Reliability aware Dynamic Voltage and Frequency Scaling for Improved Microprocessor Lifetime

Naga Pavan Kumar Gorti
Iowa State University
Ames, Iowa, USA
gnpavan@iastate.edu

Arun K. Somani
Iowa State University
Ames, Iowa, USA
arun@iastate.edu

ABSTRACT

Dynamic voltage and frequency scaling (DVFS) is heavily used for power management in real-time environments. Although the schemes leveraging DVFS provide significant power reduction, adverse effects on chip reliability are possible. Alternate increase and decrease in operating voltage and frequency leads to thermal cycling. Increasing transistor packing density leads to a larger range of possible operating temperatures, exacerbating the thermal cycling problem. Also, the chip reliability quantification process does not include and represent the effects of small scale thermal cycles. A good number of in-field chip failures are attributed to the consequences of these. Thus, it is imperative to include their effects into the processor voltage and frequency selection process. Our work develops an integrated processor thermal and performance management technique centered on novel polynomial time scheduling algorithms that lead to lowering of thermal cycles in soft real time environments. Our technique leverages application awareness and runtime monitoring for improving chip lifetime, while achieving considerable energy savings. We show that a significant reduction in thermal cycles and peaks is possible, leading to longer chip life expectations.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*; D.4.1 [**Operating Systems**]: Process Management—*Scheduling*; D.4.8 [**Operating Systems**]: Performance—*Measurements, Modeling and prediction*

General Terms

Algorithms, Reliability, Performance, Temperature

Keywords

DVFS, Thermal cycling, Chip Mean Time To Failure, Soft real-time systems

1. INTRODUCTION

Managing the power consumption is a primary challenge in microprocessor design. Higher power consumption leads to higher operational costs, reduced chip lifetime, and the need to design expensive heat sinks. The power consumed by the chip is roughly given by the product of the power consumed per transistor and their number. Technology scaling reduces the power consumed per transistor, along with its size. On the other hand, the increased demand for computation drives the chip designers to pack more transistors on chip, leading to overall increase in power. To avoid expensive communication delay, the chip real estate does not expand in proportion to accommodate the increased transistor count. This leads to increased power density, and consequently higher operating temperature. The expected lifetime of the chip reduces as a result of the increased temperature. Research has shown that the chip failure rate doubles as the operating temperature rises by 10^{o}C [7]. Green and reliable computing is a hot topic of research.

Power consumption can be broken down into two components, namely dynamic and static. Dynamic power is the power consumed by the transistor when it switches between *on* and *off* states. For a given processor, the net dynamic power consumption is dictated by the transistor switching frequency, which in turn is dependent on both the operating frequency, and the *activity factor* of an executing application. On the other hand, static power is a result of tunneling and leakage currents on the chip, and is dependent on the supply voltage and the transistor design parameters, like the gate width. Lowering the supply voltage and frequency decreases both these components of power. Hence, proper DVFS can lead to significant power reduction.

The effect of DVFS on both performance and power consumption is dictated by how an executing application interacts with the hardware. Performance of applications that stress the main memory and hard disk is not significantly affected by lowering the frequency. Simultaneously, performance of applications that are compute intense is severely affected by DVFS. Also, the scaling down of dynamic power is more pronounced in a compute intense application compared to a memory intense application, when voltage and frequency are lowered. These observations indicate that application awareness provides important information that can be utilized to improve the effectiveness of DVFS.

Heterogeneous tasks executing on a processor drive the chip to different temperatures. The various architectural components are exercised to different extents, due to the varying CPU and memory intensities of these tasks. In [16], the authors report the peak temperatures of the chip when different SPEC2000 integer benchmarks are executed on processors from different technology generations. Their report provides the insight that the temperature difference on a single processor when executing different benchmarks goes up as the feature size shrinks. Executing a series of tasks with different temperature profiles causes the processor to heat up and cool down alternatively, a phenomenon known as thermal cycling. Thermal cycles are categorized into large and small scale cycles. The large scale thermal cycles are often a consequence of switching the processor on and off, and are expected to happen infrequently. The small scale thermal cycles occur more frequently, and are a function of the executing task characteristics.

Electronic circuits essentially constitute interconnection of materials with different thermal co-efficients (e.g. metals and dielectrics). These individual materials are subjected to differential expansion and contraction when the chip is heated and cooled. Due to their physical interconnection, a mechanical stress develops in this process, leading to die cracking, thin film cracking, solder joint fatigue, etc., over time. The authors of [13] mention that the corresponding failure mechanisms contribute to a large chunk of chip failures. Most of these failure mechanisms have not been analyzed statistically due to the complexity involved in the intricate interplay among the various mechanisms. Even the JEDEC testing standards [10], used for chip reliability quantification do not account for the consequences of small scale thermal cycles. We make an effort to reduce both the thermal cycles and the steady state temperatures (a relatively constant temperature to which a chip is heated when executing an application) in an integrated fashion, thereby increasing the expected chip lifetime.

We choose a soft-real time system model for application scheduling and execution. Task schedulers in real time environments typically base their schedule construction on the worst case execution times (WCET) of the tasks. When deadlines are not tight, static slack accumulates in the schedule. Since the actual execution times (AET) differ from the worst case assumptions, dynamic slack also accumulates in the schedule. Most modern processors typically support execution of the tasks at multiple frequencies (and correspondingly voltages), and as a result any positive slack generated in the system is currently exploited by lowering the operational voltages and frequencies. This reduces the overall energy consumption. A majority of the existing DVFS approaches are not thermally aware and aggressively tune down the operating voltages (frequencies) to achieve maximal energy savings. To the best of our knowledge, there have been no approaches targeted to handle the small scale thermal cycles.

2. RELATED WORK

DVFS has long been proposed as a means to achieve energy savings in real time environments. The research in this area can be classified based on three factors. The first classification factor is the optimized entity. Schemes proposed in [6, 12] concentrate on reducing dynamic power while the works in [1, 2, 4, 11, 19] integrate the reduction of static power along with the dynamic power.

