. one example is shown in the figure below.

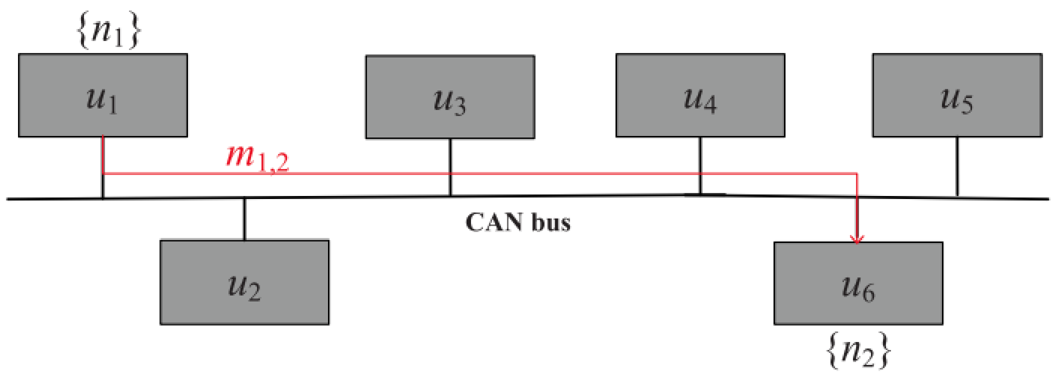

FIGURE 1.1: Heterogeneous distributed embedded system

Every processor in this system will contain a CPU, a random access memory , a non-volatile memory and a NIC (network interface card).

A task which is represented as a node in the application graph executes in onr processor and then sends messages to all of its immediate successor task which might be located on different processors.

The parallel application which runs on this embedded systems is represented by a directed acyclic graph (DAG) G. One example of such DAG is shown in the figure.

U represents the set of heterogeneous processors and $|U|$ is the size of this set.

The number of nodes N is total number of tasks which is to be scheduled where each node $n_i \in N$ represents a task. $pred(n_i)$ is the set of immediate predecessor tasks of n_i , similarly $succ(n_i)$ is the set of the immediate successor tasks of n_i. These are used while implementing the algorithm. The entry task or the the task with no predecessor is denoted by n_{entry} similarly the exit task or the task with no successor is represented by n_{exit}. We can add a dummy entry task or a dummy exit task with zero edge weights in case our parallel application does not have them.

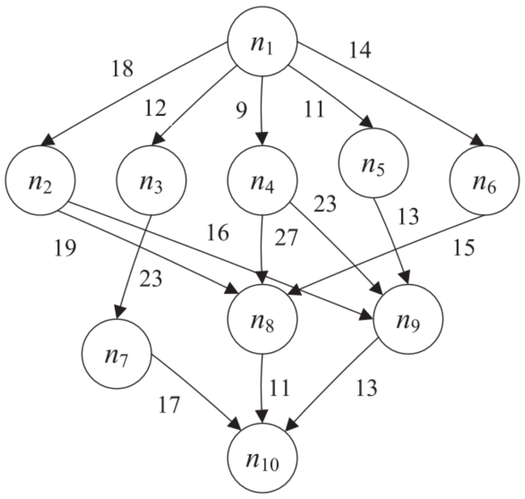

FIGURE 1.2: A parallel application represented by a DAG

Now, each edge in the application graph represents the communication message from n_i to n_j and the corresponding edge weight value is the **WCRT (worst case response time)** that is, the amount of time to transfer the message from n_i to n_j if both the tasks are scheduled in a different processor.

We have a W matrix of size $|N| \times |U|$ where each element Wi, j denotes the **WCET (worst case execution time)** time taken to run the i_{th} task on the j_{th} processor with maximum frequency. The table 6.4 shows the time taken by every task for each processor.

$D(G)$ represents the deadline of the parallel application graph which should be larger than the $LB(G)$ (Lower bound on schedule length). The lower bound refers to the minimum schedule length of the application when all the task are run on the processors with maximum frequency.

This schedule length can be computed using a well studied HEFT (Heterogeneous earliest finish time) algorithm which is explained later.

Task(n_i)	u_1	u_2	u_3
n_1	13	16	8
n_2	14	20	18
n_3	12	13	9
n_4	13	8	17
n_5	12	13	10
n_6	15	16	9
n_7	4	16	11
n_8	6	11	14
n_9	14	12	20
n_{10}	11	7	16

TABLE 1.1: WCET(Worst case execution time) matrix

Now, the power model consists of 3 type of powers.Static P_s , Dynamic P_d and frequency independent dynamic power P_{id}. P_s can be removed only after we shut down the entire embedded system.P_{id} can be removed only when we put the system to sleep mode. The P_d power is the dynamic power which depends on the frequency.

