Low-Complexity Policies for Energy-Performance Tradeoff in Chip-Multi-Processors

Avshalom Elyada, Ran Ginosar, Uri Weiser

Dept. Of Electrical Engineering

Technion, Israel Institute of Technology

Haifa 32000, Israel

Abstract

Chip-Multi-Processors (CMP) utilize multiple energy-efficient Processing Elements (PEs) to deliver high performance while maintaining an efficient ratio of performance to energy-consumption . In order to utilize CMP resources, the software application is split into multiple tasks that are executed in parallel on the PEs. Dynamic frequency-Voltage Scaling (DVS) balances performance and energy consumption by dynamically varying a PE's frequency-voltage *workpoint* in order to save energy while meeting performance requirements. This work addresses DVS policies for CMP. We consider multi-task CMP applications with unknown workloads. We dynamically set frequency-voltage workpoints for each PE in the CMP, attempting to minimize a defined energy-performance criterion.. Other DVS methods typically use high complexity optimization techniques, which limits the possibility of real-time implementation in performance-driven, energy-aware systems. In contrast, we investigate simple heuristic DVS policies for simplified serial/parallel task-graphs. We compare the results of our polices to a theoretical best-case solution and show that these lightweight heuristics achieve good results with low complexity. In most cases the very simple Constant policy is the most cost-effective.

1. Introduction

Chip-Multi-Processors (CMP) achieve high performance while maintaining an acceptable ratio of performance to energy consumption, in comparison to traditional single-core architectures. Performamce improvement techniques of single-core architectures, mainly include (1) taking advantage of shrinking gate-delays in order to increase the operating frequency, and (2) using the increased transistor density to add performance-enhancing microarchitecture features [1],. Increasing the operation frequency beyond a certain point is energy inefficient, since energy consumption is roughly quadratic in frequency, Features such as large caches, deep execution pipes and complex branch predictors yield a decreasing performance/energy return [2]. High energy consumption shortens the battery life of mobile devices, and may cause power delivery and heat dispersion problems, which consequently limit feasible frequencies and performance. CMP architectures, on the other hand, integrate multiple relatively small and simple PEs, potentially enabling linear scaling of performance [3].

Dynamic frequency-Voltage Scaling (DVS) is a widely practiced [4] and researched [5-7] technique for energy-performance tradeoff. When using DVS in CMP, a PE's frequency is altered dynamically to meet current performance requirementsm while consuming no more energy than is necessary. PE supply voltage is also adjusted in conjunction with frequency; usually kept at the lowest feasible value that still enables circuit operation and timing at the current frequency. Scaling the frequency-voltage workpoint (f,V) can result in near—quadratic energy savings [6].

In a CMP running multiple dependent tasks, DVS may save energy without degrading performance. Typically, at any given time, one of the tasks constitutes the performance bottleneck. Other PEs can be slowed down, saving energy without affecting total performance. We refer to the tactic of slowing down non-critical tasks as *slack-utilization*.

When all task workloads are known in advance, the *DVS policy* sets PE frequencies to utilize precisely all available time-slacks and thus save the maximum possible energy without affecting performance. Typically however, task workloads are unknown in advance and efficient slack utilization is non-trivial. A DVS policy which assumes worst-case workloads achieves limited energy savings, since aggressive workpoints need to be set to maintain required performance in the worst case. Conversely, overestimating slack can lead to performance degradation. When workloads are unknown they must be estimated, and the method of estimation is a primary factor of the efficiency of DVS policies.

Selecting a criterion for DVS policy efficiency is not a clear-cut issue, since different DVS-capable CMPs for different applications have different energy- and power-performance requirements. For example, mobile battery-operated devices aim at long battery life as well as performance, while desktop systems and servers typically optimize power rather than energy consumption.

Consider a system whose only objective is to maximize performance. The best policy would be to run at maximum frequency at all times (in this work we refer to this policy as the *f*-max policy). Conversely, a system which is concerned only with energy minimization should always run at the minimum frequency (*f*-min policy). A more interesting and more practical case is that of a system which strives to balance between the two: tuning up frequencies only to the point where the increased energy consumption is deemed justifiable, and likewise saving energy by running slower, but only to the point considered as tolerable performance. This choice is reflected in the criterion selected to assess alternative DVS policies, as discussed in Section 2.3.

DVS policies vary in computational complexuty. If the required calculations for policy implementation are to be integrated into the system itself and done in real-time rather than offline and externally, then it is essential that the energy-consumption and delay of the calculation itself be minimal. Spending a substantial amount of execution time and energy just to calculate each workpoint may considerably offset the savings aimed at, making it impractical for use in a performance- and energy-aware system. Thus, a practical slack-utilization computation must weigh savings accomplished versus its own complexity.

Previously published DVS policies for CMP formulate precise optimization problems [8-10] seeking optimal solutions, typically at the cost of high computational complexity. In contrast, we introduce lightweight heuristic DVS-polices, and show that they achieve good results compared to theoretical bounds. We show further that in most cases the simplest policy is the most cost-effective.

This paper is organized as follows. Section 2 defines the minimization criterion and formulates the DVS minimization problem. The various lightweight huristic DVS policies are presented in Section 3, and analyzed in Section 4. Summary and conclusions are drawn in Section 5.

2. Definitions and Problem Formulation

Consider a CMP platform running a multi-task application. At any given time, each PE accommodates one software task. Each PE is independently DVS-capable, i.e., the workpoint of each PE is controlled independently of the other PEs. A DVS policy dynamically assigns (f,V) workpoints to each PE, in order to minimize a specified performance and energy-consumption criterion. For simplicity, all PEs are identical, although these results can be generalized to *heterogeneous*-PE systems [3, 11, 12] with few modifications.

2.1. DVS Hardware Model

DVS-Capable PE Model

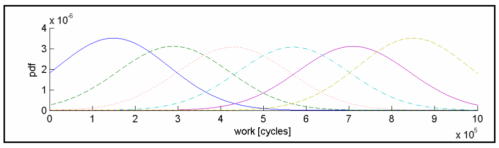
We assume that each PE in the system is capable of operating at a clock frequency within the range $f \in [f_{\min}, f_{\max}]$ $^{\text{cycles}}\!/_{\text{sec}}$, and may also be in a standby mode. At any frequency, the PE operates at the minimum feasible supply voltage, defining a frequency-voltage (f,V) operation curve (see [13] for details). Note that for simplicity we employ a continuous frequency model, while typical processors operate at only a finite set of discrete frequencies. The rate of changing the frequency is also limited in practice due to transition cost in both energy and performance, and also complexity of frequent recalculation of the workpoint. It is unrealistic to change the frequency too often, and we assume a small number of frequency changes,, therefore we can neglect the transition overhead.