The second classification factor is the time at which the optimization decision is taken. Approaches can be classified into completely offline [2, 11], combined offline/online [4, 6, 12] and completely online [1, 19]. In the offline schemes, the voltage and frequency pairs, (V, f), to be used at runtime are calculated at the schedule construction time and are directly used at runtime. The combined offline/online approaches use the knowledge of the workload at schedule time to construct possible (V, f) pairs. A selection algorithm decides at runtime the particular (V, f) pair to be used from the previously constructed pairs based on the operating conditions. Completely online approaches calculate the (V, f) pairs autonomously based on runtime conditions.

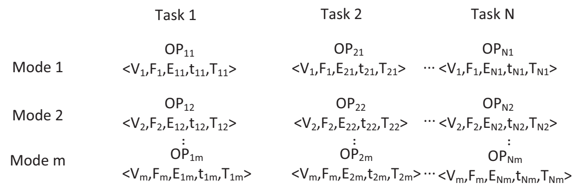
The third classification factor is the thermal awareness of the proposed approach. Thermally aware approaches [2, 3, 4] take into account the chip operating temperatures while scaling the voltages while the others do not. It is to be noted that schemes in [2] and [3] do not deal with thermal cycles. The authors of [4] show the effect of thermal cycles on lifetime reliability, but do not put forward an online approach to voltage scaling taking into account the thermal cycles. We develop a set of algorithms which reduce both the steady state temperature and thermal cycling, thereby improving the chip lifetime expectation. Our algorithms target a soft-real time environment, which is introduced next.

3. SYSTEM MODEL

Consider a soft-real time system employing an EDF scheduler. The tasks entering the system are assigned priorities based on the nearness of their absolute deadlines. Then, the tasks are scheduled for execution in decreasing order of their priority. Our algorithms take the schedule generated by the EDF or a similar scheduler as the input, as well as the deadlines for the tasks.

Let there be N tasks in the system $(T_1, T_2, ..., T_N)$ in order of decreasing priority i.e., $Pri(T_i) > Pri(T_j)$ when i<j and $Pri(T_k)$ represents the priority of task T_k. Let the underlying processor support m modes of operation, where each mode k is associated with a specific voltage v_k and frequency f_k. Each task T_i is associated with a deadline D_i, an expected execution time t_{ik}, an expected energy consumption E_{ik} (product of power consumption and execution time), and steady state temperature T_{ik} for the k^{th} mode of operation. These task characteristics are arranged in a 2-Dimensional grid, termed the operations table (OT). The OT has $m * N$ operational points (OP). Each OP is represented as a 5-tuple $< v_k, f_k, E_{ik}, t_{ik}, T_{ik} >$. Figure 1 shows the OT structure. The values of t_{ik}, E_{ik}, and T_{ik} can be obtained through profiling, statistical modeling, or a simulation based study. The entries along a column of the OT are arranged in increasing order of voltage and frequency. This implicitly arranges the entries in a column (from top to down) in the increasing order of the expected temperature, and in the decreasing order of expected execution time.

We develop a set of polynomial time algorithms to select an OP for each task in the schedule, such that the task deadlines are met and a set of thermal constraints (detailed later on) are satisfied. Since voltage and frequency are lowered from the rated maximum value to utilize any slack in the schedule, energy savings are expected as well. A set of OPs, selected one per task is referred to as an operational chain (OC). Thus, an OC dictates the set of voltages and frequencies assigned for the set of tasks in the schedule. We define the candidate OP inside the OC having the least expected steady state temperature to be the Lowest Temperature Point (LTP) and the corresponding temperature to be



	Task 1	Task 2		Task N
Mode 1	OP_{11} $\langle V_1,F_1,E_{11},t_{11},T_{11}\rangle$	OP_{21} $\langle V_1,F_1,E_{21},t_{21},T_{21}\rangle$	···	OP_{N1} $\langle V_1,F_1,E_{N1},t_{N1},T_{N1}\rangle$
Mode 2	OP_{12} $\langle V_2,F_2,E_{12},t_{12},T_{12}\rangle$	OP_{22} $\langle V_2,F_2,E_{22},t_{22},T_{22}\rangle$	···	OP_{N2} $\langle V_2,F_2,E_{N2},t_{N2},T_{N2}\rangle$
	:	:		:
Mode m	OP_{1m} $\langle V_m,F_m,E_{1m},t_{1m},T_{1m}\rangle$	OP_{2m} $\langle V_m,F_m,E_{2m},t_{2m},T_{2m}\rangle$	···	OP_{Nm} $\langle V_m,F_m,E_{Nm},t_{Nm},T_{Nm}\rangle$

Figure 1: Example Operations Table

the Coolest Operation Temperature (COT). Similarly, the hottest temperature in the OC is referred to as Hottest Operation Temperature (HOT). In the algorithms we develop, we target to keep the HOT below a critical value (T_c, which we fix at 340K) and minimize the temperature differences between the OPs selected.

In the construction of OT, we have used the thermal and performance model put forward by [8]. This mathematical model is used to calculate t_{ik}, E_{ik}, and T_{ik} for the tasks operating at different OPs, given the nominal values at the highest voltage and frequency settings, obtained through actual execution profiling. For clarity purposes, this model is elaborated in Subsection 3.1.

The following assumptions are made in developing the OC selection algorithms.

1. The soft real time environment guarantees a Quality of Service (QOS) requirement. QOS is defined as the ratio of the number of tasks meeting their deadline to the number of tasks scheduled for execution.
2. The real time tasks execute long enough so that the chip reaches a steady state temperature (may be different for each task). The transient thermal gradients that arise during task switching affect minimally.
3. The task performance and thermal characteristics are obtained in advance through extensive profiling or an analytical model.
4. The number of voltage and frequency pairs supported by the processor are discrete and finite.
5. Tasks are non pre-emptive and a task schedule constructed by an EDF scheduler is already known.

