Flicker: A Dynamically Adaptive Architecture for Power Limited Multicore Systems[*]

Paula Petrica[†] Adam M. Izraelevitz
David H. Albonesi
Computer Systems Laboratory
Cornell University
Ithaca, NY 14853
{pp234,ami23,dha7}@cornell.edu

Christine A. Shoemaker
School of Civil and Environmental Engineering
Cornell University
Ithaca, NY 14853
cas12@cornell.edu

ABSTRACT

Future microprocessors may become so power constrained that not all transistors will be able to be powered on at once. These systems will be required to nimbly adapt to changes in the chip power that is allocated to general-purpose cores and to specialized accelerators.

This paper presents *Flicker*, a general-purpose multicore architecture that dynamically adapts to varying and potentially stringent limits on allocated power. The Flicker core microarchitecture includes deconfigurable *lanes*–horizontal slices through the pipeline–that permit tailoring an individual core to the running application with lower overhead than microarchitecture-level adaptation, and greater flexibility than core-level power gating.

To exploit Flicker's flexible pipeline architecture, a new online multicore optimization algorithm combines reduced sampling techniques, application of response surface models to online optimization, and heuristic online search. The approach efficiently finds a near-global-optimum configuration of lanes without requiring offline training, microarchitecture state, or foreknowledge of the workload. At high power allocations, core-level gating is highly effective, and slightly outperforms Flicker overall. However, under stringent power constraints, Flicker significantly outperforms core-level gating, achieving an average 27% performance improvement.

1. INTRODUCTION

Microprocessors are becoming increasingly heterogeneous and power constrained. Future microprocessors will likely consist of a number of general-purpose cores augmented by a range of accelerators that are tailored to executing particular types of applications. Moreover, these microprocessors may become so power constrained that not all transistors will be powered on at once (the so-called *dark silicon* problem [17, 44]), which requires the system to be nimble in terms of power allocation. During time quanta when few applications can be mapped to the accelerators, the general-purpose cores may be allocated full power. At high accelerator usage, the general-purpose core power allocation may be severely capped, and the hardware must flexibly adapt to these constantly changing power budgets while ideally maximizing throughout at each power point. This paper addresses the problem of optimizing the performance of general-purpose multicore architectures that must efficiently adapt to varying, and at times highly stringent, power allocations.

While DVFS is currently used for dynamic power management, the push to aggressively scale voltages, perhaps into the near-threshold regime, is rendering DVFS largely ineffective. Core-level gating, in which the core voltage domain is gated in order to save power, has emerged as a viable alternative to DVFS due to its ease of implementation and simple control requirements. Each core is placed in a separate voltage domain that can be gated through the use of power transistors, and the number of cores that can be enabled under the given power budget can be easily determined. However, core-level gating has three major drawbacks: (1) Its coarse granularity of power control makes it difficult to precisely match given power requirements; (2) By uniformly allocating power to the powered on cores, it fails to adapt the hardware to the characteristics of the workload; and (3) It requires the operating system scheduler to adapt to a varying number of available cores.

Microarchitecture adaptation [1] lies at the other end of the power-gating spectrum. Here, microarchitecture resources are dynamically adapted at fine-grain within each core in order to match application requirements, which addresses the limitations of core-level gating. This approach has been proven effective in technologies where dynamic power dominates, and therefore simple gating approaches (such as disabling control and clocks to a portion of a resource) can be readily implemented. However, microarchitecture adaptation is more challenging in current microprocessors that have a prominent leakage power component and tens of cores. To address leakage power, potentially dozens of fine-grain voltage domains must be implemented within each core's pipeline. Moreover, with tens of cores, the control algorithm must determine the combination of dozens of power knobs on each of tens of cores that maximizes throughput under the given power budget, a complex task that is difficult to perform quickly enough (within a few to tens of milliseconds) given the short duration of an operating system time

[*]This research was supported by NSF grants CCF-0916821, CCF-0811729, and CNS-0708788; Cornell Engineering Learning Initiatives; and equipment grants from Intel.

[†]Currently with Intel Corporation, Hillsboro, OR.

