Formal Control Techniques for Power-Performance Management

THESE TECHNIQUES DETERMINE WHEN TO SPEED UP A PROCESSOR TO REACH PERFORMANCE TARGETS AND WHEN TO SLOW IT DOWN TO SAVE ENERGY. THEY USE DYNAMIC VOLTAGE AND FREQUENCY SCALING TO BALANCE SPEED AND AVOID WORST CASE FREQUENCY LIMITATIONS FOR BOTH MULTIPLE-CLOCK-DOMAIN AND CHIP MULTIPROCESSORS.

Qiang Wu
Philo Juang
Margaret Martonosi
Li-Shiuan Peh
Douglas W. Clark
Princeton University

Recently, microprocessors have increasingly faced vexing power and thermal challenges, such as adhering to maximum and average power dissipation limits, avoiding thermal hotspots, and providing effective voltage regulation on the chip. Researchers have proposed numerous techniques to address these problems, which will intensify with new processor generations.

Much research focuses on static, design time techniques for power management, but dynamic techniques are also appealing because they let the computer system make adjustments on the fly, according to the current power situation. Indeed, part of the difficulty with static, design time decisions is the impossibility of knowing the true worst case. With dynamic techniques, performance is no longer tied to static worst-case estimates. Instead of forcing designers to choose a clock frequency and voltage combination that accommodates the worst-case application without a power emergency, dynamic voltage and frequency scaling (DVFS) invokes restrictive frequency and voltage *only* when the DVFS controller deems the worst case is imminent. But while dynamic techniques have strong advantages, their behavior is hard to predict with enough accuracy to guarantee that they will adhere to the worst-case limits in all situations. Even if designers could bound and reason about a single on-chip dynamic-control response, analysis becomes much more complex when the response must compose the effects of several control mechanisms that operate simultaneously.

Formal control-theoretic techniques have proved useful in many dynamic control scenarios—from aerospace systems to manufacturing—because their formal underpinnings let designers analyze and bound worst case behavior. In microprocessor power control, such techniques let designers reason about individual control responses and enable them to predict response behavior when individual responses are composed into an overarching complex control response.

For the past two years, we have studied the application of formal control analysis and implementation techniques to power problems in high-end microprocessors, with a focus on managing DVFS adjustments. In

Published by the IEEE Computer Society

0272-1732/05/$20.00 © 2005 IEEE

multiple-clock-domain (MCD) processors, we have studied methods for speed balancing among voltage-frequency islands by using formal control on the occupancies of the synchronizing queues that link these islands. This work extends naturally to chip multiprocessors (CMPs). For these, we assume that each multiprocessor has several on-chip cores and that each core has its own voltage-frequency island. Together the cores run a multithreaded program. As with MCD multiprocessors, queues serve as a proxy for the processing load on each core, except that for CMPs they are each CMP's local task queues.

Our work[1-3] contributes to high-performance multiprocessing in three significant ways:

- Our rigorous modeling and stability analysis techniques provide designers with insight and guidance, and make the design more efficient and resilient.
- Our control algorithms are straightforward and hardware-efficient.
- Our DVFS controllers outperformed existing online DVFS schemes, offering a two- to eightfold savings in energy-delay product over a wide range of applications in both MCD processors and CMPs.

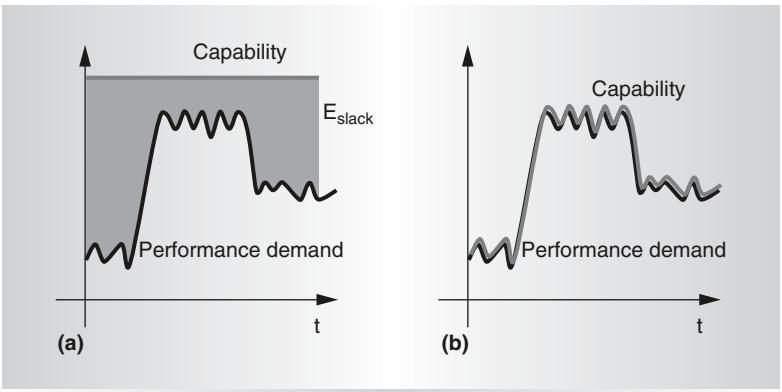

Figure 1. For a processor, online dynamic voltage and frequency scaling aims to scale frequency to match varying performance demand. Without DVFS (a), energy waste (E_{slack}) occurs (shaded area), but a perfect DVFS control scheme (b) can result in a perfect match between domain-execution capability and demand.

Modeling the problem

As Figure 1 illustrates, the goal of DVFS is to scale the frequency to match varying performance demand, or workload changes. Because changing clock frequency will alter the execution speed, a perfect DVFS scheme will lead to a perfect match between the demand and domain execution capability, with no performance degradation or energy waste because of a mismatch between the two, E_{slack}. Thus, for a perfect DVFS, as in Figure 1b, E_{slack} essentially also represents energy savings.

Queue occupancy is our DVFS controller's main input. An MCD processor typically uses interface queues between clock domains to reduce the risk of metastability, which might occur when data moves from one domain to another. These queues can also give clues about the speed balance between the sender and receiver domains. An emptying queue, for example, might mean the receiver is too fast relative to the sender, while a filling queue might mean it is too slow. A stationary queue is a perfect match of receiver and sender speeds. In a CMP, our model's queues correspond to each core's task queues, which contain the threads scheduled to run on that core. Again, an emptying task queue indicates that the core is running fast and able to absorb its assigned workload; a filling task queue indicates that the core is not handling the workload fast enough. A feedback control scheme for DVFS that uses the queue occupancy as a feedback signal to control the domain frequency can adapt execution speed to varying demand. If the adaptations are fast enough, the DVFS scheme should yield results close to the perfect matching in Figure 1b.