PE Power, Energy, and Execution Time

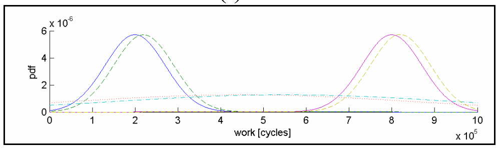
We denote the total power consumption of a PE by $P(f)$ $\frac{\text{joules}}{\text{sec}}$, where f is the PE's current frequency. As mentioned above, the operating frequency implicitly defines a corresponding supply voltage. We consider $P(0)$ to be the standby power, consumed by the PE when it is not doing any work. We further define the energy consumed *per cycle*, denoted by $e(f)$ $^{\text{joules}}\!/_{\text{cycle}}$. By definition:

$$e(f) = {}^{P(f)}\!/_{f} . \qquad (1)$$

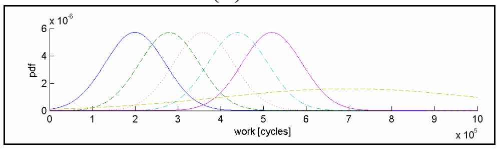
For a task of unknown workload, the cumulative density function $cdf_W(w)$ of the workload W is defined as the probability that the task will be completed within w cycles or less: $cdf_W(w) = \Pr(W \le w)$. Hence the probability that the task will take w cycles or more to execute is $1 - cdf_W(w-1)$. Some example distributions are displayed in Figure 1 below, which shows probability density functions, conventionally defined as $pdf_W(w) = cdf'_W(w)$. These distributions are used in our simulations, as described in Section 4.



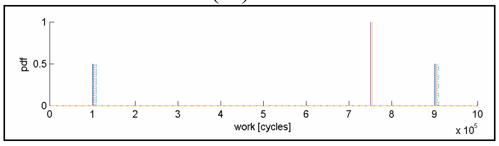
(i)



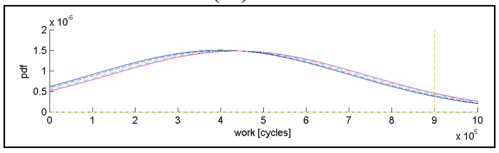
(ii)



(iii)



(iv)



(v)

Figure 1: Workload distributions: Examples on a 6-PE CMP.

If the cumulative density and energy-per-cycle functions are known, we can formulate the expected energy required to execute a task of workload W on a PE p by the following expression:

$$E_p = \sum_{w=1}^{\infty} e(f_w)\left[1 - cdf_W(w-1)\right], \qquad (2)$$

where f_w is the frequency at cycle w. Eq. (2) sums the energy-per-cycle times the probability that the task will still be running at that cycle. Convergence is assured since the task completes within a finite number of cycles.

Suppose that the task starts at time $t = 0$ and completes by time $t = \mathbb{T}$. If it completes before that, the PE goes to a standby state until $t = \mathbb{T}$. We can reformulate Eq. (2) to include the energy consumed when the PE is in standby, by dividing the power into standby power $P(0)$ and active power $P_{act}(f) = P(f) - P(0)$. If we define the corresponding active energy-per-cycle $e_{act}(f) = {}^{P_{act}(f)}\!/_{f}$ then total energy can be rewritten as:

$$E_p = \sum_{w=1}^{\infty} e_{act}(f_w)\left[1 - cdf_W(w-1)\right] + \mathbb{T}P(0). \qquad (3)$$

The energy-per-cycle functions *e*(*f*) and e_{act}(*f*) are computed from the PE power function *P*(*f*). We used $P(f) = af^3 + b$ as the PE power function for our simulations in Section 4. This power function is justified as an approximation, since the scaling of the operating voltage is proportional to the frequency scaling, while dynamic power consumption is proportional to frequency times the voltage squared, hence the power is approximately proportional to the third power of frequency. In [13] we show that with correct choice of fitting parameters a and b, $P(f) = af^3 + b$ is empirically quite close to real power measurements, although the fitting parameters do not represent meaningful dynamic or static power coefficients. Alternatively, any other power function can be used with the formulations herein, whether in closed or numeric form.

To obtain the task execution expected time, T_p, we sum the delay-per-cycle times the probability that the task will still be executing at that cycle, in a manner similar to Eq. (2):

$$T_p = \sum_{w=1}^{\infty} \frac{1 - cdf_W(w-1)}{f_w}. \qquad (4)$$

2.2. DVS Application Model

In this study we model the application as an execution timeline comrpsising alternating serial and parallel phases [3], as shown in Figure 2(a), and focus on DVS policies for the parallel phases. This simple execution model is appropriate for numerous applications [14, 15] and programming models [16].

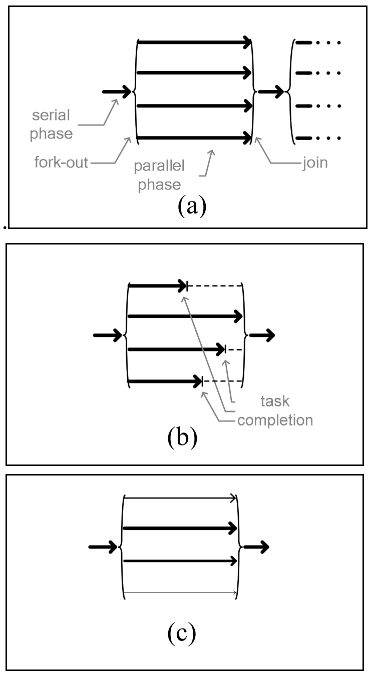


Figure 2: An execution timeline with (a) equal workloads, and (b) unequal workloads. Slack in (b) is utilized in (c) to save energy with no performance degradation. Reduced frequencies are indicated by thinner lines.

Tasks of equal workloads running on identical PEs at the same frequency-voltage work-point will achieve identical run-times, as shown in Figure 2(a). A typical example is multiple task instances performing the same work in parallel on different, equally sized data-sets (i.e., Data Decomposition [17]). In such symmetrical cases, there is no slack to utilize – although the work-point can be controlled, any energy saved will necessarily come at the expense of performance. A timeline of unequal workloads, as shown in Figure 2(b), exhibits slack and thus presents opportunity for energy saving *without* performance degradation. Unequal workloads can occur when tasks execute in parallel on different data sizes or when different types of tasks execute in parallel (i.e., Functional Decomposition [17]). A timeline such as Figure 2(b) can also occur in a heterogeneous-PE system [3, 11] where the PEs have unequal computation throughput. Combinations of the factors mentioned above are also possible.

Usually, the actual workload of a task is known only once the task completes. We must therefore estimate the workload of each task, and employ this estimation to assign (f,V) workpoints. The estimation is based on past performance of tasks of the same type.