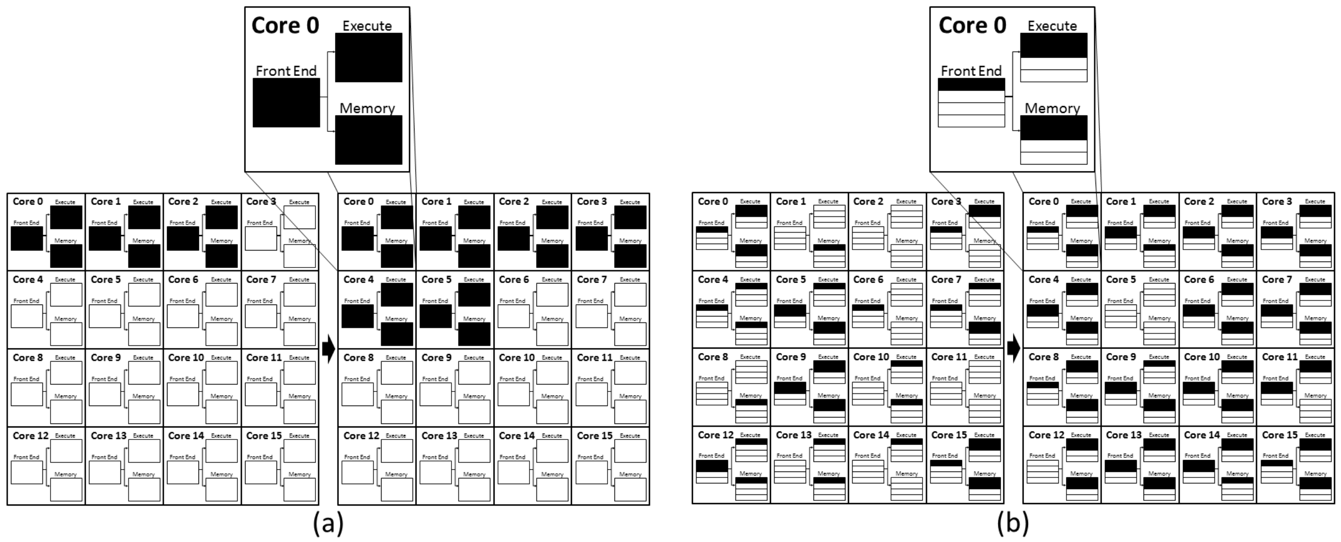


Figure 1: Operation of (a) core-level gating versus (b) Flicker for two different intervals, the second of which has a more stringent power constraint.

slice (tens to hundreds of milliseconds).

These drawbacks are addressed using a new approach proposed in this paper that is based on two observations: (1) Applications vary in the pipeline width that best balances performance and power consumption; and (2) Often one section of the pipeline is out of balance with respect to the others, which creates slack in one or more pipeline sections. The proposed approach, called *Flicker*, divides each pipeline section within each core into deconfigurable *lanes*–horizontal slices through the pipeline–that permit tailoring an individual core to the running application with lower overhead than microarchitecture adaptation. Each lane in a Flicker core constitutes an individually controllable power domain. In all, there are nine power domains within each pipeline–three within each of the front-end, execute back-end, and memory sections of the pipeline–requiring six power-gating circuits (plus three more, if needed, for core-level shutdown), which is far fewer than microarchitecture adaptation.

Figure 1 compares the operation of core-level gating and Flicker for an illustrative 16 core system under two operating intervals with different power allocations. For core-level gating, a number of cores are entirely shut down depending on the power constraint. For Flicker, all cores remain operational, but lanes are power gated within each core depending on application characteristics. In this manner, Flicker provides finer grain allocation of power to applications according to their need for particular pipeline resources.

Despite its relative simplicity compared to microarchitecture adaptation, a Flicker system with tens of cores presents a far more challenging global optimization problem than core-level gating. Given a power budget, the Flicker global controller must determine the combination of powered lanes within different cores that maximizes performance, and converge reasonably quickly to limit overhead. To address this problem, Flicker's online optimization approach efficiently finds a near-global-optimum configuration of lanes without requiring offline training, microarchitecture state, or foreknowledge of the workload. Compared with prior approaches, Flicker's approach is purely online, and–through reduced sampling techniques, application of response surface models to online optimization, and heuristic online search–converges quickly to a near-optimal solution. Due to its purely online "black box" approach, the Flicker control algorithm adapts on-the-fly to different machine microarchitectures and to machines that run a wide variety of applications.

The rest of this paper is organized as follows. In the next section, related work is discussed. The Flicker core microarchitecture is presented in Section 3 followed by a description of the runtime control approach in Section 4. The evaluation methodology is discussed in Section 5 and results in Section 6. The paper concludes in Section 7.

2. RELATED WORK

Related work in microarchitectural approaches to dynamic power management is first presented, followed by a discussion of global optimization approaches that attempt to determine the hardware configuration that maximizes performance given a power constraint.

2.1 Power Efficient Microarchitecture

DVFS is widely used for dynamic power management, and prior work explores adapting this approach to multicore processors. Isci *et al.* [24] develop the maxBIPS DVFS algorithm for multicore systems. Sharkey *et al.* [40] extend this work by exploring both DVFS and fetch toggling, and design tradeoffs such as local versus global management. Bergamaschi *et al.* [4] conduct further work on maxBIPS and compare its discrete implementation to continuous power modes. While DVFS is effective today, the movement towards processors with razor thin voltage margins (*e.g.*, near-threshold computing) and the need for potentially tens of dynamically scalable voltage domains in large-scale multicore systems threatens to render DVFS largely ineffective.

A viable alternative to DVFS is core-level gating, in which each core is placed in a separate domain within which the supply voltage can be individually gated. Core-level gating is widely implemented in processors (*e.g.*, [28]) and researchers (*e.g.*, [32]) have developed techniques for determining when to shut down cores given a power constraint.

A variety of proposed approaches dynamically adapt core microarchitecture resources to match workload requirements. Adaptation techniques that gate resources throughout the pipeline [1, 14, 22, 35] have been shown to be effective at reducing dynamic power, but are not as readily applicable to controlling leakage due to the many power-gating circuits required within the core pipeline. Moreover, with possibly dozens of power knobs within each core and tens of cores in a multicore system, finding the combination of knobs that maximizes throughput within the specified power constraint is challenging, given that the operating system may temporally reschedule threads (and change power allocations) every time slice (tens to hundreds of milliseconds). Others [2, 3, 9, 16, 18, 19, 26, 45] propose to gate only a subset of the core hardware, which limits the power savings.