Queue and clock domain dynamics

Figure 2 is a single-queue model for a clock domain with an input queue. For a clock domain, the frequency and corresponding voltage cannot change instantaneously, and the minimum time requirement is one possible frequency change. Our model thus has a control interval that fixes the frequency inside. Using T as the length of a single control interval, the kth control interval is just the time period $[kT, (k + 1)T]$, where N is the total number of sampling periods in a control interval. The length of each sampling period is Δt, so $T = N\Delta t$.

As a first step, we modeled performance demand $\lambda_{(t)}$ and service rate $\mu_{(t)}$ inside each control interval as an independent and sta-

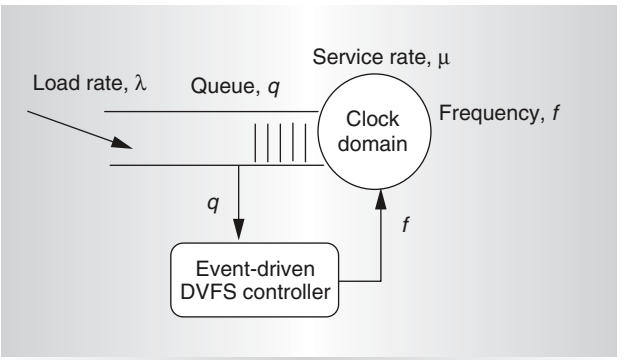

Figure 2. Single-queue model for a clock domain with input queue *q*. The circle represents the clock domain being considered, such as a floating-point functional unit. The domain has frequency *f* and an execution capability or service rate, μ, a function of *f*. The performance demand is λ, and the queue has finite size.

tionary random process along the time axis.[4] That is, they have identical distributions for all *t* inside an interval. The expected values and variances (or noise levels) of $\lambda_{(t)}$ and $\mu_{(t)}$ are $\bar{\lambda}$, $V_{(\lambda)}$, $\bar{\mu}$, and $V_{(\mu)}$, respectively. (From this point, for variable *x*, $\bar{x}$ represents *x*'s expected value, and $V_{(x)}$ represents its variance. The subscript *k* means that these values are for the *k*th control interval.)

Given a queue occupancy at time *t* of $q_{(t)}$, the basic queue equation is

$$q_{(t+\Delta t)} - q_{(t)} = (\lambda_{(t)} - \mu_{(t)})\,\Delta t \qquad (1)$$

Thus, queue occupancy change within some time period equals the number of arriving elements minus the number of departing elements in that period.

Next we used the basic queue equation to model the queue domain dynamics across different control intervals. The average queue occupancy over all sampling points in the previous interval is the feedback signal for the current interval. We denote the feedback signal for the *k*th control interval as q'_k and the queue occupancy at the *k*th interval's starting point as q_k. We can then express these two dynamic state variables in terms of values from the previous interval, and recursively expand them using the basic queue dynamics in Equation 1. A continued derivation[2] of q'_k and q_k yields an analytical model that describes the dynamics in the queue domain's system of interest across different control intervals:

$$\bar{q}'_k = \bar{q}_{k-1} + \frac{T}{2}\left(\bar{\lambda}_{k-1} - \frac{1}{\bar{t}_1 + \dfrac{\bar{C}_2}{f_{k-1}}}\right)$$

$$\bar{q}_k = \bar{q}_{k-1} + \frac{T}{2}\left(\bar{\lambda}_{k-1} - \frac{1}{\bar{t}_1 + \dfrac{\bar{C}_2}{f_{k-1}}}\right)T \qquad (2)$$

where $\bar{t}_1$ and $\bar{C}_2$ are parameters that describe $\bar{\mu}_{k-1}$ (the expected value of the service rate) in terms of the real control signal—frequency f_{k-1}.[2]

Intuitively, this means $\bar{q}'_k$, the expected value of the average queue occupancy over a control interval, equals the sum of the queue occupancy at the beginning of interval $\bar{q}_{k-1}$, and the average queue changes because of the differences between the demand and service rates ($\bar{\lambda}_{k-1}$ and $\bar{\mu}_{k-1}$). We can similarly interpret $\bar{q}_k$, the expected value of the queue occupancy at the beginning of the next interval.

Interval-based controller design

In designing an interval-based DVFS controller for MCD processors, a straightforward approach would be to control the interval frequency as in Figure 3a. As Equation 2 indicates, however, this control system is nonlinear, and tools for general nonlinear systems are very limited,[5] which makes designing an effective controller difficult. Fortunately, as Figure 3b shows, we can separate the nonlinearity in the original system dynamics and use a feedback linearization or nonlinear transformation[6] to compensate for the nonlinearity, which yields an essentially linear system.

Linearization makes it possible to choose from a rich set of linear control techniques,[6] including many variations of the proportion-integral-derivative (PID) controller. As Figure 4 shows, PID-based controllers adjust the execution rate to adapt to the workload change, considering the value and change rate, as well as the controller's observed history of workload changes.