The power consumption at the frequency f is represented by

$$P(f) = P_s + h(P_{id} + P_d) \tag{1.1}$$

where h = 1 if the system is active else 0.

The total energy consumption of the parallel application is the sum of E_s and E_d.

E_s represents the static energy consumption of the application G and is calculated by

$$E_s(G) = \sum_{i=1}^{|U|}(P_{k,s} \times SL(G)) \tag{1.2}$$

E_d represents the dynamic energy consumption of the application G and is calculated by

$$E_d(G) = \sum_{i=1}^{|N|} E_d(n_i, u_{pr(i)}) \tag{1.3}$$

where if we run the i_{th} task on the j_{th} processor then the dynamic energy consumption is ,

$$E_d(n_i, u_j) = (P_{j,ind} + C_{j,ef} \times f_{j,max}^{m_j}) \times W_{i,j} \tag{1.4}$$

Where $C_{j,ef}$ is the effective capacitance of the j_{th} processor, m_j represents the dynamic power exponent. Various power parameters related to the heterogeneous processors are shown in the 1.2.

Processor(u_k)	$P_{k,s}$	$P_{k,d}$	$C_{k,ef}$	m_k	$f_{k,low}$	$f_{k,max}$
u_1	0.01	0.02	1.30	2.90	0.19	1.0
u_2	0.01	0.05	0.50	2.10	0.32	1.0
u_3	0.01	0.04	0.20	3.00	0.46	1.0

TABLE 1.2: Power parameters of the Processors

1.2 Problem Description

The problem reduces to assign the tasks to the processors with certain frequencies and thus minimizing the total energy consumption of the application graph in this process such that the deadline constraint and the frequency constraint is satisfied.

$$E_{total}(G) = E_s(G) + E_d(G) \tag{1.5}$$

such that $SL(G) \leq D(G)$, where $D(G)$ is the deadline set by the user
and $f_{pr(i),low} \leq f_i \leq f_{pr(i),max}$, where $pr(i)$ is the processor index assigned to the i^{th} task.

Chapter 2

The HEFT Algorithm

As, scheduling tasks of the parallel application on multiprocessors is a NP-Hard optimization problem so this is the first Heuristic algorithm with which we will get an initial mapping or distribution of task on the heterogeneous distributed system such that deadline and frequency constraint is satisfied while achieving lowest energy possible.
HEFT is a type of List scheduling algorithm and consist of two phases: task prioritization and task allocation. A DAG is used to represent a parallel application and HEFT is used to obtain a lower bound of a parallel application which will be used by other algorithms presented later in the thesis. With this lower bound value we can provide a deadline of the graph which is greater than or equal to the lower bound of the graph achieved by the HEFT algorithm.
The two phases of the algorithm are:

1. *Task prioritization*: Every task which is represented by a node of the graph is given a rank values which is computed using the equation.

   $$rank(n_i) = \overline{W_i} + \max_{n_j \in succ(n_i)} C_{i,j} + rank(n_j) \tag{2.1}$$

   where $\overline{W_i}$ is the average WCET(Worst case execution time) value of the task n_i and is given by,
   $\overline{W_i} = (\sum_{j=1}^{|U|} W_{i,j})/(|U|)$ where $|U|$ is the number of processors.

   The rank equation is a recursive equation and hence we can use Depth first search on the given DAG to compute the ranks of every tasks in linear time O(number of nodes = tasks). Listing 2.1 shows the DFS approach to compute the ranks.

```
long double calculate_rank(int u)
{
    long double val = 0;
    for (auto v: graph[u])
        val = max(val,C[u][v] + calculate_rank(v));
    return _rank[u] = wAvg[u] + val;
}
```

LISTING 2.1: Method to compute Ranks for Task prioritization phase

2. *Task allocation*: We compute EST (Earliest start time) and EFT (Earliest finish time) of the task n_i on the processor u_j with the maximum frequency. Since we will allocate tasks with the maximum frequencies, this approach is termed as Non-DVFS (Dynamic voltage and frequency scaling).

$$EST(n_i, u_k) = \max\left(available[k], \max_{n_x \in pred(n_i)} (AFT(n_x) + C'_{x,i})\right) \tag{2.2}$$

For, entry node of the graph(node with no predecessor) the EST value is 0 for all the processors.EFT of the i_{th} task is given by,

$$EFT(n_i, u_k) = EST(n_i, u_k) + w_{i,k} \tag{2.3}$$

where, $AFT(n_i)$ represents the actual finish time of the i_{th} task , $pred(n_i)$ represents the immediate parent nodes of the i_{th} node in the given DAG, $available[k]$ represents the time at which the k_{th} processor is available for the task scheduling. We will keep on updating the AFT and available array values during the list scheduling for the calculation of the EST and EFT. $C'_{x,i}$ is 0 if x is equal to i and is equal to the WCRT (edge value) between task x and task i in the given DAG.