In a prior workshop paper [34], we propose a lane-based microarchitecture for fault tolerance. When a lane is deconfigured due to a wear-out defect, lanes in other pipeline sections may be deconfigured to rebalance the pipeline. The saved power is transferred to other functions to boost performance. Flicker's application is quite different and its optimization approach is not covered in this prior work.

2.2 Global Optimization Algorithms

Several offline optimization approaches have been proposed [10, 30, 31, 48], some for microarchitecture optimization, but have limited applicability to online adaptation.

Lee and Brooks [30] propose regression techniques for design space exploration, and suggest that these techniques can be applied to offline application profiling. Lee *et al.* [31] propose an offline multiprocessor model to reduce simulation time. These approaches target offline simulation rather than online adaptation where the timescale is more constrained. The offline design space exploration work of Yi *et al.* [48] relates to Flicker's application of Fractional Factorial design.

Chen *et al.* [10] present a framework for offline resource demand estimation. While they mention that the approach could be used for online dynamic resource management, they do not present online profilers to make this feasible. Moreover, the framework is only applicable to uniprocessors.

Flicker's online approach bears similarity to the work of Lee and Brooks [29], who propose an analysis framework–using sampling, splines, and the Genetic Algorithm–to reduce the computation required to evaluate whether online adaptivity should be pursued. They perform an offline analysis of the potential of online adaptivity, using regression models instead of simulations to explore the design space. Flicker's optimization approach uses sampling, prediction, and optimization to solve a different problem: strictly online, runtime adaptation of the Flicker microarchitecture.

Dubach *et al.* [15] perform online adaptation using offline training of a single core model. Flicker tackles the problem of optimizing over tens of cores, for which straightforward decisions such as maximizing one core's response (the approach in [15]) are ineffective. Moreover, Flicker's black box approach makes it suitable for machines that run applications that are much different than the training set.

Teodorescu and Torrellas [43] consider global power management in the presence of process variations and propose using linear optimization to determine how to allocate power to cores. However, as shown later, linear optimization does not perform well for adapting the Flicker microarchitecture.

Bitirgen *et al.* [5] and Ipek and Martinez [23] use machine learning (as opposed to surrogate response surfaces) for adaptation of shared caches and off-chip memory bandwidth. Their artificial neural networks approach requires calibration for each runtime application, as well as region specific counters and usage histograms. Flicker's use of Radial Basis Function surrogate models eliminates calibration and requires only two system measured responses (power and throughput).

SimFlex uses sampling theory to measure minimal yet representative samples of benchmarks to reduce multiprocessor simulation time [46]. Flicker's application of sampling theory solves a very different sampling problem: online workload characterization. Ponomarev *et al.* [35] address the dynamic online problem, but without the use of statistical sampling. However, since there is no global optimization, their approach only requires sampling current configurations.

In prior work (*e.g.*, [33, 36, 37, 38]), we use Radial Basis Functions to solve global optimization problems with RBF surfaces of dimension of up to 200. However, this prior work used offline analysis. Flicker's search method in conjunction with the RBF (which involves a combination of an RBF surface with a heuristic related to [36]) is an online approach that requires a solution within a few milliseconds.

3. FLICKER MICROARCHITECTURE

The Flicker core microarchitecture bridges the gap between fine-grain microarchitecture adaptation (which incurs high overheads) and core-level gating (which is coarse grain and cannot adapt to individual thread behavior). The pipeline is power-gated at the granularity of horizontal pipeline slices (lanes), which permits rapid adaptation of the width of different pipeline regions. While cores are homogeneous in design, they can be dynamically reconfigured into a heterogeneous multicore system that meets power constraints.

Lanes are implemented within three decoupled pipeline regions: Front End (FE - fetch, decode, ROB, rename, dispatch), Execute (EX - issue queues, register files, functional units) and Memory (MEM - load and store queues), each of which has four lanes. Two lanes remain powered on except when power gating the entire core. This power domain includes non-redundant structures, such as the Integer Multiplier, which must remain powered on for proper operation.

Each pipeline lane includes a sub-bank of the associated queues, even though they are not technically part of the pipeline "width." As the peak bandwidth of a pipeline region is reduced by deconfiguring a lane, the buffering requirements (and the issue window requirements) are reduced commensurately. Thus, the associated queues within the region can be downsized to save power.

Lane-based deconfiguration requires both *physical gating* and *logical correctness* mechanisms. The physical gating mechanisms include sleep transistors that power gate each of the blocks within a lane. Supply voltage levels must be slightly increased to account for the voltage drop across the sleep transistors, and decoupling capacitance also increased in order to reduce voltage fluctuations in the power grid [25].

The logical correctness mechanisms (which are always powered) ensure proper pipeline operation when lanes are deconfigured. For instance, deconfiguration of a front end lane requires preventing instructions from being slotted into the deconfigured lane. For the back end, a functional unit associated with a deconfigured lane is marked as perpetually in use within the issue queue selection logic.

Flicker's array structures leverage techniques previously proposed for reliability and power management. For the Fetch Queue and the ROB, Bower *et al.* [6] develop circular structures with spares that can be deconfigured at a fine-grain, per-entry level by feeding fault information into the head and tail pointer advancement logic. Flicker implements these techniques at a coarse-grain by banking the queues and deconfiguring an entire bank, thus requiring a map of only four bits, one for each bank that can be deconfigured. Unlike [6], the microarchitecture does not require spare banks; therefore, the buffer size is also updated when banks are deconfigured or reenabled. The issue queue is based on Dropsho *et al.* [14], who demonstrate a coarse-grain partitioned RAM/CAM based issue queue that dynamically adapts its size to program demands.