Our interval-based DVFS controller, which the following state equations describe, is based on a proportion-integral (PI) controller, a popular variant of the PID-based controller.

$$
\begin{aligned}
\bar{q}'_k &= \bar{q}_{k-1} + \frac{T}{2}\left(\bar{\lambda}_{k-1} - \bar{\mu}_{k-1}\right) \\
\bar{q}_k &= \bar{q}_{k-1} + \left(\bar{\lambda}_{k-1} - \bar{\mu}_{k-1}\right)T \\
e_k &= \bar{q}'_k - q_{\text{ref}} \\
\bar{\mu}_k &= \bar{\mu}_{k-1} + K_I e_k + K_P\left(e_k - e_{k-1}\right) \\
f_k &= \frac{\bar{C}_2 \bar{\mu}_k}{1 - \bar{t}_1 \bar{\mu}_k}
\end{aligned}
\tag{3}
$$

The first two subequations in Equation 3 are simply the analytic queue-domain model in Equation 2; in the other subequations, q_{ref} is the reference queue occupancy—the target or nominal operating queue point; e_k is the error signal; μ_k is the new service rate coming from the PI controller, with K_I and K_P the control parameters (or the so-called control gains); and f_k is the new clock frequency obtained from μ_k.

We then analyzed the stability and transient performance of Equation 3 and chose appropriate control gains, K_I and K_P, a process we describe in detail elsewhere.[2]

Implementation issues

Implementing our DVFS controller requires only a modest amount of hardware support.[2] Two counters are needed to frame the control interval and to record cumulative queue occupancy. Given that the control interval is typically around a few thousand cycles, and the queue size is typically around tens or hundreds, small 16- to 32-bit counters should suffice in most cases. Apart from these counters, the most complicated logic is to compute control signal f. This logic consists of two parts: one to compute μ_k and one to compute the nonlinear transformer (obtaining f_k from μ_k). Elsewhere[2] we proposed two possible implementations for these tasks, which use small, precomputed lookup tables or small multipliers. Though this part of the logic is more complicated, the control algorithm activates it only once for each control interval. Therefore, the overall hardware cost and power consumption are still negligible, relative to total system power consumption.

Reference queue occupancy q_{ref} is a key factor for our DVFS controller because its value specifies the actual trade-off between performance degradation and energy savings. Increasing that value makes DVFS control more aggressive in saving energy; decreasing it emphasizes performance. A design could also have the operating system or application software adjust q_{ref}, for example, through a special mode-set instruction. This flexibility provides the opportunity for hardware and software to cooperate in achieving energy efficiency. The hardware would be responsible for implementing the fine details of speed adaptation while the software would make the overall policy decision (through q_{ref}) on how aggressively to save energy or preserve performance.

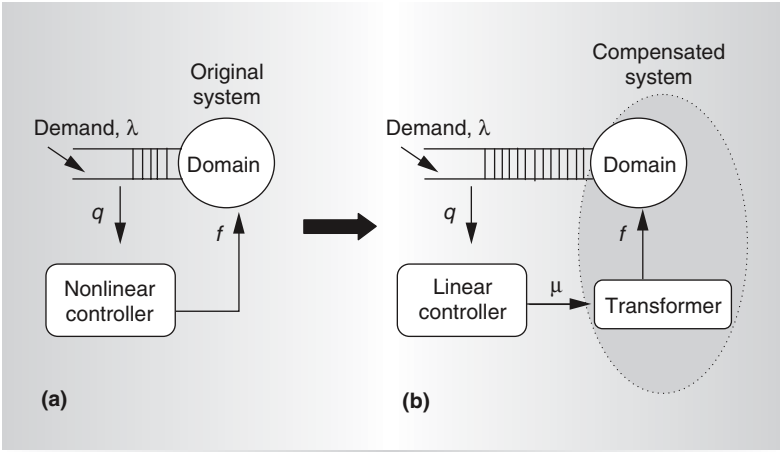

Figure 3. Making the control system linear. A simple design would aim to control interval frequency directly (a), but the current control system is nonlinear, making design difficult. A better approach is first to separate the nonlinearity inside this system to arrive at a nonlinear transformation on the feedback path (b). In this transformation, μ_k becomes the control signal for the compensated linear system, and actual or internal control signal f_k is obtainable through a transformation on the feedback path.

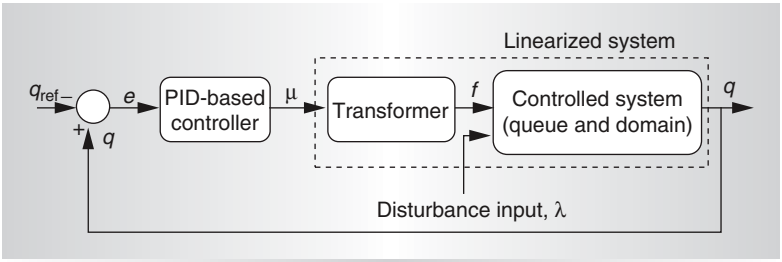

Figure 4. General block diagram for a proportion-integral-derivative (PID) controller. In the figure, q is the measured feedback signal; q_{ref} is the target or nominal queue operating point; e is the error signal and the controller input; μ is the obtained control signal (or actuate signal), which is proportional to the error value, the integral of previous errors, and the error change rate; f is the frequency, which the controller obtains from μ through a nonlinear transformation. Finally, demand λ is the disturbance input, which the controller is trying to match by adjusting the execution rate.

Table 1. Summary of simulation parameters.