So after computing the EFT value of the i_{th} node, this node is allocated to the processor which has the least EFT value and this way of allocating task is known as insertion based scheduling strategy.The HEFT algorithm has a time complexity of $O(|N|^2 * |U|)$, where N is the number of tasks(Nodes) and U is the number of processors.

So, if we run this algorithm on the given DAG 1.2 , we get a Lowerbound of 81. The total energy consumed is $E_{total}(G) = E_s(G) + E_d(G) = 2.43 + 84.12 = 86.55$. The table showing all the AST(actual start time) and the AFT(actual finish time) values alongwith the information of the processor to which every task is assigned is shown below.

Task	Processor	$AST(n_i)$	$AFT(n_i)$	$f_{i,k}$	$E_d(n_i)$
n_1	u_3	0	8	1.00	1.92
n_2	u_1	26	40	1.00	18.48
n_3	u_3	8	27	1.00	4.56
n_4	u_2	17	25	1.00	4.40
n_5	u_3	27	37	1.00	2.40
n_6	u_2	25	41	1.00	8.80
n_7	u_3	37	48	1.00	2.64
n_8	u_1	64	70	1.00	7.92
n_9	u_1	50	64	1.00	18.48
n_{10}	u_1	70	**81**	1.00	14.52

Table 2.1: Task Assignment of the Parallel Application (DAG) using HEFT Algorithm

For this run, the power parameters of the processors are given in Table 1.2 and the WCET values are given in Table 6.4.

Chapter 3

The Deadline Slack Algorithm

With HEFT algorithm we get an initial task assignment and the lower bound of the parallel application(DAG).The demerit of HEFT algorithm is that we didn't consider the energy consumption of the task on the processor while we assign the task to a processor.
In Deadline slack Algorithm, we will assign a deadline to each task in the given DAG. The deadline slack of a parallel application is the difference between the deadline and the lower bound of the application.

$$DS(G) = D(G) - LB(G) \tag{3.1}$$

Now, using HEFT algorithm we can compute the $LB(G)$ that is the lower bound of the graph and since we know the deadline , we can compute the deadline slack of the graph. So, with this we can compute deadline of every task node as ,

$$D(n_i) = LB(n_i) + DS(G) \tag{3.2}$$

where, $D(n_i)$ is the deadline of the i_{th} task node and $LB(n_i)$ is the lowerbound of the i_{th} task node which is equal to the $AFT(n_i)$ which we got using the initial run of HEFT algorithm.

In HEFT, it assigned each task to the processor with the minimum EFT with the maximum frequency (non-DVFS),but Deadline slack assigns each task to the processor with the maximum frequency and the minimum energy value such that the deadline constraint of that task is satisfied, which is given by $AFT(n_i) \leq$D(n$_i$). Similar to HEFT, Deadline slack algorithm is a non-DVFS scheduling algorithm since it doesnt́ scale frequencies while assigning tasks to the processors but assign them with the maximum frequency and hence the execution time of the i_{th} task node on the j_{th} processor takes time $W_{i,j}$ (WCET).
Similar to HEFT, in task prioritization phase the task nodes are ranked according to the rank

values in the decreasing order. In the task allocation phase, the algorithm assigns the task nodes to the processor with the minimum dynamic energy consumption such that the deadline of the task node is satisfied. If there exist no processor in which this deadline constraint of the task node is not satisfied then the task node is assigned to a processor with the least EFT value which is similar to the HEFT task allocation step.
The Deadline slack algorithm has a time complexity of $O(|N|^2 * |U|)$ where N is the number of tasks(Nodes) and U is the number of processors, which is similar to that of HEFT.
So, if we run this algorithm on the given DAG 1.2, The total energy consumed is $E_{total}(G) = E_s(G) + E_d(G) = 3.06 + 63.74 = 66.80$. The table showing all the AST(actual start time) and the AFT(actual finish time) values alongwith the information of the processor to which every task is assigned is shown below.