Physical gating of deconfigured functionality within a lane leverages power-gating techniques proposed to reduce leakage power and to implement microprocessor deep sleep states, such as C6. Intel Core i7 microprocessors implement power-gating transistors to shut off idle cores [28], and a number of designers have proposed a variety of power-gating techniques [11, 25, 41]. Power-gated functional blocks are aggregated into six individually controllable power-gated lanes, two for each of the FE, EX, and MEM regions (plus three additional power-gating circuits that are used, if needed, for core-level gating). The logical correctness circuitry is always powered to ensure correct pipeline operation.

Sleep transistor area overhead estimates vary from 2-6% depending on the implementation, size of clusters, and technology node [27, 41]. Moreover, advanced sleep transistor sizing algorithms can considerably reduce the area overhead [11]. In addition to the sleep transistors, area overheads are introduced by additional decoupling capacitance that is incorporated to reduce voltage fluctuations, resulting in a total estimated overhead of 15% [25]. While dynamic power is slightly increased (by approximately 2% according to [25]), static power can be reduced by 90%.

4. RUNTIME CHARACTERIZATION AND OPTIMIZATION

Flicker periodically characterizes application behavior and determines the combination of powered lanes that maximizes performance without exceeding the specified power constraint. The run time is divided into 100 ms time quanta comprising four phases: configuration sampling, surrogate surface fitting, optimization, and steady (Figure 2).

During the sampling phase, an application runs for short sampling periods, each of which has a different combination of enabled lanes. Since the cores are identical, sampling on all 32 cores occurs in parallel, but configurations are staggered to avoid global power overshoots. Enhanced sampling techniques (Section 4.2) significantly reduce the sampling time over full sampling and increase sampling accuracy. Next, performance and power surrogate functions are fitted to the sampled data (Section 4.3). Finally, using the surrogate functions, an optimization algorithm determines the combination of powered up lanes on the Flicker cores that maximizes global performance under the specified power constraint (Section 4.4). The system operates with this configuration for the remainder of the 100 ms interval. After this time, the process is repeated if the operating system schedules different threads or changes the power limit.

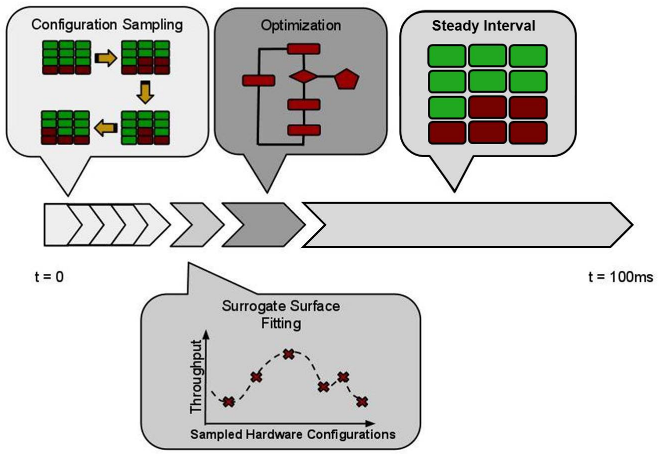


Figure 2: Online characterization and optimization–consisting of configuration sampling, surrogate surface fitting, and optimization–followed by steady-state operation.

4.1 Experimental Design Approach

The characterization of each running process is formulated as a multivariate statistical experimental design, which results in an empirical model that correlates hardware resource allocation with power and performance. This design is used by the Flicker optimization algorithm to maximize global performance within the power budget that is allocated for each 100 ms interval.

The two types of variables in a multivariate optimization procedure are *responses* and *factors*, where the responses are observed (or sampled) output values dependent on the values taken on by the factors. The response variables for this experimental design are the throughput and power consumption of the running thread, and the goal is to characterize the effect of different lane allocations on these variables in order to obtain an optimal resource allocation. The factors are the controlled independent variables that affect the response of the system. In Flicker, the three pipeline regions (FE, BE, and MEM) are the factors of the experiment, denoted as X_1, X_2, and X_3, respectively. Each of the factors can take on three different levels (four active lanes - fully provisioned, three active lanes, and two active lanes). Thus, there are $3^3 = 27$ configurations for each core.

4.2 Optimized Sampling Techniques

In a Full Factorial design, depicted in Figure 3 (left), all 27 core configurations are sampled and their effect on the response variables is measured. The design space is represented as a cube, with the edges representing the levels of the factors, the corners indicating the high and low values of each factor, and the dots marking the sampled configurations as dictated by the experimental design. The large number of samples needed for a Full Factorial design limits its usefulness in runtime applications, as a large portion of the OS time slice may be spent sampling suboptimal configurations. To address this issue, two well known methods are explored that reduce the cost of experimentation while estimating response surface parameters with high precision: Box-Behnken and Fractional Factorial designs.