Parameter	Value
Reference queue point (q_{ref})	6 for integer, 5 for floating-point, and 3 for load/store
Domain frequency range	250 MHz to 1.0 GHz
Domain voltage range	0.65 V to 1.20 V
Frequency/voltage change speed	73.3 ns/MHz, 171 ns/2.86 mV
Control interval length	10,000 instructions
Domain clock jitter	±110 ps, normally distributed
Interdomain synchronization window	300 ps

Experimental results

Our simulation environment is based on the SimpleScalar toolset[7] with the Wattch[8] power estimate extension and the MCD processor extension.[9] The MCD extension has four clock domains and includes a cycle-by-cycle computation of the synchronization overhead from independent clock frequency, phase, and clock jitter. The environment also implements an XScale-like dynamic voltage and frequency changing mechanism that lets the hardware use any frequency within the allowable range.

We implemented the online DVFS controller for local queues and domains, following the design described earlier. Also, as others have done,[10,11] we ran the front-end domain at a fixed maximum speed and allowed the DVFS controller to control the integer, floating-point, and load/store domains. Table 1 summarizes our simulation parameters. We assumed a performance degradation target of about 4 percent, roughly the same as that in other experiments,[11] and a q_{ref} roughly a third of the total size for integer and floating-point domains (6 for the integer domain, 5 for floating-point). Since the load/store domain is relatively more critical to overall performance,[12] we chose a q_{ref} value of 3 for it, roughly a fifth of the total size. We also assumed that the processor would apply clock gating whenever the unit was idle. The selected values for all other architecture parameters are the same as those in our previous work.[10,11] (Some values, such as clock jitter, are based on published industrial numbers.)

We obtained results for six Mediabench, eight SPECint, and four SPECfp applications—a program set that displays a wide range of program behavior.[2] To compare our results with those of other DVFS approaches, we held performance roughly the same for all approaches and looked at metrics such as energy savings, energy-delay product (EDP) improvement, and power-performance ratio—the percentage of power saved per percentage of performance degradation. Table 2 presents the comparative results.

SemiOracle assumes the DVFS, by oracle, has full knowledge of a program's slack, and it uses a Shaker algorithm to decide DVFS settings. Its results are thus not realistic, but rather serve as a comparison, although the SemiOracle result is not the upper bound for all possible DVFS results.[9] Heuristic represents the heuristic-based online DVFS that Semeraro et al. proposed.[11] We assumed an additional 5 percent performance degradation target for Heuristic and a 1 percent target for SemiOracle because these target values will lead to a performance degradation similar to analytic. Finally, for Synchro, which scales the frequency and voltage for the whole processor, we set the performance degradation to roughly the same value as for other approaches.

As the table shows, the overall results from our DVFS scheme (Analytic) are very promising. We achieved a power-performance ratio of 6.2 on average, relative to a fully synchro-

Table 2. Average results for a range of DVFS schemes.

Schemes*	Performance degradation (%)	Energy savings (%)	Energy-delay product improvement (%)	Power-performance ratio
Analytic	3.6	19.6	16.7	6.2
SemiOracle	3.7	18.1	15.0	5.6
Heuristic	5.8	11.9	6.8	3.0
Synchro	4.7	7.6	3.3	2.5

*Analytic represents our online DVFS scheme. SemiOracle is an existing scheme with SemiOracle-based DVFS. Heuristic represents heuristic-based DVFS. Synchro is conventional, fully synchronous voltage scaling.

nous processor. Compared with Heuristic, Analytic achieves far better results in EDP improvement and power-performance ratio. Analytic's average EDP improvement, for example, was 146 percent higher than that of Heuristic (16.7 percent versus 6.8 percent, with numbers relative to a synchronous processor. Analytic also produces a better average result than SemiOracle, with an average EDP improvement about 11 percent higher than that of SemiOracle (16.7 percent versus 15.0 percent). These results show the effectiveness of our DVFS controller's ability to automatically regulate voltage and frequency choices.

Finally, *all* MCD DVFS results (Analytic, SemiOracle, and Heuristic) are much better than those of Synchro, which shows the energy savings potential of an MCD processor with extra flexibility in DVFS control.

Extensions for chip multiprocessors

With extensions, our formal DVFS approach works equally well with CMPs.[1] Substituting individual processors for functional domains is fairly straightforward, but CMPs typically do not have the synchronization queues found in MCDs. Consequently, to extend our approach to CMPs, we used each processor's *task* queue—either a software data structure in the operating system or (as in our experiments) a hardware structure in each tile.[13] If each element in the task queue is a thread in a parallel application, then the faster a processor executes, the quicker the task queue will drain. Threads have variable execution times, just as individual instructions (especially loads) have significant variation. We assumed that a CMP consists of several tiles organized as a two-dimensional grid with a mesh interconnect and that each tile includes a processor, task queue, and associated memory controller.

Elsewhere[2] we argue that considering the interactions among MCDs significantly increases the problem's complexity and that such interactions are small enough to ignore. In contrast, in CMPs, interactions among parallel application threads are typically much stronger. Thus, although DVFS based on purely local information is simple and appealing, it is not always sufficient. In particular, for some applications, interthread relationships mean that the target q_{ref} has three critical requirements:

- It must adapt at runtime to match thread behavior.
- It must be based on global information, rather than fixed locally.
- Its setting must preserve performance.

Our solution for these requirements is *dist-PID*, a formal, online method that supports stable, distributed, coordinated DVFS control. In the experimental results section, we compare dist-PID with the purely local interval-based method we have described, which we ported trivially to work with CMPs.

Dist-PID

The idea behind dist-PID is that the threads on the critical path must run at maximum speed to preserve performance, but others can run more slowly to maximize energy savings without affecting performance. In parallel applications, critical-path threads are the last threads to reach a synchronization point. If each tile knows the expected execution time of the longest running-thread, it can adjust its processing speeds to match. Determining exactly which threads are on the critical path is highly challenging, so dist-PID attempts to identify these threads by choosing which tile has the most work left to do. This is not unreasonable because parallel sections require all threads to finish before moving on. Thus, the tile with the most work to do is likely to be the one on the critical path.

