Enabling Improved Power Management in Multicore Processors through Clustered DVFS

Tejaswini Kolpe†, Antonia Zhai‡, and Sachin S. Sapatnekar†
† Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN, USA
‡ Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN, USA

***Abstract*—In recent years, chip multiprocessors (CMP) have emerged as a solution for high-speed computing demands. However, power dissipation in CMPs can be high if numerous cores are simultaneously active. Dynamic voltage and frequency scaling (DVFS) is widely used to reduce the active power, but its effectiveness and cost depends on the granularity at which it is applied. Per-core DVFS allows the greatest flexibility in controlling power, but incurs the expense of an unrealistically large number of on-chip voltage regulators. Per-chip DVFS, where all cores are controlled by a single regulator overcomes this problem at the expense of greatly reduced flexibility. This work considers the problem of building an intermediate solution, clustering the cores of a multicore processor into DVFS domains and implementing DVFS on a per-cluster basis. Based on a typical workload, we propose a scheme to find similarity among the cores and cluster them based on this similarity. We also provide an algorithm to implement DVFS for the clusters, and evaluate the effectiveness of per-cluster DVFS in power reduction.**

I. Introduction

In recent years, there has been a growing demand for high-performance computing systems capable of performing a multitude of tasks. Meeting these demands using a single superscalar processor that uses instruction level parallelism (ILP) or simultaneous multithreading (SMT) has offered only partial relief, and on most applications, these schemes have eventually hit barriers associated with limited ILP, limited memory bandwidth, total power budgets, or some combination of these factors.

Multicore processors, or chip multiprocessors (CMP), are seeing increasing adoption as they provide a pathway to achieving high performance under power bounds [1]. These consist of several processor cores on a single die, each equipped with one or more levels of a private cache. Each core is relatively simpler and easier to design and validate than a single large SMT processor. Multiple applications can be run independently on each core of a CMP, or a single application can be split into several parallel threads and can be executed on the cores simultaneously, and a high throughput can be achieved without increasing the clock rate. The reduced complexity and smaller sizes of the cores of a CMP eliminate the necessity for long interconnects, thus eliminating significant performance bottlenecks [2]. However, multicore systems also have the potential for large area and power overheads as several processor cores operate simultaneously. Thus, a major task in working with multicore architectures is in controlling the power dissipation.

This research was supported in part by the NSF under award ECCS-0903427 and the SRC under contract 2009-TJ-1990.

A significant observation in this regard is that instructions in various cores typically do not execute at a constant rate. For example, cache misses result in memory accesses that are much slower than on-chip operations and are of the order of few hundreds of cycles. The instruction throughput is low during such periods of low activity, and hence, if the processor is operated at a high frequency during these times, the corresponding switching transitions are essentially wasted. To reduce the total power during such periods, the processor frequency and V_{dd} value can each be linearly reduced on the fly to yield a cubic reduction in the dynamic ($CV_{dd}^2 f$) power. The process of dynamically altering the supply voltage and operating frequency is commonly referred to as dynamic voltage and frequency scaling (DVFS), and the corresponding schedule is referred to as the DVFS schedule.

A key element of DVFS is the use of one or more voltage regulators that deliver power to a circuit from an energy source. On-chip regulators have the advantage of providing fast switching to different voltage levels as compared to off-chip voltage regulators: specifically, the voltage transition times are of the order of tens of nanoseconds for on-chip voltage regulators, and of the order of tens of microseconds for off-chip regulators [3].