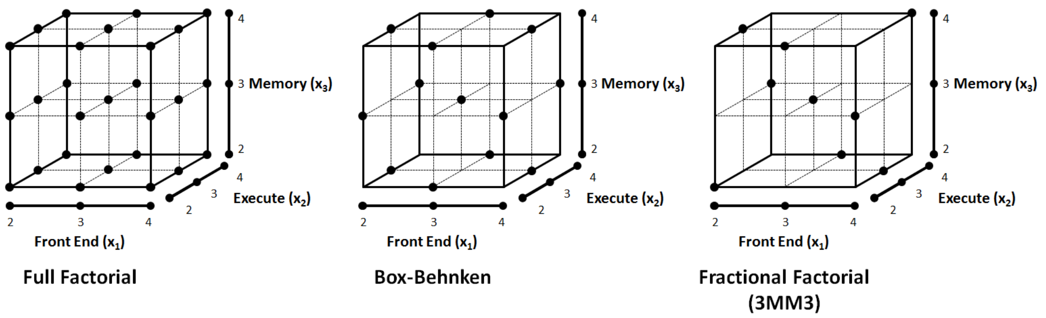


Figure 3: Sampled core configurations for the Full Factorial, Box-Behnken, and Fractional Factorial designs.

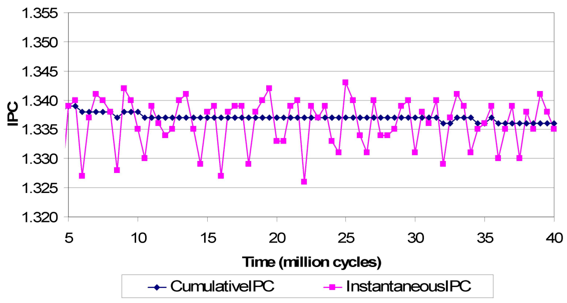


Figure 4: Variation in IPC during execution of *applu*, demonstrating the variation in application behavior among sampling points.

4.2.1 *Reducing Sampling Time: Box-Behnken and Fractional Factorial Designs*

Box-Behnken design [7] and Fractional Factorial design [47] are based on the sparsity-of-effects principle: the highest contributors to responses are likely the main (single factor) and low-level (two factor) interactions. Moreover, they are both balanced and orthogonal for optimal efficiency.

Box-Behnken design selects configurations to sample that are at the midpoints of the edges of the design space and at the center, as shown in Figure 3 (center). This design requires at least three factors with at least three levels each. Since the effect of the factors on the dependent variable is not linear, the Box-Behnken design is attractive because it allows for quadratic response surface fitting. The number of samples required for a Box-Behnken design is $2k(k-1)+C$, where k represents the number of factors and C the number of center points. There are three factors for a Flicker pipeline, and the (3, 3, 3) centerpoint is included, which results in 13 required samples.

A class of Fractional Factorial designs called 3^{k-p} design, where k is the number of factors and 3 represents the number of levels of each factor, is also considered. A 3^{k-1} design reduces the number of samples by three, and 3^{k-2} reduces the number of samples by nine. Since it is unfeasible to construct an accurate response surface for three factors using only three samples, a 3^{k-1} design, referred to as 3MM3 (“3 Minus Modulus 3”) and shown in Figure 3 (right), is considered. The 3MM3 design requires only nine samples compared to 13 for the Box-Behnken design.

4.2.2 *Improving Sampling Accuracy: Replicated Sampling*

In online sampling, the measured responses are not only a function of the change in microarchitecture but also of the temporal fluctuations in application behavior. If these temporal fluctations are large, the samples become “noisier” and thus the response surfaces are less accurate.

This noise, depicted in Figure 4 for the *applu* benchmark, can be addressed by increasing the size of the samples, but doing so increases sampling overhead by reducing the length of the steady interval. In an alternative *replicated sampling* approach, shown in Figure 5, multiple smaller samples (replicates) of each configuration are taken at different points and then averaged.

In this example, each of three original 1 ms sample periods is split into three sub-periods with a duration of 1/3 ms. Samples corresponding to the same configurations are evaluated at slightly different points in the application and their responses averaged, which filters high frequency noise. As the number of replicates increases, high frequency noise is more effectively filtered, but the sample size decreases, which has two drawbacks. First, microarchitectural events such as cache misses and branch mispredictions are less “smoothed out” by shorter samples. Second, the deconfiguration and reconfiguration overheads become more prominent with more frequent sampling. As is shown in Section 6, using eight replicates significantly improves sampling accuracy.

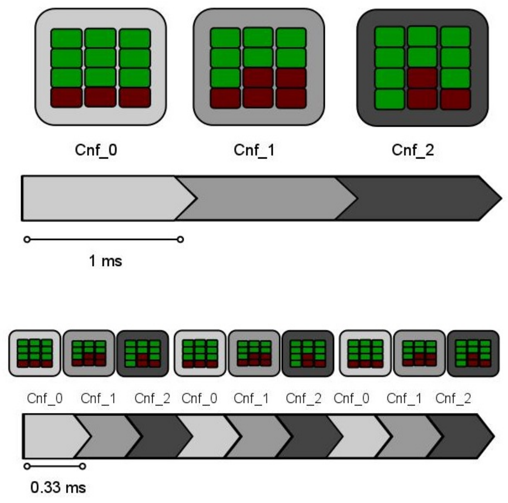


Figure 5: Typical sampling approach where each configuration is run once for a 1 ms period (top). Replicated sampling where each configuration is run S times, for $1/S$ ms sub-periods, and the results averaged over the S sub-periods (bottom).