Dist-PID operation has three steps. First, at each tile, it estimates future queue occupancy (tile workload) assuming the maximum service rate, and denotes this as q_{target}. Second, through pairwise information exchange, each tile identifies the tile with the critical-path thread by keeping track of the highest q_{target} received. Finally, with this information, each tile determines the new service rates (tile frequency settings), slowing down tiles not executing critical-path threads. Figure 5 shows a snippet of execution for a simple parallel application in which we assume that a thread running on a tile has spawned three tiles: A, B, and C. Mathematically A, the slowest running core (or most heavily loaded), must equal the maximum frequency (1 GHz), running the tile on the critical path at highest frequency. In all other tiles, the equation produces a frequency lower than maximum because those

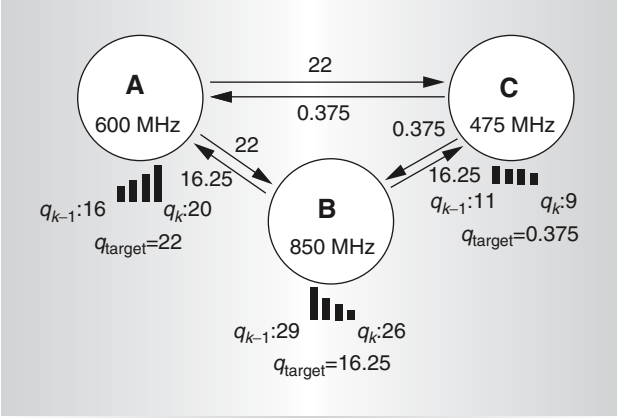

Figure 5. How dist-PID works. On tile A, queue occupancy increased (from 16 to 20) because the tile cannot keep up with the processing requirements, while the reverse is true at B (11 to 9) and C (29 to 26). Thus, the frequency should be less for B and C, and more for A. The idea is to have each tile evaluate its own maximum service rate (q_{target}). Because tile A is experiencing the highest load, it has the highest q_{target}. Each tile then selects A's q_{target} to use as the reference queue occupancy (q_{ref}).

Table 3. Architectural parameters.

Parameter	Value
Processor clock	Two-way, 1-GHz, 7 stage pipelining
Issue/decode/commit width	2/2/2 instructions per cycle
L1 data and instruction caches	32-Kbyte, 4-way, 32-byte blocks, 1-cycle latency
L2 cache	None
Memory	20 cycles
Network topology	2D mesh
Channel width and flit size	256 bit/256 bit
Link traversal	1-cycle latency between hops 3-cycle network injection/ejection
DVFS transition and setup delays	73.3 ns/MHz, 171 ns/2.86 mV

tiles' original q_{target}s are lower than q_{ref}.

Estimating queue occupancies

Because parallel applications have vastly different granularity ranges, to estimate the occupancy, or workload, of a particular tile relative to that of another, we had to augment the task queue information with a normalizing load factor. This has two strong advantages. First, the controller can identify and account for longer-running threads and can thus differentiate between lightly and heavily loaded tiles. Second, different programs can use the same hardware controller because the program carries information about the weight of each of its threads (which the compiler or programmer has set), thus normalizing that weight to the hardware.

The controller associates a normalizing load factor (0 to 1,000) with each thread, which either the compiler or programmer provides as an estimate of the thread's lifetime or its load on the processor. Various methods are suitable for obtaining load factors, including regression-based tools that model performance.[14,15] For an application such as Quicksort, we needed only a simple model based on the input argument size and requiring only one multiply. The model yielded only a 2 percent error in estimating runtime over various input distributions. For Othello (a game-playing algorithm), a linear model with a single multiplication led to a 7 percent error for random board configurations. Given that our technique requires only relative accuracy in estimating thread runtime, we could simplify these models even more to reduce their processing overhead. Queue occupancies at each tile are then the summation of the normalizing load factors in the tile's task queue.

Experimental results

To evaluate dist-PID and local-PID, we coded five multithreaded benchmarks. Two are kernels that are aggressively multithreaded to tax our technique (recursive Quicksort and Othello). Three others are SPEC (http://www.spec.org) benchmarks (equake, twolf, and mcf) that we manually partitioned.

We generated our results using a multiprocessor simulator that is a modification of Xtrem[16]—a validated SimpleScalar ARM[7] simulator. Our modifications added support for multiprocessing and networking, as well as additional power modeling for new modules. We modeled the chip's interconnect power using Orion[17] (with a nominal voltage of 2.08 V). For DVFS, processors operated at frequencies between 100 MHz (0.45V) and 1 GHz (2.08V).

We modeled a 16-core CMP with the architectural parameters in Table 3, scheduling threads with a simple heuristic policy:

- Look for free processors.
- If none are available, schedule onto the processor with the lightest load.