2.3. CMP Problem Formulation

The total system energy $\mathbb{E}$ of a parallel phase is the sum of the energies of all PEs $p = 1..N$, while the combined execution time $\mathbb{T}$ of the parallel period is the maximum of task execution times:

$$\mathbb{E} = \sum_{p=1}^{N} E_p \quad , \quad \mathbb{T} = \max_{p=1..N} T_p \, . \tag{5}$$

$\mathbb{T}$ is determined by the completion time of the last task, i.e. the critical task running on the critical PE. Note that it is wasteful for any PE to complete before the combined execution time $\mathbb{T}$, since it potentially could have run slower and consumed less energy, finishing at time $\mathbb{T}$ and causing no performance degradation. We would like to utilize the slack of non-critical tasks in order to save energy. This notion is illustrated in Figure 2(c), where the reduced frequencies are indicated by thinner lines.

Alternative DVS policies are judged according to the balance that they manage to achieve between energy and performance. Different criteria may assign different weight to energy and performance. In this study we employ the criterion of minimal $\mathbb{E}\mathbb{T}^{\alpha}$, i.e., minimize the product of energy $\mathbb{E}$, and execution-time $\mathbb{T}$ to some power *α*. The exponent *ρ* is used to control the relative weight of execution time and energy. We prefer *α*=2, since $\mathbb{E}\mathbb{T}^2$ has the useful characteristic of *frequency invariance* [18]: $\mathbb{E}\mathbb{T}^2$ measures policy *quality* regardless of the actual frequencies used. A task's energy consumption is approximately proportional to the square of the voltage, which is in turn approximately proportional to the frequency, hence $E \propto f^2$. On the other hand, $T \propto \frac{1}{f}$ so $ET^{\rho} \propto f^{2-\alpha}$. In support of this we find in our simulations that using *α* < 2 gives inherent advantage to the *f*-min policy, while *α* > 2 favors the *f*-max policy. Further supporting this choice is the fact that any policy which always uses a single constant frequency (for example *f*-min and *f*-max, see Figure 9) achieves a constant $\mathbb{E}\mathbb{T}^2$ measure, regardless of frequency.

Optimization

A straightforward approach to finding an optimal policy is to solve the following minimization problem:

$$\begin{aligned} &\min \mathbb{E}\mathbb{T}^2 \\ &s.t.\ f_p(w) \in 0 \cup \left[f_{\min}, f_{\max} \right], \\ &\forall p = 1..N,\ \forall w = 1, 2..\infty \end{aligned} \tag{6}$$

Namely, minimize $\mathbb{E}\mathbb{T}^2$ while frequencies are constrained to an operating range, or 0 if the PE is idle.

Substituting $\mathbb{E}$ and $\mathbb{T}$ from Eqs. (5) into Eq. (6), and further substituting E_p and T_p (the individual PE energy and delay) from Eq. (3) and (4), we note that the ensuing optimization problem seems very hard, despite the simplifications already assumed in the model. Rigorous analysis of this minimization problem is beyond the scope of the present work. Moreover, we suggest that although this problem may be solvable, yielding some optimal frequency assignment per set of task distributions, the computation required is likely to be too great to justify implementation in any practical system. In a performance-driven energy-aware system, and assuming frequency assignment is integrated as part of the system's computational requirements in real-time, we must weigh the improvement that a computation offers versus the cost of the computation itself, both in energy and performance. This observation rationalizes the approach of searching for simple lightweight heuristic DVS policies.

3. DVS Policies

In this section we describe a group of relatively simple frequency assignment policies for a CMP. Since all policies share the same outline and differ only in certain details, we first describe the common outline and then specify the differences for each policy.

3.1. Common Outline for All Policies

At the *t=0* fork-point of the parallel phase in the timeline, all policies perform the same steps:

(1) Find the PE (task) estimated to have the most remaining work, which is referred to as the *estimated-critical PE* (ECP).
(2) Set a joint-target-time (JTT) for all PEs to complete their tasks, and assign PE frequencies (and voltages).
(3) Run until the ECP finishes (or time interval elapsed).
(4) Update workload estimations.
(5) Loop back to Step (1) above.

In Step (1): We compute the expected value of the (remaining) estimated work for all PEs (currently running tasks):

$$\hat{W}_{p,rem} = E[W_p \mid W_{p,comp}] \tag{7}$$

Note that at time *t=0*, the completed work $W_{p,comp} = 0$. From this estimation we find the ECP, the PE for which the estimated work is maximal:

$$\begin{aligned}\hat{W}_{ECP,rem} &= \max_{p=1..N}\{\hat{W}_{p,rem}\} \\ ECP &= \arg\max_{p=1..N}\{\hat{W}_{p,rem}\}\end{aligned}. \tag{8}$$

In Step (2): The ECP is assigned $f_{\max}$. Intuitive reasoning for this is that $\mathbb{E}\mathbb{T}^2$ is frequency invariant if we ignore $P(0)$, which means that results depend solely on the amount of utilized slack, while actual PE frequencies are insignificant. However, $P(0)$ cannot be ignored. Therefore, assigning $f_{\max}$ to the ECP, i.e., running as fast as possible, will clearly minimize standby energy consumption. Optimality of assigning $f_{\max}$ to the critical PE is proven in Section 3.2 for a case of known workloads.

To utilize the available time-slack, we want all the other (non-critical) PEs to complete together with the critical PE. We therefore set a *joint-target-time* for all PEs, which is the expected completion time of the ECP:

$$JTT = \frac{\widehat{W}_{ECP,rem}}{f_{\max}}, \tag{9}$$

where $\widehat{W}_{ECP,rem}$ is the estimated (remaining) work of the ECP. Frequency assignment of the other (non-critical) PEs is described per each policy below.

In Step (3): All PEs run at their assigned frequencies until the ECP completes. (In case of the *Interval* policy described in 3.4, re-estimation and re-assignment are performed also at intermediate fixed time intervals.) The ECP is expected to complete last by definition, and this will typically be the case. However since workloads are statistical it is possible that the ECP will complete before other PEs.

In Step (4): If the ECP completes (or a time-interval elapses) while other PEs still have remaining work, re-estimation is performed, taking into account the work done by each PE so far. The cycle is repeated until all PEs have completed their work.

Figure 3 shows a flowchart of the common outline. The specific variations of the policies are described next.

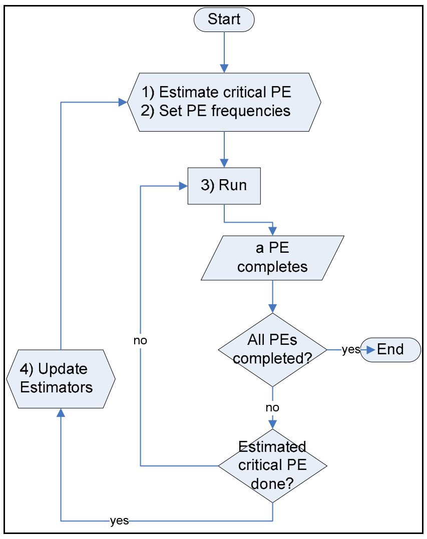


Figure 3: Flowchart of common outline for all polices

3.2. The "Oracle" Policy

The Oracle policy is a non-causal, hypothetical policy which assumes future knowledge of the workloads. When simulating the Oracle policy we still generate workloads statistically, however we assume that they are known in advance, before run time. We use the Oracle policy results as a lower bound for comparing to causal, implementable policies in which the exact workloads are not known in advance.