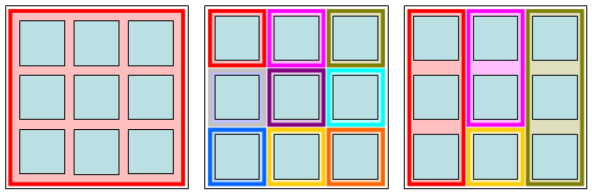

(a) Per-chip DVFS (b) Per-core DVFS (c) Clustered DVFS

Fig. 1. DVFS domains in a CMP

In principle, DVFS for a CMP can be performed at various levels of granularity:

- *Per-chip DVFS*, as shown in Figure 1(a), uses the same power delivery network to reach every core, and consequently, binds each core to the same DVFS schedule.
- *Per-core DVFS*, illustrated in Figure 1(b), uses a separate voltage regulator for each core and therefore allows every core to have an independent DVFS schedule.
- *Cluster-level DVFS*, shown in Figure 1(c), uses multiple of on-chip regulators drive a set of DVFS domains, or clusters, so that one or more cores are associated with each cluster.

The single voltage regulator required by per-chip DVFS may be located on-chip or off-chip. Per-core DVFS requires one regulator per core: due to the scales involved, it is essential for all of these regulators to be located on-chip. However, on-chip regulators incur significant area and power overhead by introducing large inductors and capacitors; moreover, it is difficult to build inductors with sufficient Q factors to support a large number of on-chip regulators. Therefore, per-core DVFS is practically difficult beyond 4 or at most 8 cores. Additionally, it does not yield commensurate benefits that justify its overhead [4]; on the other hand, per-chip DVFS is too constrained by the worst case over all cores, and cannot leverage the full power of DVFS. Cluster-level DVFS represents an intermediate point between the extremes of per-chip and per-core DVFS, and is a reasonable design scheme.

In this work, we address the problem of power dissipation in a homogeneous CMP consisting of identical cores at the design stage, by grouping the cores into a specified number of clusters such that the performance and power dissipation of the clusters is optimized. There has been little prior work in the area of clustered DVFS and researchers have focused primarily on per-core [5] or per-chip [6] DVFS for CMPs. The clustering of cores in a 16-core CMP and the implementation of DVFS on a per-cluster basis has been analyzed in [4], but the clustering is not based on any similarity metrics or typical workloads. Two clustering methods are used: cores with consecutive indices are grouped together and cores with the same index mod 4 are clustered together: in each case, the assignment of cores to domains is uniform. Our proposed method for core clustering, on the other hand, is based on the temporal correlation between them in the required voltage values based on running a representative set of benchmarks.

The problem solved in this paper can be stated as follows. Given a representative workload for a multicore processor and a target number of DVFS domains, we determine how the system should be partitioned into DVFS clusters. Our work is targeted towards a multiprogramming environment where each core runs a separate program, and has a private cache and a shared main memory, and there is little to no communication between cores since the programs are largely independent. Based on this representative set of benchmarks, we determine the optimal number of cores for each DVFS cluster.

Our approach first finds the optimal voltage/frequency schedule on a per-core basis using a bounded enumeration scheme, and then clusters processors together, depending on an affinity metric based on the per-core voltage/frequency schedule. Finally, the efficacy of the clustering is examined by determining the voltage/frequency schedule for each cluster.

Note that the optimization in this paper is a one-time, off-line optimization. Once these clusters have been detected and implemented in hardware, runtime techniques can be used to run programs on the multicore processor by mapping applications to these DVFS clusters. The development of such techniques is beyond the scope of this paper, but examples of prior work that solves related problems include [7], [8]. Using offline or online compiler analysis, the compiler can identify program regions for V_{dd} and frequency adjustments, and this information can be coupled with the set of available processors at the time the program commences. Alternatively, one could extract a signature from the programs in the representative workload that captures its key performance features. When a new program is to be scheduled, one could compare the signature from this program with those of the representative workload to determine which domain it best belongs to.