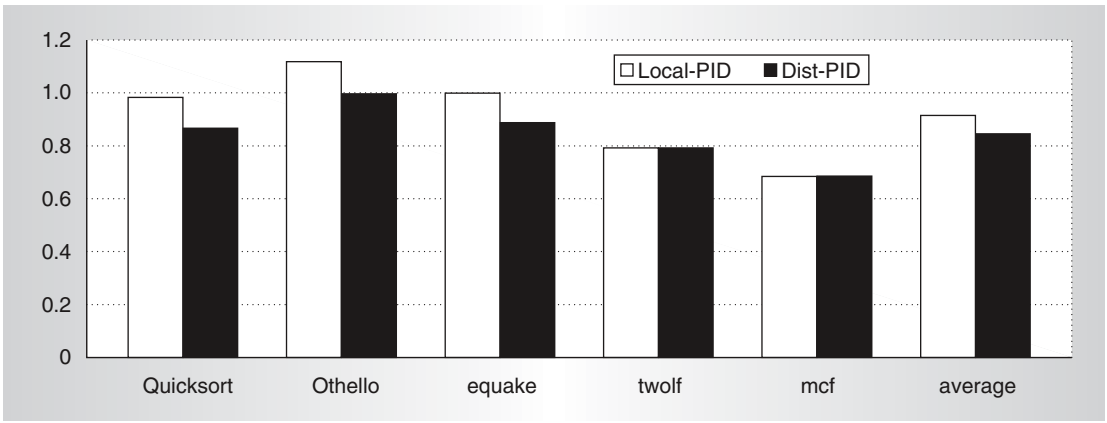

Figure 6. Energy-delay product with local-PID and dist-PID. We normalized both schemes against a baseline (EDP of 1.0) in which a processor shuts off the tile when its task queue is waiting for threads. In all cases, dist-PID equaled or outperformed local-PID.

The processor cores broadcast queue occupancies and load factors every 2,500 cycles while carrying out DVFS (if needed) every 50,000 cycles.

Savings in energy-delay product. To evaluate the efficiency of dist-PID and local-PID schemes, we normalized them against a baseline scheme—a processor with no DVFS but that "tile gates" a tile when all the threads in a tile's task queue are stalled (waiting for other threads) or when the tile has no threads. During this time, the processor shuts off the tile, and thus it consumes no dynamic power.

Figure 6 shows the EDP with normalized dist-PID and local-PID for the five benchmarks. Local-PID uses a q_{ref} of only 300, a third of the maximum expected queue occupancy.[2] Overall the local scheme had 86 percent of the baseline's energy consumption but increased runtime by 7 percent, producing an EDP that is 97 percent of the baseline. By comparison, dist-PID saved 20 percent of energy but increased runtime only 6 percent, giving an EDP of 85 percent. Othello is problematic for both schemes, since it has long-running threads punctuated with short bursts where it launches many threads. This combination makes it difficult to figure out the critical path. Quicksort, though, maps well to dist-PID: The size of the input arguments proved a good proxy for computation complexity.

Sensitivity to load-factor variation. Figure 7

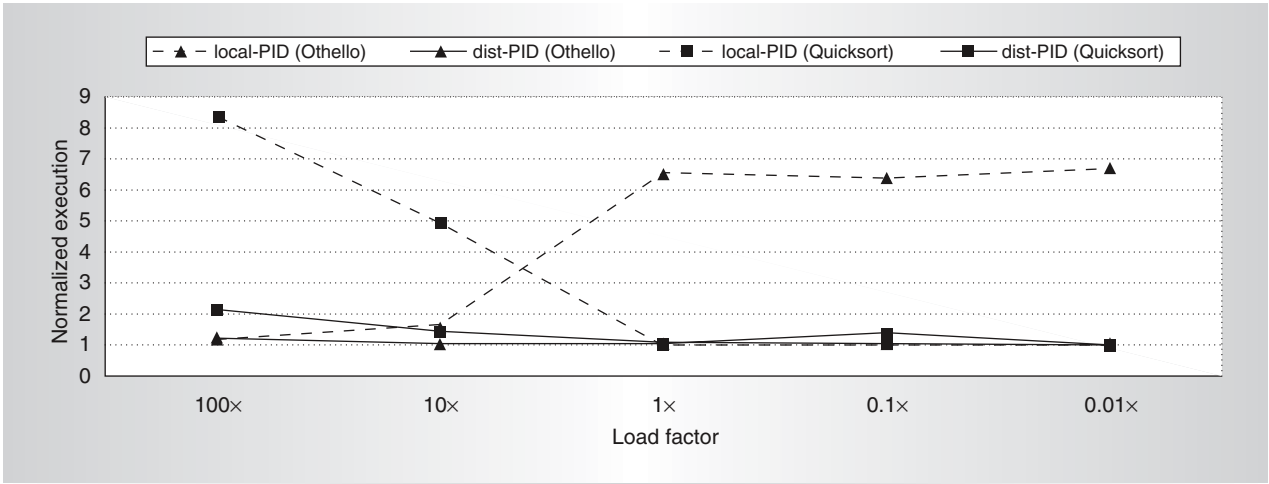

Figure 7. Runtime variations with varying normalizing load factor. Again, dist-PID outperformed local-PID.

Dynamic Voltage and Frequency Scaling Schemes

Dynamic voltage and frequency scaling (DVFS) is a common technique for improving energy efficiency in processors. DVFS schemes vary according to level of dynamism, either online or offline, and formalism. Most existing DVFS schemes are offline,[1-4] and typically use profiling to do offline analysis. Either a compiler[1,4] or a binary editor[3] then writes the DVFS configuration into the application. The effectiveness of such a profile-based scheme depends on the quality of the profile. Our schemes are online and thus require no profiling information. Instead, they respond to workload changes at runtime.