Given that the workloads are known, we calculate the optimal frequencies f_p to assign to each PE $p = 1..N$. In particular we show that $f_{\max}$ is optimal for the critical PE. We denote the known task workloads on each PE W_p, and define T to be the joint execution time for all PEs. Because the workload of each PE is known and the execution time is set, energy can be minimized by running at a constant frequency that is just fast enough to finish the task by its deadline. This is a well established result which is due to the convexity of *e*(*f*), the energy-per-cycle [19]. Therefore we assign constant frequencies $f_p = \frac{W_p}{T}$ to each PE. We initially assume that frequencies are not restricted to any range, and later incorporate frequency bounds. All PEs complete exactly at *T*, so that full slack utilization is attained.

Assuming that there exists a T as above and it minimizes $\mathbb{ET}^2$, that T can be found by differentiating $\mathbb{ET}^2$ w.r.t. T and equating the derivative to 0:

$$\mathbb{ET}^2 = T^2\sum_{p=1}^{N} E_p = T^2\left[\sum_{p=1}^{N} W_p e_{act}\left(\frac{W_p}{T}\right) + NP(0)T\right] \Rightarrow$$

$$\frac{d}{dT}\left(\mathbb{ET}^2\right) = 2T\left[\sum_{p=1}^{N} W_p e_{act}\left(\frac{W_p}{T}\right) + NP(0)T\right] +$$

$$+T^2\left[\sum_{p=1}^{N} W_p e'_{act}\left(\frac{W_p}{T}\right)\left(-\frac{W_p}{T^2}\right) + NP(0)\right] =$$

$$= 2T\sum_{p=1}^{N} W_p e_{act}\left(\frac{W_p}{T}\right) - \sum_{p=1}^{N} W_p^2 e'_{act}\left(\frac{W_p}{T}\right) + 3NP(0)T^2$$

Now if we assume the power model described in Section 2.1, then $e_{act}(x) = ax^2, e'_{act}(x) = 2ax$ and substituting in (10) we find that the two sums cancel out:

$$\frac{d}{dT}\left(\mathbb{ET}^2\right) = 2T\sum_{p=1}^{N} W_p a\left(\frac{W_p}{T}\right)^2 - \sum_{p=1}^{N} W_p^2 \bullet 2a\left(\frac{W_p}{T}\right) + 3NP(0)T^2$$
$$= 3NP(0)T^2 \quad . \qquad (11)$$

Note that if P(0)=0 then this means the criterion is truly frequency invariant, since the measure does not depend on the operating frequencies. Otherwise, *T=0* is required in order to minimize the criterion, but since that is not possible, the shortest execution time we can set is $T = \max\left(W_p\right)/f_{\max}$, so all frequencies are correspondingly set as follows:

$$f_p = map\left(\frac{W_p}{T}\right) =$$
$$map\left(\frac{W_p}{\max\limits_p(W_p)} f_{\max}\right),\ p = 1..N \quad . \qquad (12)$$

No re-estimation is necessary since all task workloads are known. The *map*() function maps the frequency in (12) to a feasible frequency. In the simple case of a continuous frequency range, this merely means trimming out-of-range values to $f_{\min}$ and $f_{\max}$ correspondingly. (Also note that in the particular case of Eq. (12) the inner result is already bounded by $f_{\max}$.) All PEs complete precisely at T, with the exception of non-critical PEs with a small workload which run at $f_{\min}$ according to Eq. (12), finishing before T.

3.3. The Constant Policy

The Constant policy is a simple policy in which a constant frequency is assigned to each PE. After calculating the JTT and assigning the ECP to run at the maximum frequency $f_{\max}$, we set non-critical PE frequencies with an aim to complete at the joint-target-time:

$$f_p = map(\frac{\widehat{W}_{p,rem} + \lambda \widehat{\sigma}_{p,rem}}{JTT}) = map(\frac{\widehat{W}_{p,rem} + \lambda \widehat{\sigma}_{p,rem}}{\widehat{W}_{ECP,rem}} f_{\max}) \ , \ p = 1..N \qquad (13)$$

where $\widehat{W}_{p,rem}$ is the remaining work estimation of PE p, $\widehat{\sigma}_{p,rem}$ is the standard deviation of $W_{p,rem}$, and $\lambda \geq 0$ is the bias parameter. At time t=0 no work has been done so the remaining work is the total work. The $map()$ function maps the result of Eq. (13) to a feasible PE frequency as described above.

For $\lambda = 0$, f_p is set so that PE p completes $\widehat{W}_{p,rem}$ work during the time it takes the ECP to complete $\widehat{W}_{ECP,rem}$ work. However since delay outweighs energy for the $\mathbb{ET}^2$ criterion, we can achieve better results by setting the bias parameter $\lambda > 0$, as described below in Section 4.1.

Note that the critical PE frequency can also be formulated using Eq. (13) if we assume $\lambda \geq 0$ and substitute $\widehat{W}_{p,rem}$ with $\widehat{W}_{ECP,rem}$.

Early Completion of the Estimated-critical PE

If the ECP completes while other PEs still have remaining work (step (3) in the outline), then the assumptions by which frequencies were assigned in step (2) no longer hold. In this case we update the estimations $\widehat{W}_{p,rem}$ $\widehat{\sigma}_{p,rem}$ to reflect the work done so far, $W_{p,comp}$, and repeat steps (1) and (2): set a new joint-target-time and a new ECP, and assign new PE frequencies. We continue steps (1) to (4) repeatedly until all PEs have completed.

Figure 4 shows an example of applying the *Constant* policy in a 3-PE system. Figure 4(a) shows the workload distributions, and then two scenarios are illustrated. In Figure 4(b), the workloads are such that the ECP (red, dotted) completes last. This is the expected, common scenario. In Figure 4(c), the ECP unexpectedly finishes first, causing re-estimation; a new ECP is selected (green, dashed) and work-points are reassigned.

If we regard the complexity of one (*f,V*) work point assignment (including the preceding remaining workload estimation) as $O(1)$, then the complexity of the Constant policy can be approximately regarded as $O(N)$, N being the number of PEs in the system. This approximation ignores the occasional work-point re-estimation that is required when the ECP completes before other PEs. However, this is justified since the probability of this scenario is (a) generally small, (b) distribution dependent, (c) very hard to calculate and incorporate into complexity estimations, and (d) common among all described policies, so it generally shouldn't affect comparing them.

