DPS: A Dynamic Procrastination Scheduler for Multi-core/Multi-processor Hard Real Time Systems

Shubhangi K. Gawali, Biju K. Raveendran
Department of Computer Science, BITS PILANI K. K. BIRLA GOA CAMPUS
Goa, India
{shubhangi,biju}@goa.bits-pilani.ac.in

***Abstract*— Energy consumption plays an important role in designing embedded devices. In recent years, leakage energy gained significant importance in overall energy consumption. This paper addresses leakage energy at operating system level by optimizing scheduler level energy consumption. This is achieved by (i) modifying the first fit allocation algorithm (MFF) to increase shutdown duration of some processors by allocating low frequency tasks to that processors and (ii) maximizing the shutdown duration using dynamic procrastination algorithm (DPS). The procrastination is achieved by postponing the execution of upcoming jobs whenever possible by meeting all timing constraints. This helps in improving shutdown duration along with reducing decision points and static energy consumption. Shutdown decision is based on pre-computed shutdown threshold which is architecture and task set dependent. The experimental evaluation of the proposed algorithm with synthetically generated benchmark program suites shows 88.66%, 14.75% and 1% of saving in total energy consumption over no procrastination (NOPRO) with MFF task allocation, static procrastination (STATICPRO) with MFF task allocation and DPS with FF task allocation respectively.**

***Keywords*— procrastination, Multi-core real-time scheduling, Multi-processor real-time scheduling.**

I. Introduction

Advancement in CMOS technology offers increased transistor count per unit area [1]. It provides more capabilities at the cost of run-time complexity and energy consumption. In embedded systems, energy efficiency is one of the most widely addressed research areas in recent years as it plays a vital role in prolonging battery life. Some of these embedded systems are working with hard real-time tasks where meeting all task deadlines with minimum energy consumption is a major design consideration [2]. Optimization of energy consumption in uniprocessor, multicore or multiprocessor hard real-time systems can be addressed at various levels of design hierarchies such as technology level, circuit level, architecture level, operating system level, compiler level and application level [2, 3]. Operating system and architecture level energy consumption can be addressed at process execution, scheduling, memory, file systems, synchronization, instruction set architecture, interconnection network, cache memory, voltage and frequency scaling etc [4].

This paper addresses energy consumption because of process execution and scheduling as these consume majority of the system level energy consumption. Energy efficiency can be achieved by optimizing architecture independent parameters like preemptions, function calls, system calls, decision making etc and architecture dependent parameters like supply voltage and clock frequency [5-7]. Supply voltage and clock frequency can be optimized only if the architecture supports slowdown and shutdown techniques [7-10]. The slowdown techniques like dynamic voltage and frequency scaling (DVFS) helps in reducing dynamic energy consumption which is due to switching activities [6, 7, 11-13]. The DVFS technique reduces supply voltage and clock frequency to reduce dynamic energy. These techniques do not address static energy consumption due to leakage current [8, 14]. The static energy is equally an important component like dynamic energy as channel length reduces with advancement in technology which resulted in increasing leakage current of the CMOS transistor [8, 14, 15]. The leakage current is 0.01μA/μm for the 130*nm* and is 3μA/μm for 45*nm* technology [8]. Shutdown technique tries to shut the processor down if the processor is underutilized which is the only way to avoid leakage current [9, 16]. There exists a possibility of increasing the impact of platform independent parameters because of platform dependent parameters [7, 9]. For instance, applying DVFS technique in a schedule will result in increased execution time which may increase the time spent for preemption and scheduler decisions. The shutdown has hidden overheads of switch off and on time and energy. The shutdown technique saves energy only if the duration of shutdown is beyond a pre-computed threshold time [8]. This can be achieved by finding the latest start time of tasks without missing deadlines. Procrastination [8] is one of the most widely used techniques to achieve this. In a periodic task set, if the largest procrastination possible is lesser than the shutdown threshold, the shutdown technique is not recommended as the time and energy required for shutting down and restarting the processor will be more than the time and energy saved because of shutdown. This paper addresses these issues in a Multi-core (MC) / Multi-processor (MP) hard real-time environment. This work proposes an efficient task partitioning scheme using first fit bin packing approximation algorithm to improve the possibility of shutdown while allocation of tasks to cores/processors. This

Authorized licensed use limited to: Birla Institute of Technology and Science. Downloaded on September 25,2020 at 12:58:45 UTC from IEEE Xplore. Restrictions apply.

work also proposes an energy efficient scheduler for MC/MP hard real-time non-DVFS systems which supports processor shutdown.