4.3 Exploiting Radial Basis Functions for Online Optimization

A response surface model (or surrogate model) is an inexpensive approximation of computationally expensive functions for which an *a priori* description or formula is not available, and information can only be obtained through time-consuming direct evaluation of the functions. These models are particularly well-suited to approximating global throughput and power on a Flicker core, since they construct response functions from a subset of function evaluations obtained through reduced sampling. Two metamodels, $T(x_1, x_2, x_3)$ and $P(x_1, x_2, x_3)$, are constructed to approximate the throughput and power responses.

The simplest response surfaces that are frequently used for online architectural adaptation are first and second order polynomials (linear and quadratic functions). However, linear functions poorly predict the responses of the Flicker system (Section 6.2), and quadratic functions require a larger number of datapoints (sample points). Moreover, both first and second order polynomial functions are non-interpolating: the values on the response surface are not necessarily equal to the values obtained during sampling.

An interpolating model that places a Radial Basis Function (RBF) at each sampled point [20, 33, 36, 37, 38] overcomes the limitations of simpler response surfaces. Moreover, the approach requires no offline training and very little state: 45 unique matrix elements (integers) for 3MM3 and 91 for Box-Behnken, which are fixed and independent of the workload. Refer to the Appendix for a discussion of the RBF model. The performance of a RBF response surface is compared to simpler response surfaces in Section 6.2.

4.4 Global Performance Optimization Algorithm

Once metamodels T and P are constructed to fit each core, the optimization step finds the set of core configurations that maximizes performance while abiding by the provided power constraint. Since each core can be configured in 27 ways, an N core system will have a total of 27^N possible sets of core configurations, making runtime exhaustive search of the space impractical for even modest values of N. To permit the OS to have the flexibility to temporally reschedule threads within a reasonable time granularity (tens to hundreds of milliseconds), and given the time required for sampling, the optimization algorithm should converge to a near-optimal combination of enabled lanes with a runtime no longer than a few milliseconds.

Specifically, the optimization algorithm must solve the constrained integer global optimization problem of maximizing performance under a given power budget. In order to take fairness into account, the geometric mean throughput is chosen as the performance objective function:

$$f(\vec{x}) = \sqrt[N]{\prod_{i=1}^{N} \hat{T}_i(x_{1_i}, x_{2_i}, x_{3_i})} \qquad (1)$$

where N is the number of cores, $\vec{x}$ is a vector of size N consisting of the current configuration for each core, and $\hat{T}_i(x_{1_i}, x_{2_i}, x_{3_i})$ is the BIPS of the i^{th} core.

The objective function further has the constraint of meeting a certain power budget. Deb's constraint handling method [12] differentiates between feasible and infeasible (over power budget) solutions by penalizing configurations that consume more power than allowed, ensuring that infeasible solutions are never chosen over feasible solutions. The final function has the form:

$$F(\vec{x}) = \begin{cases} f(\vec{x}) & \text{if } g(\vec{x}) \leq maxPower \\ 1 - g(\vec{x}) & \text{if } g(\vec{x}) > maxPower \end{cases} \qquad (2)$$

where $g(\vec{x})$ is the constraint violation function and is defined as the current power consumption of the entire core: $g(\vec{x}) = \sum_{i=1}^{N} \hat{P}_i(x_{1_i}, x_{2_i}, x_{3_i})$.

The solution for the objective function is the vector $\vec{x}$, the configuration of each core that results in the best global performance. The solution vector consists of discrete rather than continuous variables, which makes it difficult to apply classical mathematical techniques such as derivative or limit-based methods.

Heuristic algorithms are effective in searching complex spaces, and their computational time can be adjusted by limiting the number of objective function evaluations at the expense of solution accuracy. Flicker's runtime system incorporates the Genetic Algorithm, which uses information gathered from past searches about an unknown space to bias future searches towards more useful subspaces. Each core configuration is encoded as one gene of a chromosome, and each gene takes the integer values 0 to $C-1$, where C is the number of possible configurations for each core. A combination of N genes form one chromosome of a population, where N is the number of cores.

Tournament selection with replacement is used to pick the mating pool, with two children from each parent pair created using single point crossover at the boundary of the genes and a high mutation probability. To prevent the algorithm from losing the current best solution, elitism is implemented by replacing a random child with the best parent. A variety of parameter values are explored offline and a desirability function [13, 21] constructed in order to find parameters that would optimize the algorithm over a variety of power constraints. This methodology produces the following parameters: a population size of 20 individuals, a crossover probability of 0.9, and a mutation probability of 0.6. The simulation is run for 25 generations (which corresponds to 500 objective function evaluations) as a compromise between algorithm accuracy and computation time, and consumes 1% of the 100 ms interval for a 32 core configuration.

5. EVALUATION METHODOLOGY

The evaluation methodology combines enhanced versions of SESC [39], Wattch [8], Cacti [42], and HotLeakage [49]. An extensive design space study is performed to determine a balanced baseline core microarchitecture whose parameters are shown in Table 1.

The simulated 32 core systems run 20 multiprogrammed workloads. Each workload is constructed by randomly selecting one of 13 SPEC CPU 2000 benchmarks (*applu*, *wupwise*, *mesa*, *mgrid*, *ammp*, *apsi*, *twolf*, *crafty*, *parser*, *vpr*, *vortex*, *bzip2*, and *gcc*) to run on each core. Each benchmark runs with the reference input set and is fast-forwarded five billion instructions, after which the workload is run for 100 ms, the length of a time quantum.