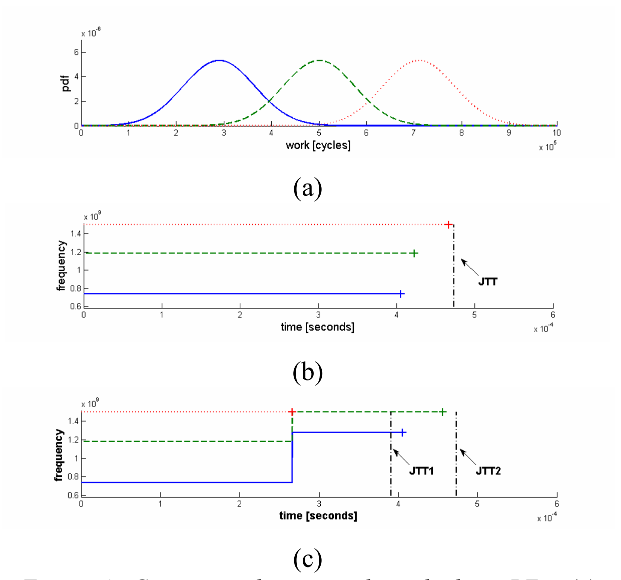


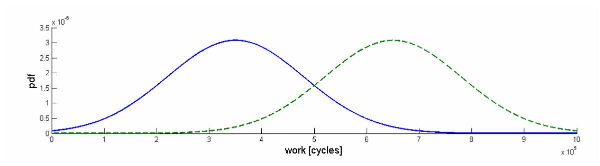
Figure 4: Constant policy example with three PEs: (a) workload distributions, (b) the ECP (red, dotted) finishes last as expected, (c) the ECP finishes first, re-estimation is performed.

3.4. *The Interval Policy*

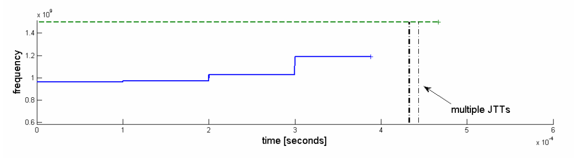
The Interval policy is an enhancement of the Constant policy described above. The Interval policy assigns constant frequencies in the same way as the Constant policy. But in the Interval policy, the critical PE is re-chosen and frequencies are reassigned at intermediate fixed time intervals. For each time interval, estimated *remaining* workloads are used to choose the critical PE and frequencies for the next interval. Re-estimating remaining workloads and re-assigning work-points following the re-estimation may offer a significant advantage: the difference between the estimated remaining workloads at time *t*=0, compared to the estimations at a later time when all PEs have done a certain amount of work, may be substantial. Interval can be viewed as a refinement of the Constant policy: the shorter the time interval, the more accuracy can be achieved.

Note that it is possible to calculate frequency assignments not just for the current interval but for *future* intervals as well, since all the needed information is available at time *t=0*. The only information that is not available at time *t=0* is actual task workloads. So at any given time we can calculate frequencies to be assigned for future intervals, and these calculations will be valid until the time the ECP completes. Therefore, although it is easier to understand Interval as a process of recalculating frequencies at each interval, in practice Interval calculates frequencies at time t=0, and recalculates only when an ECP completes.

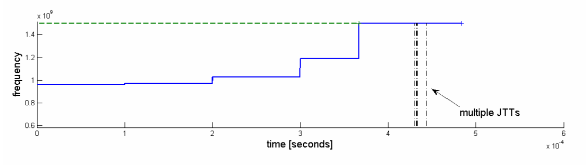
A bias parameter λ exists also for the Interval policy and allows tuning in the same way as for the Constant policy. Figure 5 shows an example of the Interval policy with 2 PEs.



(a)



(b)

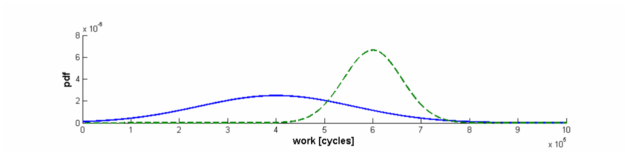


(c)

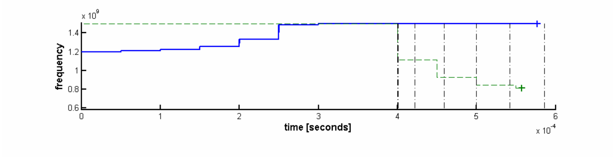
Figure 5 : Interval policy example with two PEs: (a) workload distributions, (b) the ECP (green, dashed) finishes last as expected, (c) the ECP finishes first.

In typical examples, the frequency of non-critical PEs increases with time, as shown in Figure 5. This behavior is similar to the behavior of PACE [20], as explained in the following section. However, decreasing frequencies can occur, since PE frequencies are a function of the estimated remaining workload relative to the estimated remaining workload of the ECP at each interval. Notably, when there is a substantial difference in both the mean and variance of workload distributions, both increasing and decreasing frequencies can occur, as shown in Figure 6 below.

Note that in Figure 6, there is an ECP switch at a certain time during the run. The ECP (green, dashed) is initially estimated to have more remaining work, and thus it is designated as the ECP and assigned $f_{\max}$. However as time progresses the ECP does more work relative to the other (blue, solid) PE, until a time where the other PE's estimated remaining work surpasses that of the ECP, thus the other (blue, solid) PE is designated the new ECP.



(a)



(b)

Figure 6: Another interval policy example, with 2 PEs having substantial difference in both the mean and variance of their workload distributions, (a). Both increasing and decreasing frequencies are observed, (b). The PE marked in dashed green is initially the ECP, but after some time the PE marked in blue becomes the new ECP.

Since the complexity of the Constant policy is $O(N)$, the complexity of the Interval policy is $O(kN)$, where k is the number of time intervals at which re-estimation occurs.

3.5. The Multi-PACE Policy

Energy-performance tradeoff in a single standalone PE has been studied extensively [9, 21-24]. The Multi-PACE policy presented in this section is an attempt to generalize from the well-known single-processor approach PACE [20] onto CMP systems.

PACE Scheduling for a Single Processor

Consider a task of given workload probability distribution $pdf_W(w)$, running on a PE with a continuous frequency range $f \in [f_{\min,} f_{\max}]$, with some soft deadline D, i.e. a deadline that is required to be attained only in a certain fraction of cases, not necessarily all the time. We represent this by the probability PMD (Probability of Meeting the Deadline). Given $pdf_W(w)$ of a task, a deadline D, and PMD, we can find the maximum workload w_{PMD} for which a task will meet its deadline.

The PACE scheme is an analytically optimal method for minimizing the expected energy subject to probabilistically meeting the deadline [20]. Given the above inputs, PACE computes the optimal frequency schedule per cycle:

$$f(w) = PACE([f_{\min,}f_{\max}], pdf_W(w), D, PMD), . \quad w \in [0, w_{PMD}] \tag{14}$$

Conversion from $f(w)$, which expresses the frequency as a function of work cycles done, to a more intuitive $f(t)$ time function, is straightforward.