The rest of the paper is organized as follows: Section II discusses the recent related work in the area of task allocation and shutdown techniques in scheduling for MC/MP systems. In section III, the proposed work is explained. Section IV discusses the experimental setup. The experimental results in comparison with first fit (FF) allocation and static procrastination are presented in Section V. Section VI concludes the paper with future directions.

II. Related work

A. Task allocation algorithms in MC/MP systems

The main objective of the task allocation algorithms in MC/MP systems is to find the optimal number of cores/processors required for the given task set. This can be achieved using online, semi-online and offline algorithms [17]. In online algorithms, the task is assigned to the processor as soon as it arrives to the system by considering all the tasks which already arrived [17]. This is achieved by carrying out a schedulability test for the newly arrived task with all the allocated tasks. The semi-online algorithms follow online algorithms by keeping an upper bound on rearrangements of allocated tasks [17]. The offline algorithms are static algorithms where the number of tasks in the task set and the order in which they are assigned to the processor is fixed. Bin packing approximation is one of the most efficient offline task allocation used in MC/MP systems [14]. In this technique, a set of tasks, a set of processors and the utilization of each processor are given as input parameters. It divides the task set into sub task sets based on processor utilization and allocates it to the processors for scheduling. The optimal Bin packing is one of the classic NP-complete problems. The widely used polynomial-time Bin packing approximation algorithms are First-Fit (FF), Best-Fit (BF), Best-k-Fit (BkF), Next-Fit (NF) and Worst-Fit (WF) [17-19]. FF allocates a new task to a non-empty processor with the lowest index, such that the utilization of the new task along with the utilization of the tasks already allocated to that processor, do not exceed the capacity of that processor [17, 18]. BF allocates a new task to the non-empty processor with smallest capacity available, in which this task can be allocated with a tie braking strategy as index number [9, 20]. WF is similar to BF, with the difference that it allocates tasks to the processors with the largest capacity available, in which task can be feasibly allocated [9, 20]. FF, BF and WF algorithms allocate the new task to a new processor only if it is not fitting in any of the processors whose allocation already started. BkF allocates like BF but considers only the last k open processors for allocation [21]. It allocates the new task to a new processor only if it is not fitting in any of the last k processors. In NF, if the new task can fit in the last allocated processor, it is allocated to the processor. Otherwise a new processor is chosen and the task is allocated to that processor. Some of the online versions of Bin packing algorithms are FF Decreasing (FFD), Refined FFD (RFFD), Modified FFD (MFFD), BF Decreasing (BFD) and Group-X Fit grouped (GXFG) [17]. The performance metric of these algorithms is the ratio of number of processors in approximation algorithms over optimal method. The performance ratio given by FF is the best which is 1.2 [17]. The task assignment can be improved to optimize not only the number of processors but also the energy consumption. Algorithm proposed by Tarek and Hakan allocates tasks to the processor and computes CPU speed assignments for minimizing the total energy consumption [22]. This paper proposes modification to FF algorithm for task allocation – MFFBP in which the tasks are arranged in non-decreasing order of their periods instead of utilization. This helps in getting longer shutdown duration to the processors having higher period jobs. The detailed explanation of MFFBP is given in next section.

B. Shutdown techniques while scheduling in MC/MP systems

Optimization of energy during task scheduling in MC/MP systems can be addressed using processor slowdown and/or processor shutdown techniques. The processor slowdown mechanism aims at minimizing the dynamic energy consumption where as processor shutdown aims at minimizing the static energy consumption. Processor slowdown can be implemented using platform dependent techniques like voltage and frequency scaling [6, 7]. Though DVFS saves energy, it never takes care of static energy. To reduce static energy, various shutdown techniques are proposed when cores/processors are not in use [7-9]. Processor consumes energy both in active and idle state. In [15], it is mentioned that the power dissipation when the processor is in the idle state can be in the order of 10^3 of that when the processor is shutdown. Processor shutdown can be implemented using Procrastination techniques [8, 14-16, 23-26]. Procrastination is one of the most efficient techniques to optimize the energy consumption during idle period by delaying the task execution and increasing the shutdown duration while meeting all timing requirements. Various offline and online procrastination solutions have been proposed for energy savings in real-time under different application/device settings [7]. Earlier work on procrastination is based on pre-computed (static) task procrastination intervals on considering the worst case execution time (*wcet*). In this method, the latest start time for every job is pre-computed with all timing constraints [8]. Most of the jobs completes before *wcet* thereby leaves some slack before its deadline. With this method, the pre-computed idle period for the processor is not the maximum procrastinated interval and thus underestimates the procrastinated shutdown duration. Online (Dynamic) procrastination algorithms use this slack to increase shutdown duration of the processor [14, 15, 23]. In [14] Krishna et. al. explained how idle duration can be extended by delaying the active period by an interval given by the sum of pseudo execution time of all tasks that are preempted