II. Overview of the approach

A. Objective function

We consider a CMP executing a multiprogrammed workload where each core runs an independent application. As a result, note that we often use the terms "core" and "application" interchangeably in the following discussion. We consider determining the optimal DVFS schedule with minimum cost, under several circumstances:

1) Hard constraint: The application is required to execute completely in a given duration T and the cost function is the power consumed

$$\text{Cost} = \text{Power} \tag{1}$$

2) Soft constraint: The application is not required to execute completely, but there is a penalty associated with the incomplete instructions and this is built into the cost function. The cost function for this case is defined as

$$\text{Cost} = \text{Power}\ (\ 1 + K \times (1 - \eta))\ ) \tag{2}$$

where K is a user-defined constant and η is the fraction of instructions that are completed by the deadline. In choosing K, it can be seen that the higher the value of K more is the penalty for incompletion and vice versa. In selecting the value of K, the relative weight of the two components of the objective function can be determined by looking at possible values of η. Note that $0 \leq \eta \leq 1$ by definition, and typically its value is closer to 0 than to 1. Moreover, to ensure a reasonable solution, we insert an additional requirement that

$$\eta > \eta_{spec} \tag{3}$$

In our experiments, we set $\eta_{spec} = 0.85$.

The second model above is based on the intuition that allowing a small percentage of incompletion might allow greater power savings since the core can operate at lower frequencies for more time. For instance, if the runtime of one of the programs is large but relatively insensitive to the value of V_{dd}, then the advantage of speeding it up is minimal, but its large runtime makes it liable to seek the highest frequency at most times. Allowing some level of incompletion relaxes the frequency requirements for this processor.

However, any decision to do so must involve a requirement that a minimum number of instructions is executed, and must contain a penalty component for incomplete instructions. If the penalty for incomplete instructions is lower than the power that would be consumed to ensure completion, an incomplete execution may be favored by the deadline, so that execution is completed slightly later than the deadline.

In general, as we will see, the soft deadline model better captures the essence of DVFS. For example, if a benchmark has more memory-bound cycles and periods of low activity, the first objective function may assign it to high-frequency operation, but this simply increases the total power dissipation with little improvement in the throughput.

B. Outline of the method

Under our scheme, clustered DVFS in a multicore processor is based on their performance on a typical workload in a multiprogramming environment. For this workload, cores that have good temporal correlation in the required voltage and frequency of operation must be clustered together.

Our optimization approach consists of the following steps:

- *Finding the per-core DVFS schedule for each core*: In the first step, we temporarily assume that each core can be independently biased, as in a per-core scenario. Under this assumption, we use an offline approach to determine the best possible DVFS for each core, as an intermediate step in the overall solution. The idea of this step is to determine a DVFS schedule that corresponds to the best overall flexibility and optimal power dissipation over all cluster assignments. We develop an implicit enumeration scheme to solve this problem, incorporating procedures to ensure that at a practical level, its computational cost is reasonable.
- *Clustering "similar" cores together*: The per-core DVFS schedule presents the "unconstrained" optimal solution, and our goal is to find a cluster whose performance is as close to this as possible. We use the per-core schedule as a starting point and define a similarity metric between these schedules for all processors. If we cluster together groups of cores for which the per-core solution is temporally similar over a number of intervals, each core can eventually achieve a temporal assignment that is "close" to the per-core assignment. We use the K-means algorithm to perform this clustering based on the output metric.
- *Finding the DVFS schedule for each cluster to evaluate the solution*: The final step is an evaluation step to measure the performance of the clustering procedure. To determine the quality of clustering, we now find the optimal DVFS schedule for each cluster, using an approach similar to the first step. This can now be compared with the original performance to obtain a measure of the solution quality.

III. Ingredients of the algorithm

In this section, we will describe the basic ingredients of the algorithm. To begin with, Section III-A will describe how we profile the representative workload to compactly capture the information that is required by our algorithm. Next, in Section III-B, we present an implicit enumeration scheme, which is very computationally efficient in practice, to find the optimal per-core DVFS schedule. The information from the per-core DVFS is then used to define affinities, and cores are then clustered using the K-means clustering algorithm, described in Section III-C. Finally, we verify the performance of the clustered DVFS using a variant of the implicit enumeration algorithm, as described in Section III-D.

Note that since this optimization is performed offline, our DVFS is not strictly dynamic scheduling (which is performed online), but represents the best DVFS schedule possible.

Note that all cores are homogeneous, i.e., structurally similar. Here, similarity refers to their workload characteristics.