Under these assumptions, we design three different polynomial time global operational point selection algorithms (GOPS), to select the OPs for the tasks in the schedule. These algorithms are global in the sense that they consider the whole set of tasks together at once. Our approach is based on the expected execution time (EET) of the tasks rather than the WCETs, hence catering for both static and dynamic slack in the system. We monitor the AET values during task execution. Significant deviation of the AET from the expected time triggers a local operational point selection algorithm (LOPS) to deal with this incremental slack when each single task gets launched for execution. The algorithms are detailed in Section 4.

3.1 Calculation of t, E, and T

For an OP, the normalized voltage v_{norm} is defined as the ratio between the operating voltage V and the maximum permitted operating voltage V_{max}. Similarly, the normalized frequency f_{norm} is defined as the ratio between the operating frequency F and the maximum permitted operating frequency F_{max}.

$$v_{norm} = V/V_{max}, f_{norm} = F/F_{max} \tag{1}$$

The scaling factor for voltage is much lower than that of the frequency for commercially available processors. The relation between the operating voltage and frequency is approximated as

$$V = aF^{\alpha} \tag{2}$$

where a and α are hardware related parameters. Hence, the relation between v_{norm} and f_{norm} can be modeled as

$$v_{norm} = f_{norm}^{\alpha} \tag{3}$$

where α is an architecture dependent constant.

Two additional parameters, namely ρ and μ, are introduced to model the dependence of energy consumption, execution time and temperature on the nature of tasks and the underlying architecture. The parameter ρ represents the normalized value of the leakage power consumption to the total power consumption when the processor is operating at V_{max} and F_{max}. This is a measure of the transistor leakage characteristics, and is highly dependent on the microarchitectural design. The parameter μ represents the CPU intensity of the task. It is defined as the ratio of CPU computational time to the net execution time for the task. μ can be obtained by runtime profiling of the tasks. ρ and μ are given as

$$\rho = static\ power/(static\ power + dynamic\ power) \tag{4}$$

$$\mu = CPU\ time/(CPU\ time + memory\ access\ time) \tag{5}$$

The normalized task execution time, t is calculated as

$$t = (1-\mu) + (\mu)/f_{norm} \tag{6}$$

This is because the execution time component corresponding to the memory accesses does not change when changing the processor frequency and voltage settings alone. Only the CPU intensive component speeds up (slows down) at a more aggressive (less aggressive) OP.

Both the static and dynamic power scale down as voltage and frequency are decreased. If the static and dynamic powers at the maximum voltage and frequency are represented by P_S and P_D, then the corresponding values when using v_{norm} and f_{norm}, as given by [5] are

$$P_{S_{norm}} = \rho * v_{norm} \tag{7}$$

$$P_{D_{norm}} = (1-\rho) * {v_{norm}}^2 * f_{norm} \tag{8}$$

Using the definition of static and dynamic power, along with (3) yields the value for the normalized energy consumption.

$$e = (1-\rho)\mu f_{norm}^{2\alpha} + \rho(1-\mu) f_{norm}^{\alpha} + \rho\mu f_{norm}^{\alpha-1} \tag{9}$$

Finally, the normalized steady state temperature is approximately proportional to the power consumed by the processor. Hence, it is modeled as

$$T = ((1-\rho)\mu f_{norm}^{2\alpha+1} + \rho(1-\mu) f_{norm}^{\alpha+1} + \rho\mu f_{norm}^{\alpha})/((1-\mu) f_{norm} + \mu) \tag{10}$$

We use this equation to calculate the steady state temperatures for a set of synthetically generated input task sets used for our experiments in Section 5.1. We also gather real temperature data on a commercial processor for quantifying the improvement in reliability provided by our OC selection process. This procedure is described later on in Section 5.2.

4. OC SELECTION

Our algorithms select voltage and frequencies for the operation of different tasks scheduled for execution based on their EETs and the actual start times. As mentioned earlier, we formulate a two-step approach for the OC selection. The first step incorporates a GOPS that selects an OC based on EET values. If additional positive or negative slack arises in the system during execution, it is handled in the second step using LOPS. Since the job of the LOPS algorithm is primarily to mange and use the slack, it can increase the temperature gradient between the selected OPs. To minimize this effect, the LOPS algorithm selects an alternate OP

for a task in the case of positive runtime slack only if this gradient is not greatly aggravated (described later). The next two subsections detail the GOPS and LOPS algorithms.

4.1 GOPS Algorithms

The job of the GOPS algorithm is to select an OP for each task in the schedule to satisfy the following constraints, in order: 1) the task deadlines are met, as per the QOS requirement, 2) the thermal balance is maintained, and 3) energy savings are maximized. The processor is assumed to be in thermal balance if the peak temperature is low (below 340 K), and the sum of temperature gradients between adjacent tasks in the schedule (Sum_{Tdiff}) is minimized. The problem of assigning voltages and frequencies to the given task set can be viewed as selection of the proper OC from the OT that satisfies the above constraints.

Before a GOPS algorithm is employed to select OCs, the OT is trimmed to avoid the OP entries which would not find suitable OC partners on account of the large temperature differences with the OPs of the other tasks. The set of OPs are analyzed to find the common operating temperature range ($COTR$) for the different tasks. Let each task T_i has an operating temperature range given by $[T_{i\_min}, T_{i\_max}]$ where the two temperatures are reached when the OP is operated using the lowest and the highest (V, f) setting available, respectively. $COTR$, when j varies from 1 to N, is defined as $[max(T_{j\_min}), min(T_{j\_max})]$. The OT entries are initially pruned to remove the OPs whose operating temperatures fall outside the $COTR$.

A total of m^N OCs can be constructed from the OT since each task can operate at any of the available m operating modes. Using a brute force approach to compare the merit of all these combinations becomes a computationally daunting task as the number of tasks in the schedule increase. We develop and describe below a polynomial time *Peak reduction algorithm* (Pkr) to select the best possible OC that leverages the fact that the information stored in every column of the OT is ordered in terms of performance and temperature.