In formalism, existing DVFS schemes range from purely heuristic to formal, analytic schemes. Although many offline schemes take mathematical optimization-based formal approaches,[1,2,4] nearly all existing online DVFS schemes are heuristic-based.[5-7] At runtime, the scheme enables monitoring of certain processor metrics, such as cache miss rate[6] or queue occupancy.[5,7] The scheme then compares these metrics to the threshold values and applies some rules according to the comparison results. Currently, the best known online DVFS scheme for a multiple clock domain multiprocessor is the AttackDecay algorithm by Semeraro et al.[7] Heuristic schemes impose significant limitations, however. First, it is not analytically clear how to improve them and thus make DVFS more effective. Second, the trial-and-error tuning process for parameters is time-consuming. Third, it is generally hard to scale the heuristics for large systems, since the rules and tuning effort required can grow exponentially.

Recently, researchers have increased efforts to apply control theory or other system theories in CPU design and control.[8,9] An example is the application of control theory to thermal control.[8] Previous control theoretic techniques address energy efficiency in processors,[10] but only for multimedia processors with predictable workloads. Our techniques, in contrast, are suitable for general workloads.

References

1. C-H Hsu and U. Kremer, "The Design, Implementation, and Evaluation of a Compiler Algorithm for CPU Energy Reduction," *Proc. Programming Language Design and Implementation* (PLDI 03), ACM Press, 2003, pp. 38-48.
2. J.R. Lorch and A.J. Smith, "Improving Dynamic Voltage Scaling Algorithm with PACE," *Proc. Sigmetrics*, ACM Press, 2001, pp. 50-61.
3. G. Magklis et al., "Profile-Based Dynamic Voltage and Frequency Scaling for a Multiple Clock Domain Microprocessor," *Proc. Int'l Conf. Computer Architecture* (HPCA 03), IEEE CS Press, 2003, pp. 14-25.
4. F. Xie, M. Martonosi, and S. Malik, "Compile-Time Dynamic Voltage Scaling Settings: Opportunities and Limits," *Proc. Programming Language Design and Implementation* (PLDI 03), ACM Press, 2003, pp. 49-62.
5. A. Iyer and D. Marculescu, "Power Efficiency of Multiple Clock Multiple Voltage Cores," *Proc. Int'l Conf. Computer-Aided Design* (ICCAD 02), IEEE CS Press, 2002, pp. 379-386.
6. D. Marculescu, "On the Use of Microarchitecture-Driven Dynamic Voltage Scaling," Workshop on Complexity-Effective Design (WCED) 2000; http://www.ece.cmu.edu/~dianam/conferences/wced00.pdf.
7. G. Semeraro et al., "Dynamic Frequency and Voltage Control for a Multiple Clock Domain Microarchitecture," *Proc. Int'l Symp. Microarchitecture* (Micro-35), IEEE CS Press, 2002, pp. 356-367.
8. K. Skadron, T. Abdelzaher, and M. Stan, "Control-Theoretic Techniques and Thermal-RC Modeling for Accurate and Localized Dynamic Thermal Management," *Proc. Symp. High Performance Computer Architecture* (HPCA 02), IEEE CS Press, 2002, pp. 17-28.
9. Q. Wu et al., "Formal Online Methods for Voltage/Frequency Control in Multiple Clock Domain Microprocessors," *Proc. Int'l Conf. Architectural Support for Programming Languages and Operating Systems* (ASPLOS 04), ACM Press, 2004, pp. 248-259.
10. Z. Lu et al., "Control-Theoretic Dynamic Frequency and Voltage Scaling," *Proc. Int'l Conf. Compiler, Architecture, and Synthesis for Embedded Systems*, ACM Press, 2002, pp. 156-163.

shows the execution time (normalized against the baseline) after varying the normalizing load factor of Othello and Quicksort from 100× to 0.01× for both schemes. For both applications, dist-PID is relatively resilient under normalizing load factor variation. Local-PID, however, is quite fragile. For Othello the performance of local-PID jumps quite suddenly at some point, when the normalizing load factor crosses over q_{ref}. Once the normalizing load factor consistently stays above q_{ref}, local-PID tries to preserve performance. If it is below q_{ref}, local-PID tries to save energy, without regard to the overall program state. This crossover point is not always foreseeable and not particularly predictable.

These results show that distributed, coordinated control improves EDP and is also largely more resilient because of the flexibility that intertile coordination affords. For Quicksort,

local-PID performs well when the normalizing load factors are overestimated—for example, when the list to be sorted is larger than what we tuned for. This tells the local-PID controller to try to preserve performance. When normalizing load factors are underestimated, though, performance skyrockets—much more than in dist-PID. However, if the load factor is tuned for a large Quicksort, a smaller one will take a massive hit in performance. Common sense would thus be to lean toward tuning for smaller Quicksorts. Taken to the extreme, only very small Quicksorts would be eligible for energy savings, which misses the point of DVFS.

Our work shows that formal techniques for managing power-performance tradeoffs are effective on several types of high-performance processors—both single-core MCD processors and multicore CMPs. Although these individual solutions are appealing and show substantial EDP improvements for many applications, the future of this work lies in designing and implementing multiloop control techniques that coordinate several metrics at once, using a variety of policies and mechanisms. We are working toward such implementations and believe that they offer great promise for allowing future microprocessors to actively manage their power, performance, and thermal goals. MICRO

Acknowledgments

We thank David Albonesi and his research group for their MCD simulator, which we used as a base for some of our simulation infrastructure. Thanks also to Julia Chen and Kevin Ko for their contributions to the CMP architecture and simulator used in portions of this work. We also thank Diana Marculescu, Anoop Iyer, Greg Semeraro, and YongKang Zhu for helpful discussions during the early development of this work.

This research was supported by NSF grants CCR-0086031 (ITR) and CNS-0410937. Martonosi's work is also supported in part by Intel, IBM, and SRC.