A. Initial profiling

Our assumption is that the workload provided is representative of a real workload, and that conclusions drawn from an intensive analysis of this workload can lead us to create DVFS clusters that operate for a wider range of typical applications to be run on the multicore processor, with intelligent scheduling algorithms.

Each program in the workload runs on a separate core in a multiprogramming environment, and as a starting point, we profile each of these programs to obtain an idea of the voltage/frequency needs of the core. Given a set of discrete frequencies at which the programs will be executed, we execute these programs in a cycle-accurate simulator to obtain the power dissipation at this frequency. We store the data for further analysis; given the large volume of data, we sample each benchmark once in every 1000 cycles with no significant loss in accuracy.

B. Implicit enumeration scheme

The first step in our implementation of DVFS for multicore system requires finding the optimal frequency and voltage schedule for each core such that the cost is minimized. A similar scheme is used in the third step, where the performance of the clustering approach is analyzed. This section explains the algorithm used in the implicit enumeration scheme that drives both methods.

The multiprogrammed workload scenario requires applications to be run on a core each. Let us consider one such core. Let the target execution time for the application running on the core be T. Let this time T be divided into M equal time steps such that

$$\sum_{i=1}^{M} t_i = T \tag{4}$$

Here, each time step corresponds to a DVFS interval, i.e., the voltage and frequency may be changed at the beginning of this interval. We will first assume per-core DVFS in illustrating this algorithm, and then show the extensions of this approach to per-cluster DVFS later.

The supply voltage and frequency are chosen from the set of available choices, (V_j, f_j), $j \in 1, \ldots, N$, which form a set of N discrete V_{dd}-frequency pairs available. This set is considered to be an input to the procedure, and captures the DVFS capabilities of the multicore processor. Here, V_j is the minimum supply voltage required to sustain an operating frequency of f_j. The objective of the offline implicit enumeration scheme is to find the assignment (V_j, f_j) for each t_i such that the cost function is minimized. This assignment is the optimal DVFS schedule for the core for that interval.

The algorithm is applied to each core, and starts at the first time step and proceeds till the M^{th} step, computing a list of possible tuples that capture its performance up to the end of the step. Each tuple is represented by (I, C), where I is the number of instructions committed till the end of the time step and C is the associated cost till the end of the time step. The value of C is computed according the objective function that is selected from among the possible choices in Section II-A.

A naïve (and impractical) way of computing the (I, C) tuples is as follows. At time 0, the only tuple is $(0, 0)$. At the end of the first time step, the value of I and C is computed for each possible (V_j, f_j) pair: therefore, there are N such tuples. At the end of the k^{th} time step, the set of tuples is taken by combining the tuples from time step $k - 1$, and considering an execution in time step k at each of the N possible voltage-frequency pairs.

The profiling information from Section III-A presents a quick and compact table-lookup operation that can be used to predict the number of instructions at frequency f_j, starting from a given instruction count, performed during a given interval, and also the corresponding contribution to the cost function.

The drawback of this naïve method is its complexity. It is easy to see that after M time steps, we would have $O(M^N)$ tuples for each core. This exponential complexity is clearly not a scalable solution.

Therefore, we explore ways of pruning this list. We employ a mix of exact and heuristic strategies for this purpose.

- *Provably suboptimal tuples*: Let (I_a, C_a) and (I_b, C_b) be two elements in the list of tuples. Then, if $I_a \leq I_b$ and $C_a > C_b$, we can be certain that (I_a, C_a) is suboptimal. Stated another way, if there is a tuple (I_b, C_b) that completes an equal or greater number of instructions than I_a at a lower cost than C_a, then it will always be preferred to (I_a, C_a) in any optimal solution, and therefore the latter can be eliminated.
- *Heuristic pruning strategies* are also possible. For example, for the first objective function, where we have a hard requirement that the application has to complete by a specific deadline, we can observe that if the application is to complete by the end of M^{th} time step, it has to complete approximately $\frac{i}{M}^{\text{th}}$ of the total number of instructions by the end of i^{th} time step. Given the nonuniform structure of any computation, we include a tolerance factor that makes this estimate very liberal. At the i^{th} step, any tuple that has completed fewer instructions than this limit is eliminated. A similar formulation can be created for the second objective function: assuming that $\eta\%$ of all instructions are completed by the M^{th} time step, at the end of i^{th} time step, it is reasonable to assume that about $\eta \frac{i}{M}^{\text{th}}$ of all instructions are executed; this requirement may be made more liberal through a multiplicative factor of less than 1.