4.1.1 Peak Reduction Algorithm

The *Peak reduction* algorithm is iterative in nature, and each iteration involves selection of an OC (referred to as *o_chain*) from the feasible pool of OCs that can be formed using the entries in OT. The *goodness* of the selected *o_chain* is quantified by whether it can satisfy the constraints specified at the start of Section 4.1. The best OC found by the algorithm until any point in its execution is referred to as *CurrentBest_OC*, and is maintained. The algorithm initializes with the OC consisting of the highest frequency OPs for all the tasks in the system as the *CurrentBest_OC*. Once a new *o_chain* is selected during an iteration, it replaces the *CurrentBest_OC* if found to be having more *goodness* than the existing *CurrentBest_OC*.

To select the OC during an iteration i, three different schemes are alternatively used. Their descriptions and usage conditions are listed in Table 1. In the table, we use OC_i to denote the OC selected at the end of iteration i. A *feasibilityflag* is used to denote the deadline feasibility of a schedule selected during an iteration. A selected OC is said to be deadline (in)feasible if all (any of) the tasks in the OC do (not) satisfy their deadline constraints. In addition, the algorithm utilizes a *doneflag* to signal its termination, upon finding a satisfactory OC. Listing 1 depicts the pseudo code.

Scheme	new OC	Usage condition
lowerSelect	Set of OPs lying one step below the OPs used in OC_{i-1}	$OC_{i-1} = OC_{i-2}$
nearSelect	Set of OPs lying closest to COT in OC_{i-1}	$OC_{i-1} \neq OC_{i-2}$ and $ff = 1$
alternateSelect	Set of OPs lying closest to n^{th} COP in OC_{i-1} where $(n-1)^{th}$ COP is used for previous iteration of alternateSelect	$OC_{i-1} \neq OC_{i-2}$ and $ff = 0$

ff: feasibilityflag

Table 1: OC selection schemes for Peak Reduction

Listing 1: The Peak Reduction Algorithm

```
Function PeakReduction_Algorithm
feasibilityflag=1; doneflag=0; initializer();
while(doneflag==0)
{ if(feasibilityflag==0) o_chain=alternateSelect();
        else
        { if(o_chain==prev_chain) o_chain=lowerSelect();
                else o_chain=nearestSelect(o_chain.COT);
                feasibilityflag=checkFeasibility(o_chain);
                if(feasibilityflag==1) optimizeChain(); }
        prev_chain=o_chain; }
return CurrentBest_OC;
End Function

Function alternateSelect
COPindex=1
while(COPindex < num_of_tasks)
{ o_chain=nearestSelect(o_chain.COT);
        feasibilityflag=checkFeasibility(o_chain);
        if(feasibilityflag==1) optimizeChain();
        COPindex=COPindex + 1 }
doneflag=1;
End Function

Function optimizeChain
if(CurrentBest_OC.HOT < 340 && o_chain.HOT < 340)
        if(CurrentBest_OC.Sum_Tdiff > o_chain.Sum_Tdiff)
                CurrentBest_OC = o_chain;
else CurrentBest_OC = o_chain;
End Function
```

The algorithm has two phases, namely *Tdown* and *Tup*. The algorithm starts with the *Tdown* phase by executing the *initializer* function which sets up the *CurrentBest_OC* to be the set of OPs resulting from maximum voltage and frequency operation of each task in the schedule. The *Tdown* phase iteratively finds OCs with lower operating temperatures. In each iteration of the *Tdown* phase, the COT of the OC selected in the previous iteration is set to be the *target temperature*. The OPs of the different tasks closest to this *target temperature* are then selected into the new OC. This functionality is achieved using the *nearestSelect* technique. If the newly generated OC is the same as the OC from the previous iteration, we utilize the *lowerSelect* technique to select a new OC consisting of OPs for each task that are just one step below in temperature when compared to the OPs in the current OC. As a result of these two techniques, the processor temperature is reduced. Also, the tasks are slowed down in the process. Once the algorithm reaches an OC which is not deadline feasible, *feasibilityflag* is made 0 and the algorithm enters the *Tup* phase.

In the *Tup* phase, the *alternateSelect* function is used to select OCs with incrementally higher operating temperatures by successively employing the temperatures of the different OPs consisting the currently selected deadline infeasible OC as the *target temperatures*. For a given OT, there can be a maximum of $m * N$ *Tdown* iterations and a maximum of $N - 1$ *Tup* iterations. Each *Tup* or *Tdown* iteration has a time complexity of O (mN). Hence, the worst case complexity of this algorithm is O(m^2N^2).

To decrease the runtime of the peak reduction algorithm, two other algorithms are developed. The details of these algorithms are presented in the next two subsections.

4.1.2 Peak Reduction- Forced Termination

The first alternative to peak reduction algorithm is the forced termination algorithm. This algorithm, *Pkr_force*, parallels the peak reduction algorithm until the HOT in the $CurrentBest\_OC$ drops below 340 K and all the OPs constituting the $CurrentBest\_OC$ are constrained within a window of 5^o K. Under such conditions, the thermal balance is assumed to be achieved, and the algorithm terminates. Although this algorithm has the same worst case complexity as the original *Peak reduction* algorithm, we expect the actual number of iterations to go down. Experiments have been conducted to verify this claim and the relevant results are presented later. However, the energy savings provided by this algorithm are expected to be lower than those provided by *Peak reduction* due to premature termination. Hence, this algorithm trades off the effectiveness of *Peak reduction* with the execution time. A *Window based selection* algorithm, described below, performs the same tradeoff more aggressively.