Authorized licensed use limited to: Birla Institute of Technology and Science. Downloaded on September 25,2020 at 12:58:45 UTC from IEEE Xplore. Restrictions apply.

plus the delay of the currently executing earliest deadline task before the active period started. In [15, 24], Niu and Quan extended the idle interval length by computing the latest start time of the job set without missing the deadline of any future jobs. Latest start time for every higher priority jobs is computed and the earliest of all these is considered to be the start of active job set. A hard real time MC/MP scheduler can be energy efficient only when there exist a long term planning while task allocation and by employing energy efficient techniques like slowdown and shutdown while scheduling. This work aims at MC/MP hard real-time systems supporting shutdown with single operational voltage and frequency. This work combines the efficient Bin packing approximation technique for task allocation with Dynamic Procrastination techniques for scheduling in MC/MP hard real-time systems.

III. Proposed work

This paper proposes Dynamic Procrastination Scheduler (DPS) - an energy efficient scheduler for MC/MP hard real-time systems. DPS uses Modified First Fit Bin Packing (MFFBP) approximation for task allocation and Earliest Deadline First (EDF) with procrastination for job scheduling. Unlike the conventional FF Bin Packing (FFBP) approximation algorithm where the tasks are arranged in non-increasing order of utilization, MFFBP arranges tasks in non-decreasing order of their periods to increase the shutdown duration of the processor. In MFFBP, to increase shutdown duration, the tasks with higher periods are allocated to some set of processors. The tasks with lower periods are allocated to a different set of processors. The processors with lower period tasks are not considered for shutdown as it can never create idle time beyond shutdown threshold. This allows the processors having higher periods to have longer shutdown duration. This is achieved with a time complexity of O(M) where M is total number of tasks.
The algorithm for MFFBP is as follows:

```
Input: Task set T with M tasks and N processors.
Output: Minimum number of processors required (K) and
set of sub task sets T={S_1, S_2, …S_k} allocated per processor.
ALGORITHM (MFFBP)
Begin
Step1: Sort M tasks in non-decreasing order of their periods.
Step2: Initialize K to 1 //start allocation by taking processor
count as 1.
For each task i in M
Begin
        Flag = 0
        For each processor j in K
        Begin
           If j.remUtil >= i.Util
                Add task i to processor j's list
                j.remUtil  = j.remUtil  - i.Util
                Flag = 1 and break
           Endif
        Endloop
        If (Flag = = 0)  K = K + 1
           If K>N error "Insufficient Processor" and Exit
            Add task i to processor K's list
            K.remUtil  = K.remUtil  - i.Util
        Endif
Endloop
Step3: Output K and list of tasks in each processor
End MFFBP
```