Further approaches for pruning tuples may be used: for example, if a tuple is within some ϵ of being provably suboptimal to another, it could be deleted. However, we found that further strategies were unnecessary in our implementation, and that the above two approaches worked well enough.

An example of the trend showing the growth in the number of tuples with respect to program time, which is observed in practice for benchmark galgel, which is run repeatedly from $t = 0$ to $t = 0.9s$, using three candidate frequencies and for different values of M, is shown in Figure 2. It can be seen that the growth without pruning suboptimal tuples is exponential whereas, with pruning, the growth is very manageable.

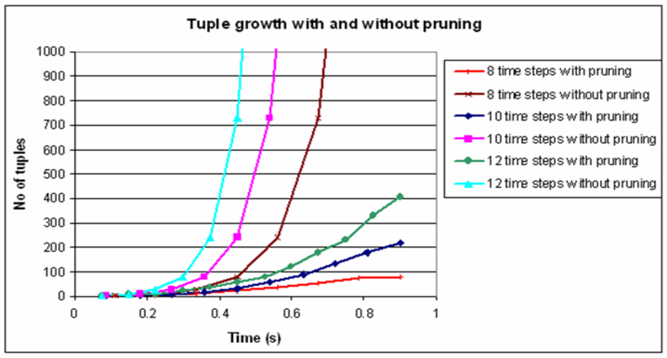


Fig. 2. Tuple growth as a function of time for different values of M.

C. Finding the DVFS clusters

The optimal voltage assignment for each core obtained will have voltages assigned for each of the M time steps and hence can be seen as a point in an M-dimensional space. The $M-$dimensional points corresponding to the cores are then clustered using the K-means clustering algorithm. Since this is a standard clustering technique with several software implementations in the public domain, we do not present the details of the approach. The algorithm has the limitation that it converges to a local minima and the result depends largely on the initial centroids chosen. Hence, several trials with random initial centroids are carried out in an attempt to find global minimum. The K-means algorithm provides an optimal set of clusters that correspond to the optimal clustered DVFS solution.

D. Verifying the performance of the clusters

Once the optimal clusters have been determined and cores assigned to these clusters, it is important to evaluate the performance of the clustering approach. For this purpose, we run the clustered programs under the assumption that all cores within a cluster must have their voltage and frequency values changed simultaneously.

The problem formulation for finding this optimal DVFS schedule for a cluster of cores is similar to that of implementing per-core DVFS. Let the target execution time for the applications running on all the cores in the cluster be T. As before, let this time T be divided into M equal time steps such that $\sum_{i=1}^{M} t_i = T$. In this case, the objective is to find the assignment (V_j, f_j) for each t_i such that the sum of the associated costs of all the cores in the cluster is minimized, under the constraint that a certain percentage of the total number of instructions present in each of the applications is completed. The implicit enumeration procedure to find the frequency and voltage schedule for a cluster proceeds in the same way as implementing DVFS for a core. The difference is that instead of calculating tuples for one core at each time step, we calculate tuples for all the cores in a cluster at each time step. Another difference is that the suboptimal tuple for a core can be eliminated only if the corresponding tuples for all other cores in the cluster are also suboptimal. The complexity of the algorithm is the same as that of per-core case, since we do essentially the same but for multiple cores at a time. However, as before, the use of tuple elimination, coupled with our heuristics, keeps the complexity reasonable.