4.1.3 Window Based Selection Algorithm

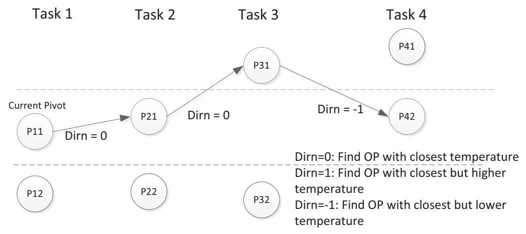
The second alternative to the *Peak reduction* algorithm is a Window based OP selection ($WOPS$) algorithm. This algorithm restricts the selection of OPs in favor of reducing Sum_{Tdiff}, while limiting the possibilities considered in the selection process. This algorithm maintains a virtual temperature window and selects OPs for each task within/ closer to this window. The $WOPS$ algorithm starts by initializing the $CurrentBest\_OC$ similar to the *Peak reduction* approach. This algorithm also works iteratively. In each iteration, a *pivot* OP is chosen for the first task in the schedule. Once a *pivot* is chosen, a virtual window (bounded by lower and upper virtual bounds) is constructed encompassing the temperatures lying within a difference of 5°K from the chosen *pivot's* temperature. After the virtual window is formed, the algorithm proceeds to select an OP for each of the other tasks in the system that ensures the chip steady state temperature stays within, or at least closest to the window.

Listing 2: The WOPS Algorithm

```
Function WOPS_Algorithm
initialize();
while((o_chain[0]=selectpivot())!=NULL)
{ calculateVbounds();
        while(next_task_id < num_of_tasks)
        { o_chain[next_task_id]=selectionAlgo(o_chain,next_task_id,dirn);
                dirn=setDirection();
                next_task_id++; }
        current_schedule_feasibility=checkFeasibility(o_chain);
        if(current_schedule_feasibility)
        { calculateNetSwing(); updateBestChain(); updateFlags(); }
}
End Function
```

Listing 2 shows the pseudo code for our WOPS algorithm. The function *selectionAlgo* takes the updated *o_chain* as the input to find the next OP to be included in the chain. A direction variable *dirn* is also maintained to guide the selection process. This variable is updated on the fly to force the selection algorithm to choose OPs closer to the virtual bounds, if required. This process is illustrated in Figure 2. The worst case complexity for one iteration of *selectionAlgo* is O(Nlogm), since there are N tasks and a binary search can be used to find the candidate OP for each task that is closest to the window.

Figure 2 shows the *o_chain* selection round when there are 4 tasks in the system, each of which can operate at 2 different OPs. Assume that the point *P11* is selected as the



Task 1
Task 2
Task 3
Task 4
P41
P31
Current Pivot
P21
Dirn = 0
Dirn = -1
P42
P11
Dirn = 0
Dirn=0: Find OP with closest temperature
Dirn=1: Find OP with closest but higher temperature
Dirn=-1: Find OP with closest but lower temperature
P12
P22
P32

Figure 2: Example OC selection using WOPS

pivot currently. We term the most recently selected OP as the *o_chain* header. The dashed lines in the figure represent the virtual temperature bounds. Initially, the *o_chain* header lies between the virtual bounds. *dirn* is set to 0 to indicate this. For Task 2, there are two potential OPs available for selection. The selection algorithm selects the closest point to *P11*, which is *P21*. *P31* is selected for Task 3 using similar logic. Since *P31* falls out of the virtual bounds, it is beneficial for the selection algorithm to select the next OP towards the upper virtual bound, in order to avoid a large thermal gradient with respect to *P11*. *dirn* is set to -1 to achieve this effect. The selection algorithm thus selects $P42$ instead of $P41$.

Once a complete OC is selected, its deadline feasibility is calculated using the *checkFeasibility* function, which takes up O(N) processing time. If the feasibility check succeeds, the newly selected OC replaces the $CurrentBest\_OC$ utilizing the same logic as that used for *Peak reduction*.

At the end of the round, the *pivot* is marked as invalid and the algorithm starts another round by selecting a new *pivot*. When the algorithm runs out of valid *pivots* to choose from, it terminates. Since there are m *pivot* points that can be chosen, the total complexity of the algorithm is $O(Nmlogm)$. The time complexity can be further reduced to $O(Nlogm^2)$ by using a binary search for selection of the *pivots*. However, the reduction in algorithm execution time is not anticipated to be very high since m is typically small (4-16).

4.2 LOPS Algorithm

The LOPS algorithm takes the best OC selected by a GOPS algorithm as input and deals with the runtime slack by (potentially) optimizing each task's OP locally before it starts execution. Since the execution of LOPS falls into the actual task execution schedule, it is designed to be faster. The runtime slack is calculated as the difference of the estimated start time (obtained from schedule constructed by GOPS) of a task and the actual start time of the task (obtained during execution). When there is negative runtime slack in the system, the LOPS algorithm selects a more aggressive OP compared to the one preselected by GOPS. On the other hand, when there is a positive runtime slack, the LOPS algorithm makes a change to the OP selected by the GOPS algorithm only if the newly selected point does not cause a local Sum_{Tdiff} of *Thresh* with respect to its adjacent selected OPs in the schedule. For our experiments, we set the value of *Thresh* at 5^o K. The LOPS algorithm, as listed in Listing 3, has a time complexity of O(m).

As described earlier, the GOPS algorithm creates an initial OC based on EET values. To account for different degrees of runtime slack, the GOPS algorithm creates additional OCs based on modified EET values. Each OC is formed by utilizing the modified EET values for each task obtained by multiplying the original EET with a multiplier, which we term the scaling factor. We have limited the bounds of this scaling factor to 0.8 and 1.2 (with a step of 0.05). Using this methodology, we can cater for dynamic slack of 20% (both +ve and -ve). The LOPS algorithm can employ any of these OCs utilizing the different scaling factors during task execution.

The entire task execution schedule is split into windows. Each window contains W tasks. For our experiments, we have set the value of W to 1024. During task execution, a miss counter (MC) is employed to monitor the number of deadline misses so far in the schedule. After i^{th} window of tasks finishes execution, the scaling factor used for the next window $i+1$ is adjusted based on the criterion below.

$$(MC\ value\ + W)/(W * (i+1)) < 1 - QOS \qquad (11)$$

If the above criterion is satisfied and there exists positive slack in the system, the scaling factor employed for the next window is reduced by a step to improve the energy savings. If there exists negative slack in the system, the OCs corresponding to the next higher scaling factor is utilized by the LOPS algorithm for the next window. If continuous negative slack is observed, OCs corresponding to much higher scaling factors (multiple step difference) are utilized. More details are not provided in the interest of space.