Consider a task set T consisting of seven periodic hard real-time tasks T_0 to T_6 represented as (period, wcet) $T_0(40,9.4)$, $T_1(50,20)$, $T_2(60,15)$, $T_3(80,19)$, $T_4(100,20)$ $T_5(120,20)$ and $T_6(140,25)$. In both FFBP and MFFBP, the number of processors required to execute all the tasks in the task set is 2. According to FFBP, the tasks are ordered as FFT = {T_1, T_2, T_3, T_0, T_4, T_6 and T_5} where as in MFFBP, the tasks are ordered as MFFT = {T_0, T_1, T_2, T_3, T_4, T_5 and T_6}. In FFBP, the input task set is divided into two sub task sets FS_1 and FS_2 with tasks T_1, T_2 and T_3 in FS_1 and tasks T_0, T_4, T_5 and T_6 in FS_2. In MFFBP, the input task set is divided into two sub task sets MS_1 and MS_2 with tasks T_0, T_1 and T_2 in MS_1 and tasks T_3, T_4, T_5 and T_6 in MS_2. The task allocation in MFFBP guarantees that the higher period tasks are allocated to the last processor, i.e., MS_2 is allocated to processor 1. To have uniformity in analysis, the utilization of FS_2 is made almost same as MS_2 which is approximately 78%. While scheduling with DPS, total shutdown duration offered by task set FFT is 1722.02 units and task set MFFT offered 1803.2 units. The higher value of shutdown duration shows that MFFBP offers longer shutdown, thus better static energy saving in comparison with FFBP. MFFBP outputs the number of processors required for executing the task set and the sub task set allocated to each processor. Each sub task set is given as the input to the processor for scheduling. In DPS, the procrastination decision is made only when the ready queue is empty. This helps in reducing the computational overheads. The DPS scheduler offers the maximum shutdown duration possible by taking care of all the job deadlines. The shutdown threshold helps in deciding whether to keep the processor in shutdown state or idle state. At any point in time, the optimal procrastination time can be computed by considering all the future jobs from the current time till hyperperiod. The computation time complexity of this is exponential which make it impractical. In DPS, the shutdown duration is calculated by considering the jobs arriving between current time t and D_{next} where D_{next} is the deadline of the lowest priority job arriving before the nearest deadline. This is done in O(K) time where K is the number of jobs arriving between t and D_{next}. All the arrivals are considered with their *wcet*. For the jobs whose deadlines are after $D_{next,}$, only a portion of their executions before D_{next} are considered. The DPS scheduler assumes that all the N hard real-time tasks are in-phase, independent, preemptive and periodic in nature whose deadlines are same as periods. Each task T_i of a task set T is represented by {Φ_i, P_i, C_i, D_i} [27]. Each kth job of task T_i is represented by $J_{i,k}$ {a_{ik}, c_{ik}, d_{ik}} where $d_{ik} = a_{ik} + D_i$. The utilization of task Ti is calculated as $U_i = C_i/P_i$ which is the fraction of processor

Authorized licensed use limited to: Birla Institute of Technology and Science. Downloaded on September 25,2020 at 12:58:45 UTC from IEEE Xplore. Restrictions apply.

time used by T_i. The total processor utilization because of the task set T is calculated as $Up = \sum_{i=1 to N} Ui$. The hyperperiod over which the task set is scheduled is computed as least common multiple of all P_i. NSI represents the next scheduler invocation time.

The DPS algorithm is given below:

Input: Sub task set S_i allocated to the processor P_i and time t at which ready queue is empty.

Output: NSI and shutdown decision

```
ALGORITHM (DPS)
Begin
Step1: Find job J with earliest deadline = $D_{next1}$ after t.
Step2: If ($D_{next1}$ – t – J.c < shutdown threshold)
           NSI = next job arrival time and Return FALSE.
Step3: Find $D_{next2}$ as deadline of lowest priority job arriving
       before $D_{next1}$.
Step4: Find job list (L) releasing between t and $D_{next2}$ and
       sort them in non-increasing order of their deadlines.
Step5: Initialize next scheduler invocation (NSI) to $D_{next2}$.
Step6: For each job i in L job list
Begin
    If ( i.d > $D_{next2}$ )
         NSI = NSI - (( $D_{next2}$ - i.r ) * ( i.C / i.p ) * $U_p$ )
    Else NSI = NSI - i.C
    If ( i != K AND NSI > (i+1).d ) NSI = (i+1).d
End Loop
Step7: If ( NSI – t ) < shutdown threshold
         NSI = next job arrival time and Return FALSE
Step 8: Return TRUE
End DPS
```

The DPS algorithm decides whether to shutdown or idle the processor based on procrastinated time and shutdown threshold. If the procrastinated shutdown duration computed by DPS is more than the shutdown threshold, the processor shuts down for procrastinated shutdown duration after backing up the relevant data. The system follows EDF schedule by keeping the processor idle until the next job arrival otherwise. Consider jobs of tasks in MS_2 with actual execution time (*aet*) same as *wcet*. Fig. 1 shows the resultant schedule without procrastination. At time 187, the processor becomes idle until next job arrival i.e., till time 200.