Task	Processor	$AST(n_i)$	$AFT(n_i)$	$f_{i,k}$	$E_d(n_i)$
n_1	u_3	0	8	1.00	1.92
n_2	u_2	26	46	1.00	11.00
n_3	u_3	8	27	1.00	4.56
n_4	u_3	27	44	1.00	4.08
n_5	u_3	44	54	1.00	2.40
n_6	u_1	22	37	1.00	19.80
n_7	u_1	50	54	1.00	5.28
n_8	u_2	71	82	1.00	6.05
n_9	u_3	62	82	1.00	4.80
n_{10}	u_2	95	**102**	1.00	3.85

Table 3.1: Task Assignment of the Parallel Application (DAG) using Deadline Slack Algorithm

For this run, the power parameters of the processors are given in Table 1.2 and the WCET values are given in Table 6.4.
As we can see that in this example, the schedule length of the application exceeds the deadline of the graph which is a demerit of the Deadline slack algorithm and thus this needs further optimisations.

Chapter 4

The NDES Algorithm

In Deadline slack algorithm , we can get a schedule length greater than the deadline of the application graph. This happens due to the fact that the system is heterogeneous and there is communication time between tasks if they are not assigned to the same processor. For example, as we can see in the table 5.1, all the predecessor tasks of n_{10} has its deadlines satisfied but in case of n_{10} task, the schedule length surpasses the deadline which is set to 100. This happens as the EST of this task is 95 which is due to the fact that we take maximum for all predecessor(see definition of EST) and hence, $EST(n_{10}) = 82 + 13 = 95$ from the n_9 task. This 13 unit of time is consumed as both tasks are assigned to different processors and 13 is the communication time in between them and hence the AFT of the exit node is 102 which is greater than 100.

This can be handled if we set a deadline slack for every task but since there are so many possibilities we fix a deadline slack for each task and then vary this slack to obtain a task allocation which satisfies the deadline constraint of the application graph. A variable deadline slack $VDS(G)$ is defined for the application graph. We will iterate this $VDS(G)$ to obtain a task assignment which satisfies the deadline constraint and also the energy consumption is minimized.

$$D(n_i) = LB(n_i) + VDS(G) \tag{4.1}$$

and for $n_i = n_{exit}$, $D(n_i) = D(G)$. If the schedule length obtained by deadline slack algorithm is greater than deadline of the application graph then all the variable deadline slacks can be traversed from $[DS(G) - S(G), 0]$. While traversing, we can obtain an allocation which satisfies the constraints with the minimum dynamic energy consumption minimized.
If the schedule length obtained by deadline slack algorithm is less than deadline of the application graph then again we can try to optimize the result further by traversing the $VDS(G)$ from $[DS(G), X]$ where $X = D(G) - \max(LB(n_i))$ since the finish time of all the tasks cannot exceed the deadline of the application graph.

Following the NDES algorithm, we are assured that we can obtain a safe schedule length with minimum total energy consumption as possible , where all the tasks are running with maximum frequency since NDES algorithm follows the non-DVFS technique.

The NDES algorithm has a time complexity of $O(c * |N|^2 * |U|)$ where N is the number of tasks(Nodes), U is the number of processors and c is the number of iterations of the variable deadline slack which depends on the increment parameter.
So, if we run this algorithm on the given DAG 1.2, The total energy consumed is $E_{total}(G) = E_s(G) + E_d(G) = 2.82 + 44.49 = 47.31$. The table showing all the AST(actual start time) and the AFT(actual finish time) values alongwith the information of the processor to which every task is assigned is shown below.

As we can see that the static energy is reduced from the previous Deadline slack algorithm since in this case the schedule length obtained is 94 which is less than what we got with deadline slack algorithm(102). Also the dynamic energy consumption is significantly reduced from 63.74 to 44.49.

Task	Processor	$AST(n_i)$	$AFT(n_i)$	$f_{i,k}$	$E_d(n_i)$
n_1	u_3	0	8	1.00	1.92
n_2	u_3	27	45	1.00	4.32
n_3	u_3	8	27	1.00	4.56
n_4	u_2	17	25	1.00	4.40
n_5	u_2	25	38	1.00	7.15
n_6	u_3	45	54	1.00	2.16
n_7	u_1	50	54	1.00	5.28
n_8	u_2	69	80	1.00	6.05
n_9	u_3	54	74	1.00	4.80
n_{10}	u_2	87	**94.**/	1.00	3.85

Table 4.1: Task Assignment of the Parallel Application (DAG) using NDES Algorithm

For this run, the power parameters of the processors are given in Table 1.2 and the WCET values are given in Table 6.4.