IV. Experimental results

A. Experimental setup

We consider a homogeneous 16-core CMP in 45nm technology. Figure 3 shows the organization of the CMP that is considered. It has 16 processing cores, and each core has its private L1 and L2 cache. The L3 cache is shared.

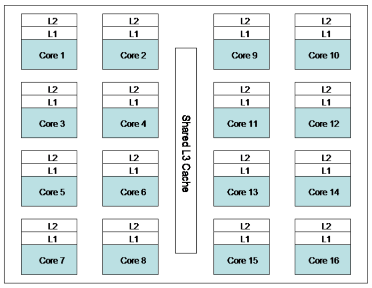


Fig. 3. Block diagram of the simulated 16-core CMP

For our multiprogramming environment with individual programs running on separate cores, we use a cycle-accurate out-of-order uniprocessor simulator, SimpleScalar [9], to simulate each individual core of the CMP. Clearly, using SimpleScalar, a uniprocessor simulator, to model a CMP system is an approximation: in particular, the shared cache cannot be modeled using this simulator. This approximation is reasonable in a multiprogramming environment since each processor runs an independent application, and realistically, the contention for shared cache is small. The simplicity and ease of use of SimpleScalar drives our choice, and for the above reasons, it is adequate. The processor configuration modeled in SimpleScalar is as shown in Table I. Each processing core is an out-of-order processor and has private L1 data and instruction caches and a private unified L2 cache. The L1 data and instruction cache access latency is 1 processor cycle, and the L2 unified cache access latency is 4 processor cycles. Since the L1 and L2 caches are private for each core, their access latency in terms of the number of processor cycles is the same at all frequencies, but the number of cycles required to accesses the main memory change with processor frequency.

TABLE I
Configuration of a single processor

Fetch/Decode/Issue/Commit width	4/4/4/4 (instructions/cycle)
RUU size	64 entries
LSQ size	32 entries
Private L1 Data cache	16KB, 4-way SA, 32B block size
Private L1 Instruction cache	16KB, 4-way SA, 32B block size
Private L2 Unified Data and Instruction cache	512KB, 8-way SA, 64B block size
Memory access bus width	8 bytes
Data Translation Lookaside Buffer	512KB, 4-way SA, 4KB block size
Instruction Translation Lookaside Buffer	256KB, 4-way SA, 4KB block size
# integer ALUs	4
# integer multiplier/dividers	4
# floating point ALUs	2
# floating point multipliers/dividers	2
# memory system ports available to CPU	2 (1 read, 1 write)

For power estimation, we employ Wattch, an architectural-level power modeling tool that is integrated with SimpleScalar. We have updated the technology-specific parameters in Wattch based on the ORION 2.0 [10] technology file.

The first step in our implementation consists of gathering the profiles of the representative workload, as described in Section III-A. Sixteen benchmarks from the SPEC CPU2000 suite were used, of which nine are integer and seven are floating point benchmarks. Each is assumed to run on a separate core, repeating after completion until 1.5s at 0.5GHz. These benchmarks were run on the SimpleScalar simulator with the MinneSPEC input sets [11].

TABLE II
Supply voltages and supported frequencies

Processor frequency (in GHz)	0.5	0.7	0.9	1	1.1
Supply Voltage (in V)	0.8	0.9	1	1.1	1.2

We assume a set of discrete (V_{dd},frequency) pairs available for DVFS, as shown in Table II, taken from [12]. The memory access latency and translation lookaside buffer (TLB) miss penalty corresponding to each of the processor frequencies from Table II are shown in Table III.

TABLE III
Memory access latency and TLB miss penalties (in processor cycles) for each frequency

	0.5 GHz	0.7 GHz	0.9 GHz	1 GHz	1.1 GHz
Memory access latency	25	35	45	50	55
TLB miss penalty	30	42	54	60	66

B. Results

We evaluate configurations of 1, 4, 8 and 16 clusters: the 1-cluster configuration corresponds to the per-chip DVFS case and the 16-cluster configuration corresponds to the per-core DVFS case. For every cluster configuration, we set upper and lower limits on the number of cores per cluster with the goal of ensuring that no regulator is excessively overloaded or underloaded. For the 4-cluster case, the number of cores per cluster must be between 3 and 5; for the 8-cluster case, this number must be between 1 and 3.