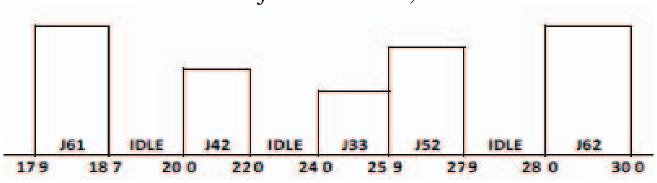

Fig. 1. Gantt chart without procrastination

In DPS, at time 187, the processor becomes idle and the scheduler finds the procrastination time. This is done by finding the job with earliest deadline i.e., job J_{42} (job 2 of task T_4) with deadline as 300. As (300 - 187 - 20) is greater than the shutdown threshold which is considered as 40 here, the algorithm decides to compute maximum procrastination time. The D_{next2} is computed as 420 which is the deadline of job J_{62}. List of jobs available for decision making is {J_{44}, J_{53}, J_{35}, J_{62}, J_{43}, J_{34}, J_{52}, J_{33} and J_{42}}. By iterating through all these jobs, DPS finds the latest time to start executing next job as 278.25. As the next start time is much higher than the shutdown threshold, the scheduler decides to shut the processor down till 278.25. The resultant schedule between 179 and 317.25 is shown in Fig. 2. The same task set while executing with MFFDPS saves 19.8%, 15.9% and 3.76% of static energy in comparison with no procrastination, static procrastination and FFDPS respectively. The difference widens with increase in ratio of wcet over aet. The detailed energy calculation is explained in the next section.

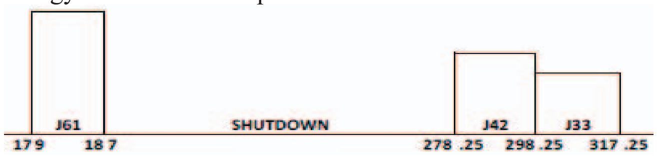

Fig. 2. Gantt chart with DPS

IV. EXPERIMENTAL SETUP

A new framework named Multi-processor Dynamic Procrastination Schedulers (MPDPS) is designed and implemented to measure performance of the proposed algorithm in comparison with seminal algorithms. The framework evaluates scheduling algorithms based on shutdown duration, static, dynamic and total energy consumptions. Similar to [8, 14, 15, 23], the experimentation is conducted by using several randomly generated task sets each containing 20 tasks. Such randomly generated tasks are used as the common validation methodology in real-time scheduling. Based on [8], tasks were assigned a random period between 250ms and 8000ms and wcet between 35% and 80% of the period such that the total utilization of the task set varies from 210% to 270% in a set of 3 processors. The experiments obtain the *aet* using Gaussian distribution with mean, $\mu=(wcet+aet)/2$ and *standard deviation* = $\int(wcet-aet)/no\ of\ tasks$. The aet of the task is varied between 10% and 100% of its wcet in steps of 10%. All tasks are simulated to execute up to hyper period i.e; least common multiple of periods of all tasks in the task set since the pattern of the schedule repeats after hyper period. MPDPS uses FF and MFF algorithms for task allocation. These task sets are then scheduled using MFF with Static procrastination (MFFSTATICPRO), MFF without DPS (MFFNOPRO), FF with DPS (FFDPS), and MFF with DPS (MFFDPS) algorithms.

The energy components are considered for 70*nm* technology for Transmeta Crusoe processor reported by [8]. The total energy consumption (E_{tot}) during execution is measured by considering energy components like Static energy (E_{stat}), Dynamic energy (E_{dyn}), Processor shutdown and wakeup energy (E_{PSD}), Scheduler decision making energy (E_{sched}) and Idle state energy (E_{idle}) while executing with various scheduling algorithms. E_{stat} is the energy due to leakage current and reverse bias junction current. It is computed as the product of active duration and static energy per unit. The static energy is assumed to be 22*nJ* per cycle. E_{dyn} is the energy due to switching activity in a circuit. It is computed

Authorized licensed use limited to: Birla Institute of Technology and Science. Downloaded on September 25,2020 at 12:58:45 UTC from IEEE Xplore. Restrictions apply.