Chapter 5

The GDES Algorithm

GDES or **Global DVFS-Enabled Energy efficient scheduling algorithm** solves the problem of task scheduling using the DVFS technology.NDES algorithm does not scale frequencies of the task nodes and all the tasks are scheduled with maximum frequency. Also, there exists lots of free slacks between the tasks on the processors.
GDES algorithm eliminates these slacks as much as possible by scaling down the frequencies on which the task nodes run on the processor to minimimze the energy utilization. The key point is of GDES algorithm is that the schedule length of the application is always equal to the deadline of the input graph. Some new terms will be introduced to implement DVFS-enabled scheduling algorithm and they are:

- *Earliest start time(EST):* In HEFT algorithm, we considered availability of the processor also while computing the earliest start time of the i_{th} task on the k_{th} processor but in GDES its not excluded as task can be re-scheduled to other processor if it consumes less energy and satisfies the constraints.

$$EST(n_i, u_k) = \max_{n_x \in pred(n_i)} (AFT(n_x) + C'_{x,i}) \tag{5.1}$$

  EST is 0 for the entry task node.

- *Latest finish time(LFT):* Latest finish time is computed for task for every processor and this alongwith the EST is used to find the suitable slack on which we can assign the task.

$$LFT(n_i, u_k) = \max_{n_x \in succ(n_i)} (AST(n_x) - C'_{x,i}) \tag{5.2}$$

  LFT is $D(G)$ for the exit task node.

- *Available Slacks:* After getting an initial mapping of the task distribution , there are slacks left in between the two tasks in the same processor. This can computed and the set of available slacks are used to find the best suitable slack on which the energy utilization is minimized for the given task such that all the constraints are also satisfied. To find the available slacks for a task node n_i , n_i should be removed from the task distribution.

```
vector<vector<pair<int,int>>> computeSlacks(vector<tasks> info, int id)
{
vector<vector<pair<int,int>>> use,ret;  // will store the final answer
use.resize(NUM_PROC);
for(auto t: info) // iterating through the task distribution
{
    if(t.id != id) // exclude the current task
        use[t.proc_num].push_back({t.st_time,t.en_time});
}
for(int i = 0; i < NUM_PROC; i++)
    sort(use[i].begin(),use[i].end()); // sort the task mappings intervals

for(int i = 0; i < NUM_PROC; i++)
{
    vector<pair<int,int>> temp;
    if(use[i].size() == 0) // if no task runs on the processor then available
slack = [0,D(G)]
    {
        temp.push_back({0,D_G});
        ret.push_back(temp);
        continue;
    }
    for(int j = 0; j < use[i].size(); j++)
    {
        if(j == 0) // first task on the jth processor
        {
            if(use[i][0].first != 0)
                temp.push_back({0,use[i][0].first});
        }
        if(j != 0 && use[i][j].first != D_G && use[i][j-1].second != use[i][j].
first) // middle tasks
        {
            temp.push_back({use[i][j-1].second,use[i][j].first});
        }
        if(j == use[i].size()-1) // last task on the jth processor
        {
            if(use[i][j].first != D_G && use[i][j].second !=  D_G)
                temp.push_back({use[i][j].second,D_G});
        }
    }
    ret.push_back(temp);
}
return ret;
}
```

Listing 5.1: Method to compute available slacks

- *Maximum execution time(MET)* To handle the precedence constraints, we can only assign tasks to those slacks among the available ones which satisfies the following condition.

$$MET(n_i, u_k) \geq w_{i,k} \tag{5.3}$$

where MET is defined as,

$$MET(n_i, u_k) = LFT'(n_i, u_k) - EST'(n_i, u_k) \tag{5.4}$$

such that,

$$EST'(n_i, u_k) = \max(EST(n_i, u_k), leftInterval(Slack)) \tag{5.5}$$

$$LFT'(n_i, u_k) = \min(LFT(n_i, u_k), rightInterval(Slack)) \tag{5.6}$$

So, if the condition 5.3 is not satisfied then the task cannot be assigned to the current slack. If it is satisfied we find that slack in which the energy utilization is minimized.

Now, since we have the slack on which we can assign the task such that the energy consumption is minimzed we can compute the frequency with which the task will run on the processor. We can compute the maximum time the task can take if it runs on a processor using the lowest frequency which is given and this UBET(Upper bound execution time) can be computed by,

$$UBET(n_i, u_k) = \frac{f_{k,max}}{f_{k,low}} \times w_{i,k} \tag{5.7}$$

So, the Maximum execution time can be updated as $MET(n_i, u_k) = \min(MET(n_i, u_k), UBET(n_i, u_k)$ Since with maximum frequency, the n_i task takes $w_{i,k}$ time on the u_k processor, therefore the frequency with which it runs is equals to ,

$$f_i = \frac{w_{i,k}}{MET(n_i, u_k)} \times f_{k,max} \tag{5.8}$$

with this frequency we can compute the dynamic energy consumption for the n_i task node which runs on the u_k processor using equation 1.3.