As mentioned above in Section 3.2, running at a constant frequency is optimal in case workloads are known, but not so when workloads are unknown. PACE stands for Processor Acceleration to Conserve Energy, reflecting the fact that the optimal frequency schedule is an increasing function when workloads are unknown. An intuitive explanation for this is that a task workload may be small or large, so it is worthwhile to run slowly at first. If the workload is small then the task can easily complete by the deadline, saving energy by running at a low frequency. As the deadline approaches, if the task has not yet completed the frequency is gradually increased in order to assure meeting the deadline with probability PMD.

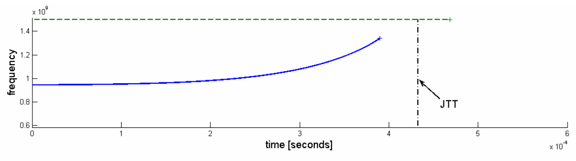
Multi-PACE (*f,V*) Work-point Assignment

We introduce Multi-PACE, which utilizes PACE to form a DVS policy for a CMP, as follows. After setting the ECP to run at $f_{\max}$ and setting the JTT as in Eq. (9), non-critical PE frequencies are set according to PACE with the JTT as a deadline:

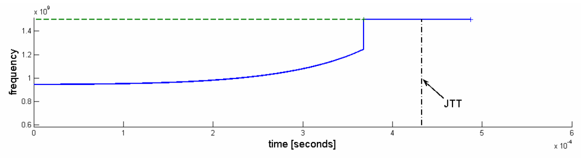
$$f_p(w) = PACE([f_{\min,}f_{\max}], pdf_{W_{p,rem}}(w), JTT, PMD), . \quad w \in [0, w_{PMD}] \tag{15}$$

To clarify, note that JTT is *not* an application deadline. Rather, it is computed following Eq. (9). We use the PACE deadline mechanism to synchronize completion times between PEs.

PACE does not specifically define which frequency to use for the post-deadline part (i.e., cycles greater than w_{PMD}). Multi-PACE runs at $f_{\max}$ during the cycles subsequent to w_{PMD} in an attempt to minimize delay past the JTT. Figure 7 shows an example of applying Multi-PACE, using the same workload distributions as in Figure 4(a).



(a)



(b)

Figure 7: Multi-PACE example with two PEs, workload distributions as in Workload distributions are as in Figure 4. (a) the ECP (green, dashed) finishes last as expected; (b) the ECP finishes first.

The value of PMD has a significant effect on the overall results. If PMD is too high, multi-PACE sets overly aggressive frequency schedules, resulting in excessive energy consumption. On the other hand, choosing PMD too low increases the probability of missing the JTT, causing increased overall execution time. In Section 4.1 we experiment with different PMD values for Multi-PACE. The use of PMD is similar to the use of the λ bias parameter for the Constant and Interval policies.

Multi-PACE requires PEs to have a continuous frequency range and to be able to change frequency every cycle, but this is not practical for reasons previously discussed. Practical methods of implementing PACE, which can apply to Multi-PACE as well, are described in [20, 23].

As explained above, the complexities of the Constant and Interval policies are $O(N)$ and $O(kN)$ respectively, where N is the number of PEs and k is the number of intervals. In principle, frequencies are recalculated in Multi-PACE *for each cycle*. In a practical system, however, the workload distributions would be built by collecting data into a histogram, and their granularity would therefore be according to the number of histogram bins, denoted by B. Thus the number of frequency changes in multi-PACE is in practice proportional to B, and thus the complexity of the multi-PACE policy is $O(BN)$ [20].

4. Simulations and Results

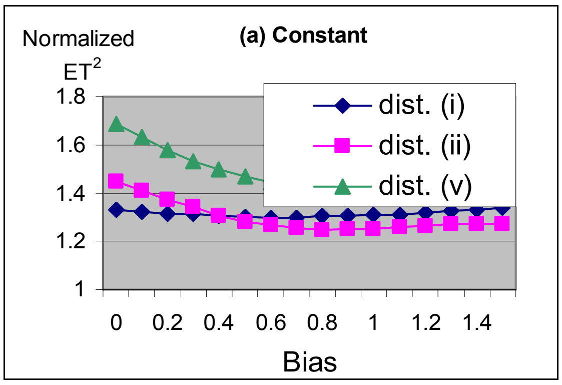
In this section we present simulation results of the proposed DVS policies. We used a few sets of synthetic probability distributions to represent the task workloads, as shown above in Figure 1.

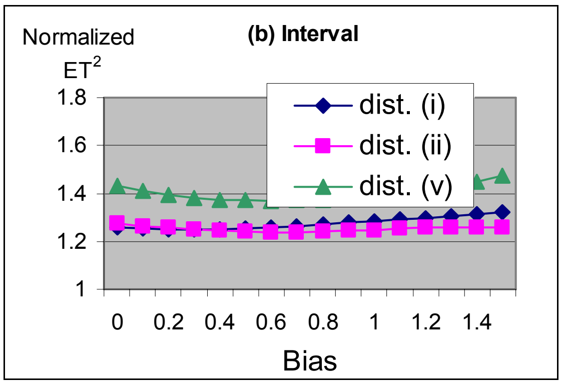
We simulated a system with six identical PEs, each with a continuous frequency range of 0.32GHz to 1.5GHz. Energy was calculated using the power model of $P(f) = af^3 + b$ with $a = 0.8\left[\frac{Watt}{GHz^3}\right]$, $b = 0.2\left[Watt\right]$ as fitting parameters, as described in section 2.1.

4.1. Choosing Bias Parameters

Prior to comparing the policies, we consider the issue of selecting bias parameters: λ for Constant and Interval policies, and PMD for Multi-PACE. As previously mentioned, setting PE work-points so tasks complete their estimated workloads at the JTT is sub-optimal. Delay outweighs energy for the $\mathbb{ET}^2$ criterion, therefore missing the JTT by a certain time margin incurs a greater penalty than finishing before the JTT by that same time margin. Better results can be achieved by setting the bias parameter $\lambda > 0$ in Eq. (13), thereby running faster. A similar effect can be achieved for the Multi-PACE policy by tweaking the PMD parameter in Eq. (15).

Figure 8 shows policy results, in terms of $\mathbb{ET}^2$ values, normalized to the Oracle policy, for distributions (i), (ii), and (v) of Figure 1, using different bias values λ and PMD. As can be seen in Figure 8, each case shows a certain optimal choice of the bias parameters λ and PMD. However, the results are not very sensitive to small variations, and thus it is reasonable to use the same bias parameter (λ or PMD) for all distributions. Empirically, a good choice lies in the range of 0.5-0.8. The effect of λ on $\mathbb{ET}^2$ is less accentuated in Interval than in Constant, since Interval performs periodic re-estimations of $W_{rem}^{(i)}$ and $\sigma_{rem}^{(i)}$.