We evaluate various target times under each of the two objective functions in Section II. Recall that at the lowest frequency, the runtime of the workload is 1.5s. We evaluate target times of 0.9s, 1.1s, and 1.3s for the first objective function, 0.9s and 1.1s for the second. Due to space limitations, we show a selected set of results. A reasonable value for the penalty K is empirically found to be $K = 5$, where only a few cores have an 85-90% completion rate; the others have a higher rate of completion of 98-100%. For a lower value of K, the penalty is insufficient and the tradeoff is worse: e.g., for $K = 2$, almost all benchmarks have a small completion rate, close to η_{spec}. For $K = 5$, as stated earlier, the incomplete programs are primarily slow because they are awaiting memory accesses, and speeding up the processor would not help their cause.

The per-core DVFS schedules of cores in these clusters, along with the per-cluster DVFS schedules for 4-cluster case is shown in Figure 4. The five frequencies are assigned indices:

0.5GHz is assigned an index of 1 and 1.1GHz is assigned an index of 5. Several observations can be made from this:

- The result of per-core DVFS assignment for each core is shown in each subfigure, and the black line indicates the cluster DVFS assignment. Given that two of the benchmarks, mgrid and art, have a per-core assignment at the maximum value for all time, clearly the per-chip DVFS solution will also be at the maximum value: clearly, this is a wasteful use of power for the programs in the other clusters. The per-cluster case, on the other hand, places them in a cluster together, and enhances the likelihood that other compatibilities and clusters can be found.
- The number of cores per cluster is not uniform, and indeed reaches the bounds of 3 and 5. This implies that using a uniform number of cores per cluster, as in [4] and other prior work, is suboptimal.
- As the deadline becomes more relaxed, there is greater flexibility for clustering cores together and the cluster sizes tend to become more uniform. For the 4-cluster 16-core case, under a 0.9s deadline, the number of cores per cluster were $\{3, 3, 1, 1\}$; when the deadline is relaxed to 1.1s, the corresponding numbers were $\{3, 2, 2, 1\}$. Similarly, for the 8-cluster case, the number of cores per cluster went from $\{3, 3, 3, 3, 1, 1, 1, 1\}$ to $\{3, 3, 3, 2, 2, 1, 1, 1\}$. All evaluations were under the second objective function for $K = 5$.

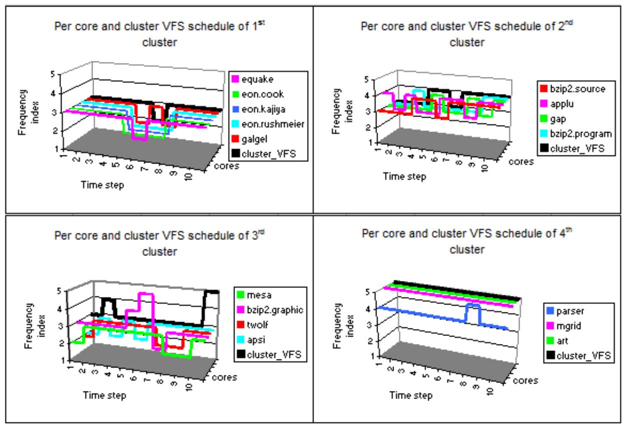
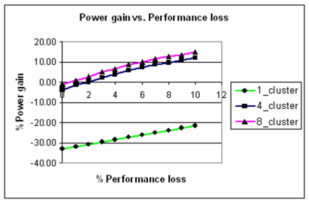
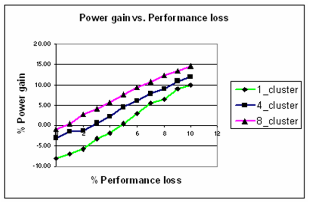