After this, we can assign the n_i task node to the u_k processor with energy E_d and the actual finish time and actual start time can be computed as,

$$AFT(n_i) = LFT'(n_i, u_k) \tag{5.9}$$

and

$$AST(n_i) = LFT'(n_i, u_k) - MET(n_i, u_k) \tag{5.10}$$

Note that GDES algorithm requires an initial task distribution which can be computed using any scheduling algorithm like HEFT or NDES. Later we will show that **GDES&NDES** algorithm gives the best results.

So, if we run this algorithm on the given DAG 1.2, The total energy consumed is $E_{total}(G) = E_s(G) + E_d(G) = 3.00 + 33.16 = 36.16$. The table showing all the AST(actual start time) and the AFT(actual finish time) values alongwith the information of the processor to which every task is assigned is shown below.

As we can see that the static energy is reduced from the previous NDES algorithm since in this case the schedule length is 100 which is equal to the deadling of the application graph. Also the dynamic energy consumption is significantly reduced from 44.49 to 33.16

Task	Processor	$AST(n_i)$	$AFT(n_i)$	$f_{i,k}$	$E_d(n_i)$
n_1	u_3	0	8	1.00	1.92
n_2	u_3	27	45	1.00	4.32
n_3	u_3	8	27	1.00	4.56
n_4	u_2	17	31	0.57	2.86
n_5	u_1	19	41	0.54	5.37
n_6	u_3	45	54	1.00	2.16
n_7	u_1	50	70	0.20	0.64
n_8	u_2	69	87	0.61	4.10
n_9	u_3	54	74	1.00	4.80
n_{10}	u_2	87	**100**	1.00	2.42

TABLE 5.1: Task Assignment of the Parallel Application (DAG) using GDES&NDES Algorithm

For this run, the power parameters of the processors are given in Table 1.2 and the WCET values are given in Table 6.4.

Chapter 6

Analysis and Conclusion

In previous chapters we studied non-DVFS scheduling algorithms like **HEFT**, **Deadline slack**, **NDES algorithm** and DVFS-enable scheduling algorithms like **GDES**. Also we saw that we can combine GDES with HEFT as well as NDES to get better results.

We will test the above mentioned algorithms on some sample test graphs like ,

- *Fast Fourier transform* task graph
- *Gaussian elimination* task graph
- *Divide and conquer* task graph

6.1 Fast Fourier transform task graph

A FFT task graph has $2 + (n + 1)2^n$ task nodes and $(n + 1)2^{n+1}$ communication edges where 2^n is the number of task nodes on the second level of the given application task graph. For example, in the given sample graph 6.1, $n = 2$ so the number of nodes is equal to 14 and the number of edges is 24.

Task(n_i)	u_1	u_2	u_3
n_1	14	16	9
n_2	13	19	18
n_3	11	13	19
n_4	13	8	17
n_5	12	13	10
n_6	13	16	9
n_7	7	15	11
n_8	5	11	4
n_9	18	12	20
n_{10}	21	7	16
n_{11}	11	5	13
n_{12}	12	4	15
n_{13}	15	2	11
n_{14}	25	4	21

TABLE 6.1: WCET(Worst case execution time) matrix for FFT application graph 6.1

From the result figure 6.2 we can conclude that,

1. Maximum energy consumption is with the HEFT algorithm and the energy consumption by HEFT does not change by the change in the deadline of task graph. This is because HEFT does not consider energy minimization while assigning tasks to the processor but only check the precedence constrains through EST and EFT.

2. GDES&NDES algorithm always produce better or similar results than the NDES algorithm. This can be explained by the fact that GDES is run on the task distribution obtained by the NDES algorithm and GDES using DVFS technology as well as the slack reduction to further minimize the energy consumption by re-distribution of the tasks. Similarly,GDES&HEFT algorithm always produce better or similar results than the HEFT algorithm.

3. As the deadline increases the GDES&NDES algorithm consumes less energy than all other algorithms and when the deadline is far away from the schedule length then both GDES&NDES and GDES&HEFT gives almost similar results [convergence point].

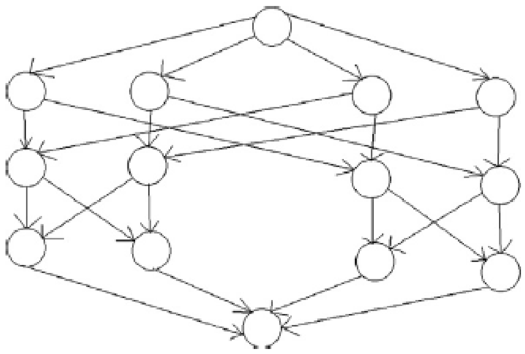

FIGURE 6.1: A sample fast Fourier transform task graph for n = 2.