Listing 3: The LOPS Algorithm

```
Function LOPS_Algorithm
int i;
if(current_slack == 0) return selected_point;
else if(current_slack < 0)
        for(i=selected_point+1;i< m; i++)
                if((i.time−selected.time)<current_slack) return i;
else if(current_slack > 0)
        for(i=selected_point−1;i >=0;i−−)
                if((i.time−selected.time)<current_slack)
                 if(cycleEffect(selected_point,i,task_id)< Thresh) return i;
End Function
```

5. EXPERIMENTATION

5.1 Experimentation with synthetic task sets

The first set of experiments we performed is intended to test the effectiveness of our GOPS algorithm. We synthetically generated 1000 task sets with μ values ranging between 0.1 and 1. We have separately generated and experimented with task sets containing 4, 8 and 16 different periodic tasks, and for architectures with varying values of ρ ranging between 0.2 and 0.7. However, we present the results for 8 task schedules only, in the interest of space. We assumed that the processor supports 8 different (V, f) combinations, given by $(V, f) \in$ {(1.2 300), (1.23, 400), (1.35, 500), (1.53, 600),(1.75, 700), (2.0, 800), (2.35, 900), (2.80, 1000)}, where V values are in volts and f values are in MHz. The tasks' steady state temperatures (SSTs) at the OP with maximum voltage and frequency is selected in the interval of [290K,390K] based on the value of μ. Higher the value of μ, higher is the SST selected. Once the task parameters at the nominal OPs are fixed, their corresponding values for the other supported OPs are calculated using the model detailed in Section 3.1. The task deadlines are also randomly generated. The deadline for a task i is selected between $[1.2\sum_{j=1}^{i} t_j, 2\sum_{j=1}^{i} t_j]$ where t_j represents the EET for task j.

Figures 3 shows the strength of the different GOPS algorithms in minimizing the intertask SST differences. In the figure, the $Nominal\_OP$ scheme corresponds to operating all tasks at maximum (V, f). The Pkr algorithm performs better in comparison to the other two algorithms, because it considers more OP combinations in the selection of OC.

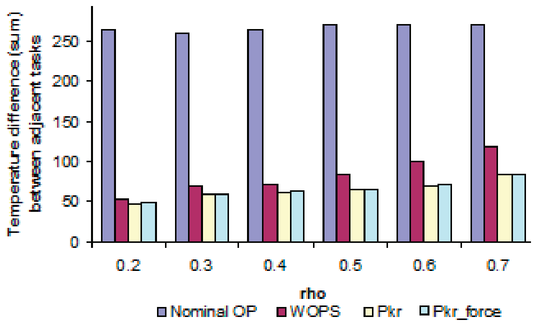


Figure 3: Sum_{Tdiff} with 8 task workloads

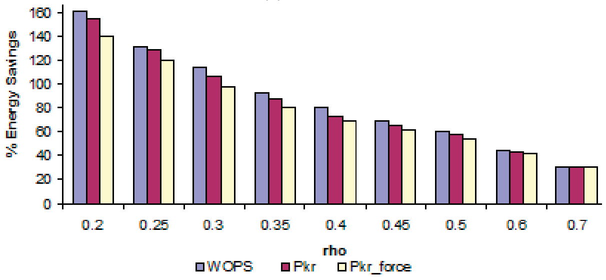


Figure 4: Energy savings with 8 task workloads

WOPS performs bad compared to Pkr since the OCs selected are restrictive in nature (each *pivot* gives rise to only one possible OC). Figure 4 shows the energy savings obtained by using the OCs selected by the different GOPS algorithms.

Two conclusions can be drawn by analyzing these results. As the value of ρ increases, the energy saving goes down. This is because the dynamic power is proportional to $V^2 f$, whereas the static power is proportional to just V. Hence, scaling the OP has a higher effect on the dynamic power compared to the static power. As such, when the underlying architecture consumes more static power (higher ρ), scaling down the OP would achieve lower energy savings. As the virtual bounds utilized in the GOPS algorithm are stretched to increase the window size, the algorithm has, at its disposal, a higher number of candidate choices for the OP selection, and selects the one minimizing the energy consumption. As a result, the energy savings are higher.

Though the theoretical maximum number of iterations in Pkr are high (m x N + (N-1)), our experiments revealed that the actual value is much smaller than this bound. For example, the average number of iterations for Pkr is observed to be 9 for 4-task workloads and 6 for 8-task workloads. On the other hand, the average number of iterations for Pkr_force is observed to be 5.3 for 4-task workloads and 3.5 for 8-task workloads. Even though the Pkr_force algorithm finishes in 50% of the time when compared to Pkr, it performs almost as good as the Pkr algorithm in minimizing SST differences and achieving energy gains.

Using this observation, we can conclude that Pkr_force can be used for GOPS instead of Pkr without compromising significantly on the energy and thermal benefits produced. It is observed that though Pkr_force performs better than WOPS in minimizing Sum_{Tdiff}, it consumes more time as the number of tasks increase. Hence, thermal (reliability) demands can be used to guide the selection between them.

To demonstrate how our algorithms perform with respect to meeting the QOS constraints set, We have scheduled the tasks separately for different QOS constraints ranging between 0.91 and 0.99. It has been verified that our LOPS scheme meets the QOS requirements specified. We also observed that as the QOS constraint is tightened, the scheduler utilizes the positive runtime slack more pessimistically, resulting in lower energy savings. However, this difference in savings is marginal (around 5%).