References

1. P.Juang et al., "Coordinated, Distributed, Formal Energy Management of Chip Multiprocessors," *Proc. Symp. Low-Power Electronics and Design* (ISPLED 05), IEEE Press, 2005, pp. 127-131.
2. Q. Wu et al., "Formal Online Methods for Voltage/Frequency Control in Multiple Clock Domain Microprocessors," *Proc. Int'l Conf. Architectural Support for Programming Languages and Operating Systems* (ASPLOS 04), ACM Press, 2004, pp. 248-259.
3. Q. Wu et al., "Voltage and Frequency Control with Adaptive Reaction Time in Multiple-Clock-Domain Processors," *Int'l Symp. High-Performance Computer Architecture* (ISCA 05), IEEE CS Press, 2005, pp. 178-189.
4. R.V. Hogg and A.T. Craig, *Introduction to Mathematical Statistics*, 5th ed., Prentice Hall, 1995.
5. B.C. Kuo, *Automatic Control Systems*, 7th ed., Prentice Hall, 1995.
6. K.J. Astrom and B. Wittenmark, *Adaptive Control*, Addison-Wesley, 1995.
7. D. Burger and T.M. Austin, "The SimpleScalar Tool Set, Version 2.0," *Computer Architecture News*, June 1997, pp. 13-25.
8. D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A Framework for Architectural-Level Power Analysis and Optimization," *Proc. Int'l Symp. Computer Architecture* (ISCA 00), IEEE CS Press, 2000, pp. 83-94.
9. G. Semeraro et al., "Energy Efficient Processor Design Using Multiple Clock Domains with Dynamic Voltage and Frequency Scaling," *Proc. Int'l Symp. High-Performance Computer Architecture* (HPCA 02), IEEE CS Press, 2002, pp. 29-40.
10. G. Magklis et al., "Profile-Based Dynamic Voltage and Frequency Scaling for a Multiple Clock Domain Microprocessor," *Proc. Int'l Conf. Computer Architecture* (ISCA 03), IEEE CS Press, 2003, pp. 14-25.
11. G. Semeraro et al., "Dynamic Frequency and Voltage Control for a Multiple Clock Domain Microarchitecture," *Proc. Int'l Symp. Microarchitecture* (Micro-35), IEEE CS Press, 2002, pp. 356-367.
12. E. Talpes and D. Marculescu, "A Critical Analysis of Application-Adaptive Multiple Clock Processors," *Proc. Int'l Symp. Low Power Electronics and Design* (ISLPED), ACM Press, 2003, pp. 278-281.
13. P. Juang et al., "Hardware-Modulated Parallelism in Chip Multiprocessors," tech.

report, Princeton Univ., Dept. Electrical Eng., 2005.
14. A. Muttreja et al., "Automated Energy/Performance Macromodeling of Embedded Software," *Proc. 40th Design Automation Conf.* (DAC 04), IEEE CS Press, 2004, pp. 99-102.
15. A. Muttreja et al., "Hybrid Simulation for Embedded Software Energy Estimation," *Proc. 42nd Design Automation Conf.* (DAC 05), ACM Press, 2005, pp. 23-26.
16. G. Contreras et al., "XTREM: A Power Simulator for the Intel Xscale Core," *Proc. Conf. Languages, Compilers, and Tools*, ACM Press, 2004, pp. 115-125.
17. H.-S. Wang et al., "Orion: A Power-Performance Simulator for Interconnection Networks," *Proc. Int'l Symp. Microarchitecture* (Micro-35), IEEE CS Press, 2002. pp. 294-305.

Qiang Wu is a PhD candidate in Princeton University's computer science department. His research interests include power-aware microprocessor design and control. Wu has an MS in electrical engineering from the University of Sydney and a BS in electrical engineering from Southeast University in China. He is a student member of the IEEE and ACM.

Philo Juang is a graduate student at Princeton University. His research interests include chip multiprocessors and sensor networks. Juang has a BSEE from the University of Virginia. He is a student member of IEEE.

Margaret Martonosi is a professor of electrical engineering at Princeton University. Her research interests include computer architecture and the hardware-software interface, with particular focus on power-efficient systems and mobile computing. Martonosi has a PhD and an MS from Stanford University, and a BS from Cornell University, all in electrical engineering. She is a senior member of IEEE and a member of ACM, where she is the vice chair of ACM SIGARCH.

Li-Shiuan Peh is an assistant professor of electrical engineering at Princeton University. Her research interests include interconnection networks and parallel computer architectures. Peh has a PhD in computer science from Stanford University. She is a member of the IEEE and ACM.

Douglas W. Clark is a professor of computer science at Princeton University. His research interests include computer architecture, low-power techniques, and clocking and timing in digital systems. Clark has a PhD in computer science from Carnegie-Mellon University and a BS in engineering and applied science from Yale University.

Direct questions and comments about this article to Margaret Martonosi, Dept. of Electrical Engineering, Princeton University, Princeton, NJ 08544-5263; mrm@princeton.edu.

For further information on this or any other computing topic, visit our Digital Library at http://www.computer.org/publications/dlib.

Coming in *IEEE Micro*

November–December 2005

Guest Editors

Sarita Adve, University of Illinois, Urbana-Champaign

Pia Sanda, IBM

Reliability-Aware Microarchitecture

The industry must develop novel microarchitectural approaches to build reliable and dependable computers out of unreliable and unpredictable elements. This issue focuses on error-tolerant microarchitectures, low-overhead spatial or temporal redundancy techniques, and other strategies for enhancing reliability.