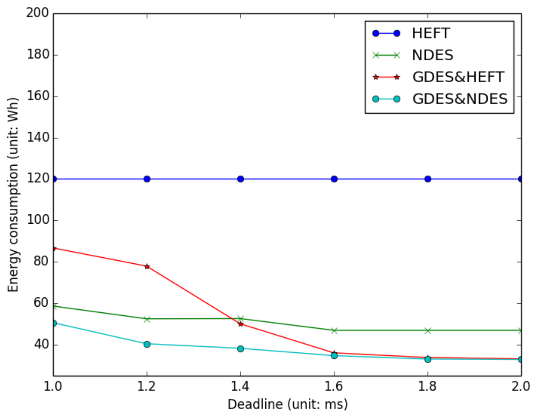

FIGURE 6.2: Results of the FFT application graph 6.1

6.2 Gaussian elimination task graph

A Gaussian elimination task graph has $(n^2 + n + 4)/2$ task nodes and $n^2 + 1$ communication edges where n is the number of task nodes on the second level of the given application task graph. For example, in the given sample graph 6.3, $n = 3$ so the number of nodes is equal to 8 and the number of edges is 10.

Task(n_i)	u_1	u_2	u_3
n_1	14	16	9
n_2	13	19	18
n_3	11	13	19
n_4	13	8	17
n_5	12	13	10
n_6	13	16	9
n_7	7	15	11
n_8	12	15	11

TABLE 6.2: WCET(Worst case execution time) matrix for Gaussian elim. task graph 6.3

From the result figure 6.4 we can conclude that,

1. Similar to FFT task graph, Maximum energy consumption is with the HEFT algorithm and the energy consumption by HEFT does not change by the change in the deadline of task graph. GDES&NDES algorithm always produce better or similar results than the NDES algorithm.As the deadline increases the GDES&NDES algorithm consumes less energy than all other algorithms

2. Other observation is that the results of GDES&NDES are alternate for gaussian elimination application but still it gives the best results among all other algorithms. When Deadline is 1.2ms, in this case the task n_7 gets assigned to the processor u_3 and not on u_1 as it does not satisfy the deadline constraint. But once we increase the deadline to 1.4ms then in this case the task n_7 gets assigned to u_1 with energy consumption lesser than previous case but now due to this greedy choice , the task n_2 in this case is assigned to u_3 with maximum frequency as there is no other choice left at that step of task allotment. So in this case n_2 consumes 4.32 energy as compared to previous case where n_2 consumed 2.39 energy.
   this provides a good example that, this greedy choice during the task allotment can result in the increase in the energy consumption of the application graph.

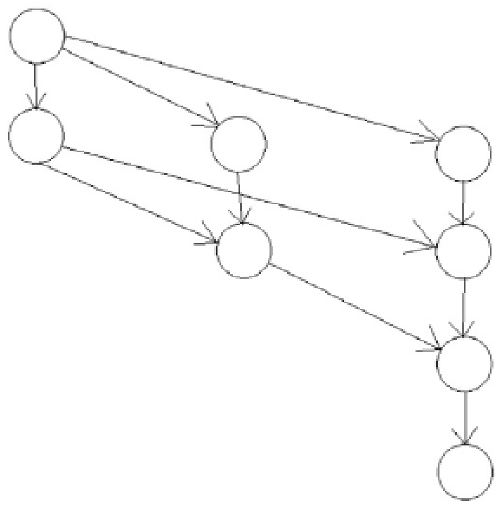
FIGURE 6.3: A sample Gaussian elimination task graph for n = 3.

6.3 Divide and conquer task graph

A sample divide and conquer task graph has $3(2^{n-1}) - 2$ task nodes and $2^{n+1} - 4$ communication edges where n is the number of task nodes on the path from the root task node(Entry node) to

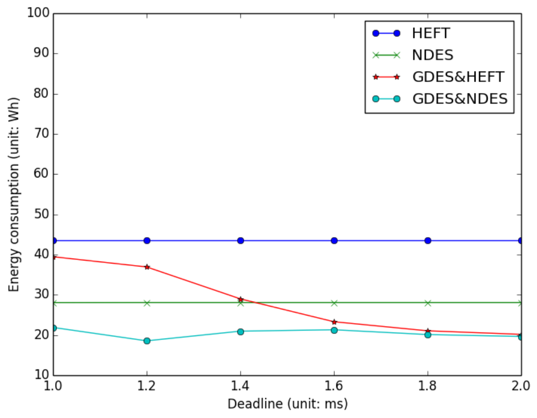

FIGURE 6.4: Results of the Gaussian elimination application graph 6.3

the middle level of the application graph. For example, in the given sample graph 6.5, $n = 3$ so the number of nodes is equal to 10 and the number of edges is 12.