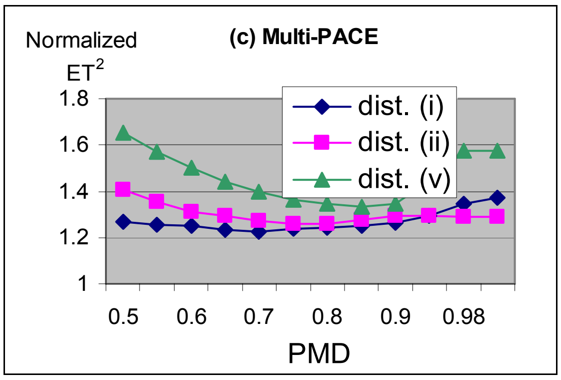


Figure 8: Comparison of bias values for policies (a) Constant, (b) Interval (50ms), and (c) Multi-PACE policies. Distributions (i), (ii), and (v) (see Figure 1) are shown.

4.2. DVS Policy Comparison

We simulated each of the workload distributions shown in Figure 1 using Oracle, Constant, Multi-PACE, and Interval policies. For Interval, we used intervals lengths of 50, 100, 150 and 200 milliseconds which correspond to roughly 16, 8, 4, and 2 intervals throughout the simulated execution time. We chose bias parameters $\lambda = 0.8$ for Constant, $\lambda = 0.5$ for Interval and PMD=80% for Multi-PACE, following the conclusions of Section 4.1.

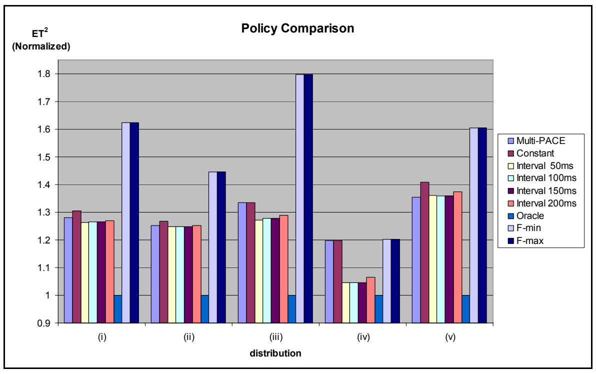


Figure 9: ET2 compared across all policies

For each distribution, Figure 9 shows $\mathbb{ET}^2$ compared across all policies, normalized as above to the results of Oracle, which we regard as a lower bound. Additionally, the results show that *f*-min and *f*-max policies reach the same results in $\mathbb{ET}^2$ for all distributions. This follows from the approximate frequency invariance of $\mathbb{ET}^2$, and likewise holds for any other policy that always uses a single constant frequency. Thus we expect our policies to achieve results that are in between Oracle and *f*-max/*f*-min, i.e., better than running at an arbitrary constant frequency (which we regard as 0% improvement), but worse than the optimal policy in which exact workloads are known in advance (100%). As can be seen, this is indeed the case.

The Interval policy usually achieves the best results while Constant, the simplest policy, usually achieves the worst results. However, the difference between the policies is generally quite small, with no more than 4-13% difference between the best and worst policies (except for example *iv*, which is discussed next.)

Despite the above similarities, distribution (*iv*) of Figure 1 is an example where the Interval policy stands out. As can be seen in Figure 1(*iv*), two of the six task workloads of example (*iv*) have a bimodal distribution ($1\bullet10^5$ or $10\bullet10^5$ cycles) while the other four are known ($7.5\bullet10^5$ cycles). At time *t=0*, the estimated critical workload is $7.5\bullet10^5$ cycles. However, once the bimodal-distributed tasks complete 10^5 cycles work, the estimation may change to 10^6 cycles, enabling the known-workload tasks to run slower, saving considerable energy. Interval performs much better than all other policies since it is the only policy that re-estimates the critical workload.

With further regard to the Interval policy, we note that increasing the interval resolution (increasing the number of re-estimations) provides only minor, insignificant improvement. A few re-estimations over a relatively large period of time can drastically change the outcome, as demonstrated by example (*iv*), while additional re-estimations have only a marginal effect.

Multi-PACE on the whole achieves better results than Constant and worse than Interval in the simulated examples. Multi-PACE is more dependent on correct estimation of the critical task than the other policies, and therefore produces slightly better results in cases where there is little uncertainty regarding the critical task (*v*), compared to cases where the uncertainty is greater (*iii*).

Computational Complexity

In Section 3, we concluded that the complexities of Constant, Interval, and Multi-PACE are $O(N)$, $O(kN)$, and $O(BN)$, respectively; where N is the number of PEs, k the number of intervals, and B the number of workload histogram bins. Since increasing the number of intervals for the Interval policy beyond a small number provides only marginal improvement (note the marginal improvement when using shorter intervals in Figure 9), it is reasonable to assume small values for k. On the other hand, Multi-PACE performs no re-estimation, so it needs a considerable large number of bins B [20]. B does not necessarily need to be of cycle granularity, but will be several orders larger than k, i.e., $k<<B$. Following this reasoning, we conclude that the relative complexity of the policies is $O(N) < O(kN) << O(BN)$.

5. Summary and Conclusions

In this work, we started by formulating an energy-performance tradeoff optimization problem of an application running on a CMP. We noted the complexity of the problem, which makes it virtually impractical for implementation.

As an alternative to direct optimization, we described several simple heuristic DVS policies for energy-performance tradeoff. These policies try to utilize available time-slack in order to save energy in a performance-aware manner. The frequency-invariant $\mathbb{ET}^2$ criterion was employed for comparing the policies. The policies described were: Constant, a policy that tries to estimate the best constant frequency to assign to each PE; Interval, which works in a manner similar to Constant but reassigns new frequencies at fixed time intervals, and Multi-PACE, applying PACE, an optimal scheme for a single-core system with a deadline requirement, for use in a CMP.

We compared these policies using various distributions, and presented several examples. We showed that, except for some isolated cases, all policies reach comparable results. Increasing the number of re-estimations (using Interval) improves results compared to estimating merely once at the beginning (using Constant). However, the marginal return sharply diminishes with the number of re-estimations. Multi-PACE produces results that are anywhere between Interval and Constant, occasionally appearing at the top or bottom of the results list, depending on the distribution.

We analyzed the policy complexities, and showed that Constant is the least complex, followed by Interval, while Multi-PACE has the highest complexity, significantly higher than Constant and Interval. Since the results are usually quite close for all policies, we conclude that the least complex policy, Constant, is usually preferred. In individual cases, such as distribution (*iv*) shown in Figure 1, there is justification for using Interval. Based on these findings, a scheme could be contemplated whereby the number of intervals is chosen dynamically based on certain characteristics of the distribution, or alternatively, start with a default number of intervals, and assess the result over time to determine if the number of intervals can be decreased. Multi-PACE generally does not achieve better results than any of the other two, and has a very high complexity, so it is not preferred.