as V^2fC where V is Supply voltage, f is clock frequency and C is capacitance. According to 70*nm* technology maximum frequency at 1volt is 3.1GHz with 0.43*nF* of capacitance. Since the experiments are performed on task sets at maximum speed, the dynamic energy is computed to be 44*nJ* per cycle. E_{PSD} is the energy due to flushing of data cache during shutdown and memory accesses during wakeup. It is computed as the product of number of shutdown decisions and shutdown overhead. The overhead of shutdown is estimated by considering the on chip cache. The cache size of the processor is assumed to be 32KB I-cache and 32KB D-cache. It is assumed that 20% lines of data cache are dirty before shutdown which results in 6554 memory writes. By considering 13*nJ* of energy per memory write, the total energy for data cache flush is 85μJ. The energy and latency of saving the registers is assumed negligible. When a task resumes its execution, the locality of reference changes which causes cache misses. This additional cache misses is assumed to be 10% of the cache size in both I-cache and D-cache. This causes the total overhead of 6554 cache misses on wakeup. By considering 15*nJ* of energy per memory access, total energy required for reading from memory and writing into cache is 98μJ. The energy required for updating TLBs and BTBs is assumed to be negligible. The energy required for charging the circuit logic is assumed to be 300μJ. Thus the total energy required to switch the processor between active and shutdown state is 85 + 98 + 300 = 483μJ per cycle. E_{sched} is the energy due to decision making at every scheduler invocation. It involves cost of computing procrastinated idle duration whenever the processor is idle or cost of selecting the highest priority job from ready queue and allocating processor to it whenever some job is ready. It also involves context switching overhead caused due to task completion or preemption and cache impact overhead caused due to context switching. E_{sched} is computed as the product of total number of times scheduler is invoked and decision making energy for scheduling or procrastination. The scheduling decision energy is assumed as 60μJ for procrastination decisions and 40μJ per cycle otherwise. E_{Idle} is the energy consumed during idle period. It is computed as the product of idle duration and energy during idle period per unit. According to [8], the idle state energy is assumed to be 0.49μJ per cycle and 2ms of idle interval threshold. The total energy consumption $E_{tot} = E_{stat} + E_{dyn} + E_{PSD} + E_{sched} + E_{Idle}$ (1)

V. Experimental Result

Fig. 3 shows the static energy consumption per unit for different utilizations. Irrespective of the utilizations, MFFNOPRO consumes constant static energy as the processor is ON even when the system is idle for longer duration. MFFDPS consumes the least static energy followed by FFDPS and MFFSTATICPRO. This is because the static energy is inversely proportional to the shutdown duration. It is also observed that the static energy consumption increases with increase in utilization because of the shorter shutdown period. On an average MFFDPS reduces the static energy by 199.59%, 33.42% and 1.99% over MFFNOPRO, MFFSTATICPRO and FFDPS algorithms respectively.

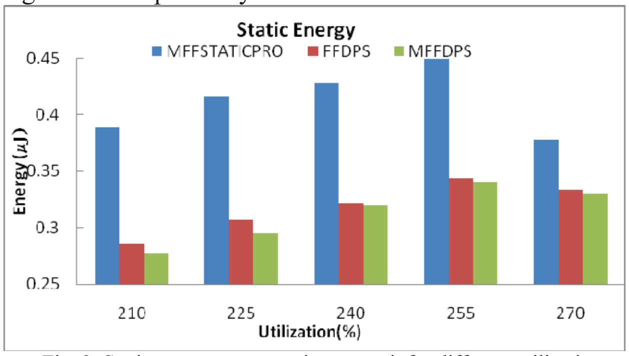

Fig. 3. Static energy consumption per unit for different utilizations

As the processor executes at maximum voltage and frequency during both active and idle period i.e., throughout the schedule MFFNOPRO consumes constant dynamic energy irrespective of the utilizations. The dynamic energy consumption follows the same trend as static energy consumption as the processor can operate only with maximum voltage and frequency i.e., no DVFS support. MFFDPS outperforms other algorithms because of this.

Irrespective of the utilizations, MFFNOPRO saves nothing as it never shuts down. On an average MFFDPS increase shutdown duration by 16% and 1% over MFFSTATICPRO and FFDPS algorithms respectively. MFFDPS outperforms MFFSTATICPRO because of the difference in wcet and aet. In procrastination approach some of the completion time decision points, when job queue is empty, becomes the procrastination decision points and consumes more energy in decision making compared to no procrastination approach. The number of procrastination decisions in MFFDPS is much lesser than MFFSTATICPRO and FFDPS. On an average MFFDPS has 23.05% and 15.84% less procrastination decisions compared to MFFSTATICPRO and FFDPS algorithms respectively. This is because the shutdown duration at each decision point in MFFDPS is high which is due to the specific arrangement of tasks by MFFBP during allocation.