Task(n_i)	u_1	u_2	u_3
n_1	14	16	9
n_2	13	19	18
n_3	11	13	19
n_4	13	8	17
n_5	12	13	10
n_6	13	16	9
n_7	7	15	11
n_8	5	11	14
n_9	18	12	20
n_{10}	21	7	16

TABLE 6.3: WCET(Worst case execution time) matrix for Divide and conquer task graph 6.5

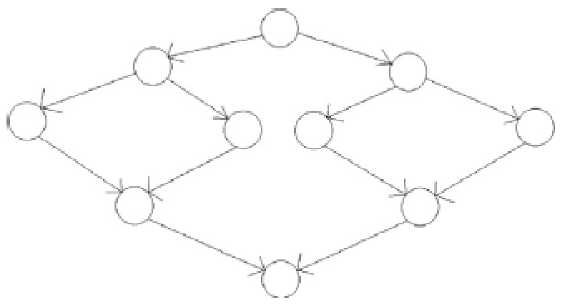

FIGURE 6.5: A sample Divide and conquer task graph for n = 3.

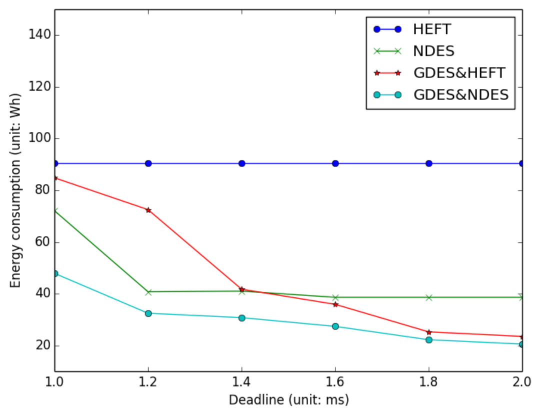

FIGURE 6.6: Results of the Divide and conquer application graph 6.5

6.4 Conclusion

So, we evaluated 2 sets of algorithms (non-DVFS and DVFS). Non-DVFS scheduling algorithms included HEFT, Deadline slack, NDES and DVFS scheduling algorithm included GDES&HEFT and GDES&NDES. The analysis was done on various test graphs like FFT, Gaussian elimination , Divide and conquer which represented a parallel application. All the analysis was done on a heterogeneous distributed embedded system.
We observed that HEFT algorithm energy consumption does not change with the change in deadline of the application graph.From the above derived results and graphs, it can be concluded that the GDES&NDES which is a DVFS enabled scheduling algorithm gives the best results i.e the energy consumption after task distribution on the heterogeneous system of processors is the least. We can also conclude that , combination of DVFS enabled and non-DVFS scheduling algorithm gives the better result than only non-DVFS scheduling algorithm. Example: GDES&HEFT always gives better results than HEFT and similarly, GDES&NDES always gives better results than NDES.

Algorithm	FFT	Gaussian	Divide and conquer
HEFT	119.99	43.50	90.50
NDES	46.92	28.08	38.61
GDES&HEFT	39.21	36.14	39.11
GDES&NDES	**35.33**	**20.83**	**27.77**

TABLE 6.4: Energy consumption (Algorithm - Task graph) for Deadline = 150

Bibliography

[1] G. Xie, G. Zeng, X. Xiao, R. Li and K. Li, "Energy-Efficient Scheduling Algorithms for Real-Time Parallel Applications on Heterogeneous Distributed Embedded Systems," in IEEE Transactions on Parallel and Distributed Systems, vol. 28, no. 12, pp. 3426-3442, Dec. 1 2017. doi: 10.1109/TPDS.2017.2730876

[2] Pramod Kumar Mishra, Abhishek Mishra, Kamal Sheel Mishra, Anil Kumar Tripathi,Benchmarking the clustering algorithms for multiprocessor environments using dynamic priority of modules, Applied Mathematical Modelling, Volume 36, Issue 12,2012, Pages 6243-6263, ISSN 0307-904X, https://doi.org/10.1016/j.apm.2012.02.011. (http://www.sciencedirect.com/science/article/pii/S0307904X12000935)

[3] Q. Huang, S. Su, J. Li, P. Xu, K. Shuang and X. Huang, "Enhanced Energy-Efficient Scheduling for Parallel Applications in Cloud," 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012), Ottawa, ON, 2012, pp. 781-786. doi: 10.1109/CCGrid.2012.49

[4] $http://www.mi.sanu.ac.rs/~tanjad/sched_results.htm$