Frequency-voltage transitions, which are not considered in this work, may degrade the results since each transition is accompanied by performance and energy penalties [6]. When the cost of transitions is considered, simple policies such as Constant become even more attractive because they use fewer transitions.

The following issues are left for future research:

1. Study of more complex task-graphs.
2. Discret (f-V) workpoint sets.
3. With regard to 2), the interval policy may be enhanced to consider re-estimation at flexible times. Such an interval policy would determine *when* to jump to an adjacent discrete workpoint, rather than directly calculating a new workpoint at an arbitrary time.
4. Test cases based on real application traces.
5. Applications may data assisting in estimation of their own remaining work, which can improve the accuracy of remaining workload estimations.
6. Real-time applications, which need to achieve a periodic deadline, can be modeled by replacing the execution time $\mathbb{T}$ in the criterion with a relative $\mathbb{T}-D$ measure which results in penalty only to the extent by which the application missed its deadline.

References

[1] F. Pollack, "New Microarchitecture Challenges in the Coming Generations of CMOS Process Technologies," in *Micro 32*, 1999. http://www.intel.com/research/mrl/Library/micro32Keynote.pdf

[2] E. Grochowski, R. Ronen, J. Shen, and H. Wang, "Best of Both Latency and Throughput," in *Proceedings of the IEEE International Conference on Computer Design (ICCD'04) - Volume 00*: IEEE Computer Society, 2004.

[3] T. Y. Morad, U. C. Weiser, A. Kolodny, M. Valero, and E. Ayguade, "Performance, Power Efficiency and Scalability of Asymmetric Cluster Chip Multiprocessors," *IEEE Computer Architecture Letters*, vol. 5, 2006.

[4] "Intel Enhanced SpeedStep(R) Technology" http://www.intel.com/support/processors/mobile/pentium4/sb/CS-007499.htm http://www.intel.com/support/processors/mobile/pm/sb/CS-007981.htm

[5] T. Pering, T. Burd, and R. Brodersen, "The simulation and evaluation of dynamic voltage scaling algorithms," in *Proceedings of the 1998 international symposium on low power electronics and design*. Monterey, California, United States, 1998, pp. 76-81.

[6] S. M. Martin, K. Flautner, T. Mudge, and D. Blaauw, "Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads," in *Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*. San Jose, California, 2002.

[7] K. Flautner, S. Reinhardt, and T. Mudge, "Automatic performance setting for dynamic voltage scaling," *Wireless Networks*, vol. 8, pp. 507-520, 2002.

[8] S. Yaldiz, A. Demir, S. Tasiran, P. Ienne, and Y. Leblebici, "Characterizing and exploiting task load variability and correlation for energy management in multi core systems," in *3rd Workshop on Embedded Systems for Real-Time Multimedia, 2005*, 2005, pp. 135-140.

[9] Y. Zhang, X. S. Hu, and D. Z. Chen, "Task scheduling and voltage selection for energy minimization," in *Proceedings of the 39th conference on Design automation*. New Orleans, Louisiana, USA, 2002, pp. 183-188.

[10] A. Andrei, M. Schmitz, P. Eles, Z. Peng, and B. M. Al-Hashimi, "Overhead-conscious voltage selection for dynamic and leakage energy reduction of time-constrained systems," *Computers and Digital Techniques, IEE Proceedings-*, vol. 152, pp. 28-38, 2005.

[11] R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, and K. I. Farkas, "Single-ISA Heterogeneous Multi-Core Architectures for Multithreaded Workload Performance," in *Proceedings of the 31st annual international symposium on Computer architecture*. Munchen, Germany: IEEE Computer Society, 2004.

[12] S. Ghiasi, T. Keller, and F. Rawson, "Scheduling for heterogeneous processors in server systems," in *Proceedings of the 2nd conference on Computing frontiers*. Ischia, Italy, 2005, pp. 199-210.

[13] A. Elyada, "Low Complexity Policies for Energy-Performance Tradeoff in Chip-Multi-Processors," in *Electrical Engineering*. Haifa: Techion, Israel Institute of Technology, 2007.

[14] M. L. Crow and M. Ilic, "The parallel implementation of the waveform relaxation method for transient stability simulations," *IEEE Trans. on Power Systems*, vol. 5, pp. 922-932, Aug 1990.

[15] R. A. Saleh, K. A. Gallivan, M.-C. Chang, I. N. Hajj, D. Smart, and T. N. Trick, "Parallel circuit simulation on supercomputers," *Proceedings of the IEEE*, vol. 77, pp. 1915-1931, Dec 1989.

[16] L. Dagum and R. Menon, "OpenMP: an industry standard API for shared-memory programming," in *Computational Science and Engineering, IEEE*, vol. 5, 1998, pp. 46-55.

[17] J. Ross, "Media Applications Shine with Pipelined Data Domain Decomposition Threading" http://www.intel.com/cd/ids/developer/asmo-na/eng/dc/digitalmedia/success/52517.htm

[18] A. J. Martin, M. Nystroem, and P. Penzes, "ET^2: A metric for time and energy efficiency of computation," in *Power Aware Computing*, *Series in Computer Science*, R. Graybill and R. Melhem, Eds. Norwell, MA: Kluwer Academic Publishers, 2002, pp. 293-315.

[19] J. Blazewicz, K. H. Ecker, E. Pesch, G. Schmidt, and J. Weglarz, *Scheduling Computer and Manufacturing Processes*. Berlin, Germany: Springer-Verlag, 1996.

[20] J. R. Lorch and A. J. Smith, "PACE: a new approach to dynamic voltage scaling," *IEEE Transactions on Computers*, vol. 53, pp. 856-869, 2004.

[21] R. Jejurikar, C. Periera, and R. Gupta, "Leakage aware dynamic voltage scaling for real-time embedded systems " *Proceedings of the 41st annual conference on Design automation* pp. 275-280, 2004.

[22] R. Xu, D. Mosse, and R. Melhem, "Minimizing expected energy in real-time embedded systems," in *Proceedings of the 5th ACM international conference on Embedded software*. Jersey City, NJ, USA, 2005, pp. 251-254.

[23] R. Xu, C. Xi, R. Melhem, and D. Moss, "Practical PACE for embedded systems," in *Proceedings of the 4th ACM international conference on Embedded software*. Pisa, Italy, 2004, pp. 54-63.

[24] D. Zhu, R. Melhem, and B. Childers, "Scheduling with Dynamic Voltage/Speed Adjustment Using Slack Reclamation in Multi-Processor Real-Time Systems," *IEEE Trans. on Parallel and Distributed Systems*, 2003.

[25] L.-F. Leung, C.-Y. Tsui, and W.-H. Ki, "Minimizing energy consumption of multiple-processors-core systems with simultaneous task allocation, scheduling and voltage assignment," in *Proceedings of the 2004 conference on Asia South Pacific design automation*. Yokohama, Japan, 2004, pp. 647-652.