The chance of converting the inactive period into shutdown period is more in DPS because of their dynamic nature. The high wcet to aet ratio increases this chance dynamically. Thus FFDPS and MFFDPS algorithms provide more chances of shutting down the processor which causes shutdown overhead of 65.9% and 25.78% respectively over MFFSTATICPRO. The shutdown overhead is high for FFDPS because of more number of shorter shutdowns in comparison with MFFDPS. The duration of shutdown is longer in MFFDPS because of the specific arrangement of tasks by MFFBP during allocation.

Fig. 4 shows the total energy consumption per unit for different utilizations with reference to equation 1. Irrespective of the utilizations MFFNOPRO's total energy consumption is very high because of its static, dynamic,

Authorized licensed use limited to: Birla Institute of Technology and Science. Downloaded on September 25,2020 at 12:58:45 UTC from IEEE Xplore. Restrictions apply.

scheduling and cache impact energy components. The total energy consumption follows the same trend as static and dynamic energy consumptions. On an average MFFDPS reduces total energy by 88.66%, 14.75% and 1% over MFFNOPRO, MFFSTATICPRO and FFDPS algorithms respectively. With increase in utilization, the MFFNOPRO offers almost constant energy consumption whereas all other algorithms increase their energy consumption with utilization. This is mainly because of the static and dynamic energy components and less chance for shutting down in all procrastination based algorithms. Irrespective of utilizations, MFFDPS offers the least energy consumption followed by FFDPS.

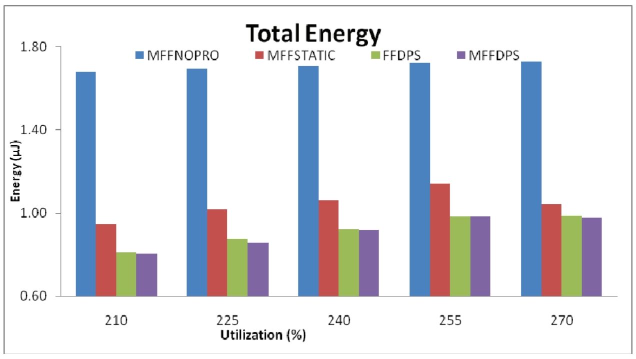

Fig. 4. Total energy consumption per unit for different utilizations

VI. CONCLUSIONS AND FUTURE SCOPE

Experimental results show that MFFDPS allows 16% and 1% more shutdown duration over MFFSTATICPRO and FFDPS algorithms respectively. The MFFDPS achieved 199.59%, 33.42%, 1.99% of static energy saving and 88.66%, 14.75% and 1% of total energy saving over MFFNOPRO, MFFSTATICPRO and FFDPS algorithms respectively. MFFDPS reduces the number of procrastination decision points by 23.05% and 15.84% over MFFSTATICPRO and FFDPS respectively. FFDPS and MFFDPS algorithms cause shutdown overhead of 65.9% and 25.78% respectively over MFFSTATICPRO. It is observed that MFFBP task allocation with DPS task scheduling in MC/MP systems offers additional 1% energy saving over FFBP. This work can be extended further by considering processors with varying voltage and frequency support to apply DVFS techniques for better dynamic energy reduction during inactive period.