QOS Constraint (as %)	QOS Delivered (as %)	Avg. Energy Savings (as %)
91	92.13	51.75
93	93.85	50.17
95	95.54	48.86
97	97.92	46.47
99	99.99	46.76

Table 2: QOS deliverance and Energy savings

5.2 Augmentation of Lifetime Reliability

Reduction of thermal cycling and operating temperatures on the chip improves its lifetime. To quantify this effect of our schemes, we utilize the chip Mean Time To Failure ($MTTF$), a metric that is widely used to quantify chip reliability [14, 16, 17]. $MTTF$ is defined as the mean expected time to fail of a non-repairable component. Accordingly, the failure mechanisms investigated should make the chip non-functional and non-repairable. The following failure mechanisms fit this paradigm- Electromigration, Stress migration, large and small scale Thermal cycles, Time dependent dielectric breakdown, and Negative bias temperature instability. Although there are many other failure mechanisms, we have restricted ourselves to using the most investigated ones, due to availability of near-accurate analytical models to predict the $MTTF$ associated. We have used the RAMP model [16] to calculate the $MTTF$ values. The $MTTF$ for just the small scale thermal cycles is derived from the model used in [18]. For each failure mechanism, we calculate the ratio of $MTTF$ when the processor executes tasks with the OPs selected using our scheme to the $MTTF$ obtained when operating at rated maximum voltage and frequency (referred to as *max-stop*). In the latter case, the tasks finish faster, and the processor enters a low power mode, which is accounted for. Since operating each task at a particular OP results in a different $MTTF$ value, we use weighted harmonic mean to calculate the average $MTTF$ value. In the following, we briefly describe the failure mechanisms and our experimentations.

5.2.1 Chip Failure Modeling

Electromigration.

The atoms in interconnects are gradually displaced due to momentum transfer by the conducting electrons. Because of this, the atoms get shifted within interconnect and lead to higher resistance values and possibly, shorts. The $MTTF$ due to Electromigration is given by

$$MTTF_{EM} \propto (J)^{-n} e^{E_{aEM}/KT} \tag{12}$$

where J is the current density in interconnect, n is a material dependent constant, E_{aEM} is the activation energy for electromigration, K is the Boltzmann constant and T is the steady state operating temperature. The value of J is directly proportional to the operating voltage and frequency.

Stress Migration.

Due to differential thermal coefficients in the interconnect material, a thermo-mechanical stress is generated when the interconnect heats leading to migration of atoms. It results in open circuits and high resistance values within interconnects. The $MTTF$ due to Stress migration is given by

$$MTTF_{SM} \propto |T_0 - T|^{-m} e^{E_{aSM}/KT} \tag{13}$$

where T_0 is the metal deposition temperature, T is the steady state operating temperature, m is a material dependent constant, E_{aSM} is the activation energy for stress migration and K is the Boltzmann constant.

Time Dependent Dielectric Breakdown.

As transistor size shrinks, so do the number of atoms contained within the gate oxide. As the gate oxide wears out during the chip lifetime, the gate control over the MOSFET channel decreases and eventually leads to a failure. The $MTTF$ for time dependent dielectric breakdown is given by

$$MTTF_{TDDB} \propto (1/V)^{a-bT} e^{(X+Y/T+ZT)/KT} \tag{14}$$

where a, b, X, Y and Z are curve fitting parameters, T is the steady state operating temperature, V is the operational voltage and K is the Boltzmann constant.

Thermal Cycling.

We have mentioned both the large and small scale thermal cycling earlier. The $MTTF$ due to large scale thermal cycling is given by

$$MTTF_{LTC} \propto (1/(T - T_{ambient}))^{q} \tag{15}$$

where T is the steady state operating temperature, $T_{ambient}$ is the ambient temperature and q is the coffin-Manson exponent which is a measure of the effect of thermo-mechanical stress. Small scale thermal cycles cause solder joint failures due to uneven expansion and contraction. We model the $MTTF$ for small scale thermal cycles based on the $MTTF$ for the solder joints. This is given by

$$MTTF_{STC} \propto |T_1 - T_2|^{n} e^{E_a/KT_{max}} \tag{16}$$

where T_1 and T_2 are the steady state temperatures of two tasks, T_{max} is the maximum value between T_1 and T_2, n is the Coffin Manson based exponent, E_a is the activation energy and K is the Boltzmann constant.

Negative Bias Temperature Instability.

The negative bias applied to the gate results in gradual increase in the threshold voltage and associated decrease in drain current and transconductance. The $MTTF$ for negative bias temperature instability is given by

$$MTTF_{NBTI} \propto \{\{(ln(E) - ln(E-C)\} * T/e^{-D/KT}\}^{1/\beta} \tag{17}$$

$$E = A/(1 + 2e^{B/KT}) \tag{18}$$

where A, B, C, D and β are curve fitting parameters, K is the Boltzmann constant and T is the steady state temperature.

5.2.2 Experimentation Details

A set of 4 tasks from varied computing scenarios is chosen for experimentation purpose. The tasks chosen are *bzip2*- a compression application, *mcf*- a vehicular scheduling algorithm, *applu*- a PDE solver, and *stream*- a stream processing benchmark used to quantify memory bandwidth. The first three tasks are obtained from the SPEC benchmark suite [15], and the last task is the industry standard for measurement of computer bandwidth [9]. These tasks have varied computing requirements, and thus provide an excellent base for analyzing the effects of executing diverse tasks sequentially on a processor. For our experiments, we have utilized a Dell Optiplex Desktop housing an Intel Core 2 Duo processor (T7700). The platform runs RHEL 5 operating system and supports four different (V, f) pairs with frequency values of 1600 MHz, 1867 MHz, 2133 MHz, and 2400 MHz. The platform houses a high-performance PWM fan and a heat sink. A PWM fan alters its rotation speed according to the current chip temperature. When temperature rises, the fan speed increases in order to compensate for this. Notice that this mechanism attenuates thermal cycles, and leaves out lesser room for thermal cycling management using our OC

selection scheme. Thus, the results we obtain are pessimistic as such a PWM control is impractical in many real-world scenarios, especially mobile computing platforms. However, any improvement noticed in the $MTTF$ on this platform will really showcase the power of our OC selection process.