Fig. 4. DVFS for 4-cluster case under first use model and 0.9s target execution time



(a) Hard constraints



(b) Soft constraints

Fig. 5. Comparison of the power saved from the different cluster configurations for 0.9 s target execution time

Next, we evaluate the reduction in the power ("power gain") as we move from per-core to per-chip DVFS, and we examine how much performance we must sacrifice to match the power dissipation of the per-core case. Results of this evaluation for first and second objective functions, for a target time of 0.9s, are shown in Figure 5. We relax the target execution time ("performance loss") to achieve the power dissipation of the per-core case. For the 1-, 4-, and 8-cluster cases, the x-axis crossing shows the performance loss at which the power matches the per-core case. This degradation is seen to reduce with the number of clusters. The per-chip case fails to match the per-core case even with 10% performance loss under the first objective function, but requires only a 5% loss in the second. This data clearly indicates (a) the importance of clustered DVFS over per-core DVFS (b) the importance of allowing the flexibility of the soft constraints, and (c) diminishing returns with increasing the number of clusters: in particular, going from 4 to 8 clusters is not worthwhile here.

V. Conclusion

We have developed an effective approach for DVFS domain assignment in a multiprogramming environment. We find the optimal per-core schedule using a novel implicit enumeration environment with aggressive pruning, and then cluster these schedules to find the best assignments. Our results indicate significant benefits from clustered DVFS over per-core DVFS, but diminishing returns with increased number of clusters.

References

[1] Y. Li, K. Skadron, D. Brooks, and Z. Hu. Performance, energy, and thermal considerations for SMT and CMP architectures. In *Proceedings of the International Symposium on High-Performance Computer Architecture*, pages 71–82, 2005.

[2] L. Hammond, B. A. Nayfeh, and K. Olukotun. A single-chip multiprocessor. *IEEE Computer*, 30(9):79–85, 1997.

[3] W. Kim, M. S. Gupta, G.-Y. Wei, and D. Brooks. System level analysis of fast, per-core DVFS using on-chip switching regulators. In *Proceedings of the International Symposium on High-Performance Computer Architecture*, pages 123–134, 2008.

[4] S. Herbert and D. Marculescu. Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 38–43, 2007.

[5] P. Juang, Q. Wu, L.-S. Peh, M. Martonosi, and D. W. Clark. Coordinated, distributed, formal energy management of chip multiprocessors. In *Proceedings of the International Symposium on Low Power Electronics and Design*, 2005.

[6] J. Li and J. F. Martínez. Dynamic power-performance adaptation of parallel computation on chip multiprocessors. In *Proceedings of the International Symposium on High-Performance Computer Architecture*, 2006.

[7] Canturk Isci, Alper Buyuktosunoglu, Chen-Yong Cher, Pradip Bose, and Margaret Martonosi. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *Proceedings of the International Symposium on Microarchitecture*, pages 347–358, December 2006.

[8] M. Martonosi and S. Kaxiras. *Computer Architecture Techniques for Power-Efficiency*. Morgan and Claypool, San Rafael, CA, first edition, 2008.

[9] D. Burger and T. M. Austin. The SimpleScalar tool set, Version 2.0. *ACM SIGARCH Computer Architecture News*, 25:13–25, 1997.

[10] A. B. Kahng, B. Li, L.-S. Peh, and K. Samadi. ORION 2.0: A fast and accurate NoC power and area model for early-stage design space exploration. In *Proceedings of Design, Automation & Test in Europe*, pages 423–428, 2009.

[11] AJ KleinOsowski and D. J. Lilja. MinneSPEC: A new SPEC benchmark workload for simulation-based computer architecture research. *Computer Architecture Letters*, 1:7–7, 2002.

[12] J. Howard *et al.*, A 48-core IA-32 message-passing processor with DVFS in 45nm CMOS. In *Digest of Technical Papers, International Solid-State Circuits Conference*, pages 108–109, 2010.