To evaluate the system under a variety of power cap scenarios, a nominal power value is determined by averaging the power consumption of the benchmarks and multiplying by the number of cores (32). The system is evaluated at

Table 1: Architectural parameters.

Front End	Branch Predictor: gshare + bimodal 64 entry RAS, 2KB BTB 128 entry ROB 4-wide fetch/decode/rename/retire
Execution Core	out-of-order, 4-wide issue/execute 80 Integer Registers, 80 FP Registers 32 entry Integer Queue, 24 entry FP Queue 32 entry Load Queue, 16 entry Store Queue 4 Integer ALUs, 1 Integer Mult/Div Unit 1 FP ALU, 1 FP Mult/Div Unit
On-chip Caches	L1 Instruction Cache: 8KB, 2-way, 2 cycles L1 Data Cache: 8KB, 2-way, 2 cycles L2 Cache: 1MB, private, 8-way, 15 cycles
Memory	200 cycle latency
Operating Parameters	1V Vdd 4.0 GHz frequency

eight different power constraints, from 90% to 55% of the nominal power. The baseline system uses core shutdown to meet the given power budget.

6. EXPERIMENTAL RESULTS

6.1 Replicated Sampling

The throughput estimation error of the sampled response for no replicates and up to eight replicates across all benchmarks is shown in Figure 6. For each configuration (combination of enabled lanes), the error is measured as the percent difference between the real response (result when running the configuration for the full 100 ms interval) and the average response of the replicated sample (1 ms total).

Two opposing effects must be balanced in order to obtain accurate samples. On the one hand, more replicates reduces high frequency noise by obtaining responses at different points of the benchmark execution. However, the smaller samples are more susceptible to noise caused by cache misses, branch mispredictions, and small software loops. As shown in Figure 6, sampling accuracy increases when increasing the number of replicates from none to eight. The spread between the 25th and 75th percentiles narrows considerably with eight replicates compared to no replication. The mean error is close to zero, indicating that sampling on average accurately captures the true response of the system. Power estimation shows similar results. The remainder of this work uses replicated sampling with eight replicates[1].

6.2 Surrogate Response Surfaces

A comparison of the accuracy of the surrogate surfaces for characterizing throughput and power is provided in Figure 7. The y-axis represents the percentage by which the predicted responses deviate from the real responses, and each box plot depicts statistics collected across all benchmarks. For each application, a response surface is built on the Full Factorial (27 points), Box-Behnken (13 points), and the 3MM3 (nine points) designs. Real (100 ms) responses are used instead of short samples for the observation points to isolate the effects of surrogate surface fitting from sampling error.

[1]Simply using no replication with shorter samples, *e.g.*, 0.1 ms, produces prediction errors in excess of 300%. Using longer samples slightly improves accuracy but at the cost of shortening the steady phase.

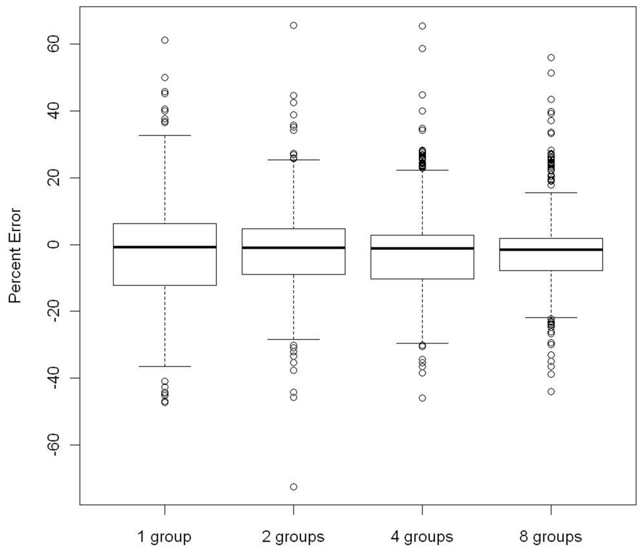


Figure 6: Box plots of throughput estimation error due to sampling for the Box-Behnken design. Each 1 ms sample period is split into 1, 2, 4, or 8 subperiods. The total number of points for each plot is the number of samples times the number of benchmarks. For replication, each sample point is the average of the replicates (1 ms total runtime).

Even when constructed using the Full Factorial design, the linear model fits the data the worst, with a residual percent error as high as 15-20% in both directions, meaning that responses are both overestimated and underestimated. The fitting of a quadratic surface to the Full Factorial observations reduces both throughput and power error. The Box-Behnken design yields very good results, with only a slight increase in the number of outliers.

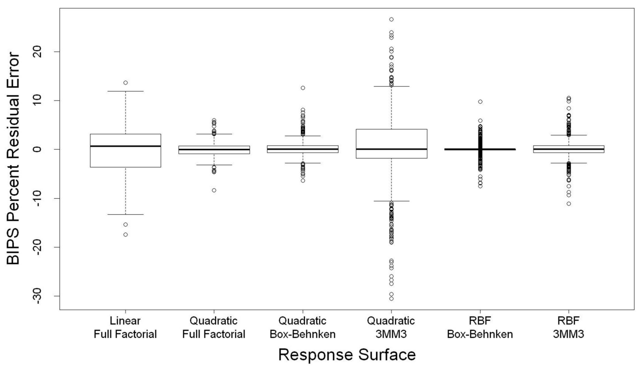
The prediction accuracy dramatically worsens when building the quadratic surface on only nine observation points (3MM3 design), since a quadratic response surface for three variables requires ten coefficients, which cannot be obtained using only nine samples. By eliminating quadratic terms, this effectively reduces the quadratic function to an almost linear one.