We have run the selected tasks individually on the experimentation platform under the different (V, f) pairs a large number of times (10,000) to obtain performance and temperature data. The Linux *time* command is used to record task execution time. *lm_sensors* is used to monitor the chip temperature and *cpufreq-utils* is used to switch between the available (V, f) pairs. A set of deadlines are generated for the tasks using the procedure outlined in Section 5.1. The task execution order is then determined using an EDF scheduler. The obtained performance and temperature data, along with the task execution order, is used by our OC selection scheme to select the OPs for each task. The temperature values resulting from the execution of the tasks utilizing our OPs are used to calculate the $MTTF$ values associated with the various failure mechanisms modeled.

In Figure 5, the improvement of $MTTF$ obtained using our scheme when compared with the *max-stop* scheme is plotted as a ratio on Y-axis. The different failure mechanisms modeled are shown on X-axis. Since we employ $DVFS$, we expect average chip temperature to go down under reasonable slack conditions, and this effect is evident from Figure 5. The highest improvement in reliability is observed for short thermal cycling, as expected. However, the huge factor (16) of improvement in spite of the PWM controlled cooling mechanism is clearly an effective advocate for our OC selection mechanism. The second highest improvement is observed for TDDB, which again is expected since the operating voltage plays a strong role in affecting the associated $MTTF$, and is exploited well by $DVFS$. Our results also indicate that *max-stop* is not very effective in addressing chip reliability issues, although it leads to longer sleep states.

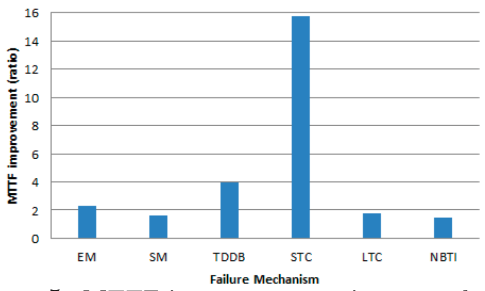


Figure 5: MTTF improvement using our schemes

6. CONCLUSION

In this paper, we have presented novel scheduling algorithms that perform OP assignment for tasks in a soft real time system, to minimize the thermal peaks and thermal cycles. Our experiments based on data obtained through actual application execution show that incorporation of these algorithms in (V, f) selection process can improve the chip $MTTF$ due to small scale thermal cycling more than 16 fold. These techniques are very relevant for future processors where chip heating becomes the number one concern.

In future, we will investigate schedule readjustment along with the OP assignment to achieve better performance. For longer tasks, we will put forward measures for intratask OP assignment. The OP assignment can be coupled with thermal balancing on multi-cores to reduce hotspots.

7. REFERENCES

[1] S. Akui, K. Seno, M. Nakai, T. Meguro, T. Seki, T. Kondo, A. Hashiguchi, H. Kawahara, K. Kumano, and M. Shimura. Dynamic voltage and frequency management for a low-power embedded microprocessor. In *Solid-State Circuits Conference*, feb. 2004.

[2] M. Bao, A. Andrei, P. Eles, and Z. Peng. Temperature-aware voltage selection for energy optimization. In *Design, Automation and Test in Europe*, march 2008.

[3] M. Bao, A. Andrei, P. Eles, and Z. Peng. On-line thermal aware dynamic voltage scaling for energy optimization with frequency/temperature dependency consideration. In *46th ACM/IEEE Design Automation Conference*, july 2009.

[4] A. Coskun, R. Strong, D. Tullsen, and T. Rosing. Evaluating the impact of job scheduling and power management on processor lifetime for chip multiprocessors. In *Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*, 2009.

[5] G. Dhiman and T. Rosing. System-level power management using online learning. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 2009.

[6] G. Dhiman and T. S. Rosing. Dynamic voltage frequency scaling for multi-tasking systems using online learning. In *Proceedings of the 2007 international symposium on Low power electronics and design*, 2007.

[7] DoD. Aci broad agency announcement w911nf-12-r-0010. "www.arl.army.mil/www/default.cfm?page=8", 2012.

[8] J. L. Hao Shen and Q. Qiu. Learning based dvfs for simultaneous energy temperature and performance control. In *ISQED'2012*, july 2012.

[9] http://www.streambench.org/. Stream benchmark. "http://www.cs.virginia.edu/stream/".

[10] jedec.org. Jedec thermal cycling test. "http://www.jedec.org/standards-documents/results/jesd22-a104", 2009.

[11] Z. Lu, Y. Zhang, M. Stan, J. Lach, and K. Skadron. Procrastinating voltage scheduling with discrete frequency sets. In *Proceedings of the conference on Design, automation and test in Europe*, 2006.

[12] S. Oh, J. Kim, S. Kim, and C.-M. Kyung. Task partitioning algorithm for intra-task dynamic voltage scaling. In *IEEE International Symposium on Circuits and Systems*, may 2008.

[13] S. Salemi, L. Yang, J. Dai, J. Qin, and J. B. Bernstein. *Physics-of-Failure Based Handbook of Microelectronic Systems*. A K Peters, UTICA, NY, 2008.

[14] T. Simunic, K. Mihic, and G. Micheli. Optimization of reliability and power consumption in systems on a chip. In V. Paliouras, J. Vounckx, and D. Verkest, editors, *Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation*, volume 3728 of *Lecture Notes in Computer Science*, pages 237–246. Springer Berlin Heidelberg, 2005.

[15] spec.org. Spec cpu 2000. "http://www.spec.org/cpu2000/", 2000.

[16] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. *Lifetime Reliability: Toward an Architectural Solution.* IEEE Micro, May/Jun 2005.

[17] V. Subramanian, P. Ramesh, and A. Somani. Managing the impact of on-chip temperature on the lifetime reliability of reliably overclocked systems. In *Dependability, 2009. DEPEND '09. Second International Conference on*, 2009.

[18] V. Vasudevan and X. Fan. An acceleration model for lead-free (sac) solder joint reliability under thermal cycling. In *58th Electronic Components and Technology Conference*, may 2008.

[19] L. Yuan, S. Leventhal, J. Gu, and G. Qu. Talk: A temperature-aware leakage minimization technique for real-time systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, oct. 2011.