VII. REFERENCES

[1] Moore's law, weblink: https://www.kth.se/social/upload/507d1d3af276540519000002/Moore's%20law.pdf cited on 26/07/15.
[2] O. Unsal and I. Koren, "System-Level Power-Aware Design Techniques in Real-Time Systems," IEEE journal, vol. 91, no. 7, pp. 1055-1069, 2003.
[3] N. Jha, "Low Power System Scheduling and Synthesis," IEEE/ACM international conference on computer aided design, pp. 259 – 263, 2001.
[4] L. Luo, "Designing Energy and User Efficient Interactions with Mobile Systems," CMU HP labs Research Thesis chapter no. 7, 2008.
[5] B. Raveendran, "Energy Efficient Techniques for Multi-Tasking Embedded Systems Cache Design and Task Scheduling Algorithms." Ph.D. Thesis, Birla Institute of Technology and Science Pilani, India, 2009.
[6] P. Pillai and K. Shin, "Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems," In 18th ACM symposium on Operating systems principles, 2001.
[7] V. Devadas and H. Aydin, "On the Interplay of Voltage/Frequency Scaling and Device Power Management for Frame-Based Real-Time Embedded Applications," In IEEE Transactions on Computers, vol. 61, no. 1, 2012.
[8] R. Jejurikar, C. Pereira and R. Gupta, "Leakage Aware Dynamic Voltage Scaling for Real Time Embedded Systems," In IEEE Design Automation Conference, pp. 275–280, 2004.
[9] G. Chen, K. Huang and A. Knoll, "Energy Optimization for Real-Time Multiprocessor System-on-Chip with Optimal DVFS and DPM Combination," In ACM Transactions on Embedded Computing Systems, vol. 13, 2014.
[10] N. Jha, "Low Power System Scheduling and Synthesis," In IEEE/ACM International Conference on Computer Aided Design, pp. 259-263, 2001.
[11] H. Aydin, R. Melhem, D. Mosse and Pedro, "Power-Aware Scheduling for Periodic Real-Time Tasks," In IEEE Transactions on Computers, vol. 53, pp. 584-600, 2004.
[12] C. Yang, J. Chen and T. To, "An Approximation Algorithm for Energy-Efficient Scheduling on A Chip Multiprocessor," In IEEE Design, Automation and Test in Europe, pp. 468-473, 2005.
[13] M. Gong, Y. Seong and C. Lee, "On-Line Dynamic Voltage Scaling on Processor with Discrete Frequency and Voltage Levels," IEEE International conference on convergence information technology, pp. 1824-1831, 2007.
[14] Y. Lee, K. Reddy and C. M. Krishna, "Scheduling Techniques for Reducing Leakage Power in Hard Real-Time Systems," In 15th Euromicro Conference on Real-Time Systems, pp. 105–112, 2003.
[15] L. Niu and G. Quan, "Reducing Both Dynamic and Leakage Energy Consumption for hard real time systems," In ACM International conference on Compilers, architecture and synthesis for embedded systems, pp. 140-148, 2004.
[16] R. Jejurikar and R. Gupta, "Dynamic Slack Reclamation with Procrastination Scheduling in Real-Time Embedded Systems," In IEEE Design Automation Conference, pages 111-116, 2005.
[17] E.G. Coffman, M.R. Garey and D.S. Jhonson, "Approximation algorithms for bin packing: a survey," Book: Approximation algorithms for NP-hard problems, chapter no. 2.
[18] O. Zapata and P. Alvarez, "EDF and RM Multiprocessor Scheduling Algorithms: Survey and Performance Evaluation," http://delta.cs.cinvestav.mx/ pmejiamultitechreport.pdf, Oct 2005.
[19] R. Davis and A. Burns, "A Survey of Hard Real-Time Scheduling for Multiprocessor Systems," In ACM journal on computing surveys vol. 43, 2011.
[20] M. Bambagini, J. Lelli, G. Buttazzo and G. Lipari, "On the Energy-Aware Partitioning of Real-Time Tasks on Homogeneous Multi-Processor Systems," In 4th IEEE International Conference on Energy Aware Computing, pp. 69-74, 2013.
[21] W. Mao, "Best k fit packing," Springer Computing, pp. 265-270, 1993.
[22] T. AlEnawy and H. Aydin, "Energy-aware task allocation for rate monotonic scheduling," In 11th IEEE Real time and Embedded Technology and Applications Symposium, pp. 213-223, 2005.
[23] V. Legout, M. Jan and L. Pautet, "A Scheduling Algorithm to Reduce the Static Energy Consumption of Multiprocessor Real-Time Systems," In 21st ACM International conference on Real-Time Networks and Systems, pp. 99-108, 2013.
[24] L. Niu, "Energy Efficient Scheduling for Real-Time Embedded Systems with QoS Guarantee", In Sixteenth IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, pp 163-172, 2010.
[25] D. Meisner, B. T. Gold and T.F. Weinsch, "PowerNap: eliminating server idle power", In International conference on architechtural support for programming languages and operating systems", 2009.
[26] S. Pagani and J. Chen, "Energy Efficiency analysis for the single frequency approximation (SFA) scheme", In IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, 2013.
[27] J. Liu, "Real-Time Systems, 5th edition" Book published by Pearson education, 2004.

Authorized licensed use limited to: Birla Institute of Technology and Science. Downloaded on September 25,2020 at 12:58:45 UTC from IEEE Xplore. Restrictions apply.