Since the RBF response surface is an interpolating model, the residual error is zero if the RBF surface is built using the Full Factorial design (not shown in the graph). The use of fewer observations points from the Box-Behnken design to create an RBF surrogate surface produces extremely accurate results, with a small spread and relatively few outliers. 3MM3 provides far more accurate results with an RBF rather than a quadratic surface, but accuracy degrades relative to the Box-Behnken design with RBF due to the fewer number of observation points.

All in all, the combination of Box-Behnken design, eight-way replicated sampling, and RBF surrogate surfaces provides accurate performance and power models for use during optimization with modest runtime overhead.

6.3 Global Optimization Algorithm

To evaluate the accuracy of Flicker's runtime characterization and optimization approach, a near-oracle optimiza-



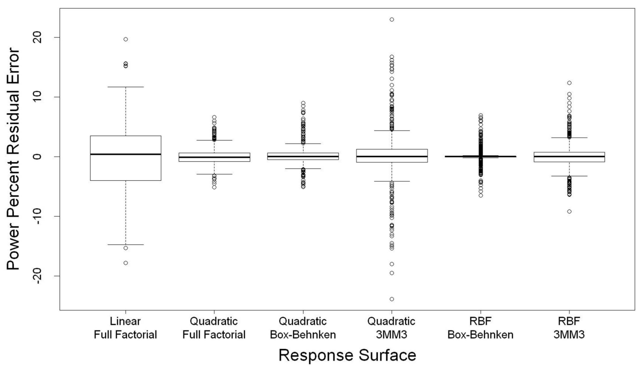


Figure 7: Box plots of percent residual error between the predicted and actual system responses for the combination of surrogate surface model and online sampling. The total number of points for each plot is the number of configurations (27, some of which may be predicted) times the number of benchmarks.

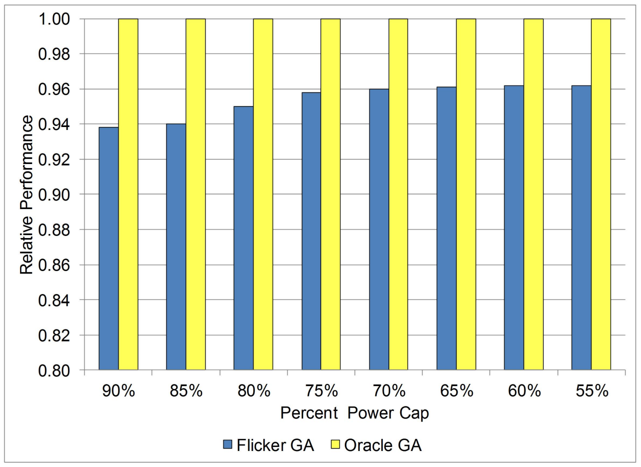


Figure 8: Comparison of Flicker's online optimization algorithm to the near-oracle offline algorithm at different power allocations, each of which is averaged over all workloads.

tion algorithm[2] is implemented that determines the steady phase configuration by: (1) Using the true 100 ms power and throughput responses to eliminate sampling error; (2) Using the Full Factorial design to eliminate errors due to response surface approximation; and (3) Running the Genetic Algorithm offline to reduce the error from limiting the runtime of the online algorithm.

As shown in Figure 8, Flicker's online algorithm closely matches the results of the near-oracle offline algorithm. Across all power caps, the worst case degradation is 6%.

Interestingly, Flicker's accuracy improves with increasingly stringent power constraints. At relaxed power caps, there are many possible solutions with a wide range of "goodness." Thus, the errors from sampling and surface fitting are more pronounced, because the Genetic Algorithm is free to pick any configurations as parents for the next generation.

At more stringent power caps, there are fewer viable solutions. Therefore, the space from which to choose individuals to reproduce (parents) is reduced, which affords several advantages: (1) The algorithm is more likely to choose the "fit" parents; (2) Sampling error is mitigated as some of the configurations with high sampling error may be filtered out for being over budget; and (3) Fewer iterations are required for convergence, which mitigates the effect of limiting the number of iterations. For these reasons, Flicker's accuracy improves as allocated power diminishes.

[2]An exhaustive search oracle algorithm requires decades of computation time for 16 or more cores.

6.4 Comparison with Core-level Gating

A quantitative comparison with core-level gating is challenging since fewer threads are operational compared to Flicker, where all cores are active. To make a fair comparison, for a given workload and power budget, the geometric mean throughput of the 32 applications on baseline fully provisioned cores is scaled by the fraction of cores that can be enabled under the specified power budget.

The performance of Flicker normalized to core-level gating for different power allocations, from 90% to 55% of the nominal power, is shown in Figure 9. At 90% power allocation, core-level gating is highly effective, and slightly outperforms Flicker overall. At higher power allocations, it may be sufficient to gate 1-2 cores, or perhaps none if the workload has a number of memory bound applications. The overheads associated with sampling and approximation, in terms of estimation error and the time spent in suboptimal configurations during sampling, makes Flicker less effective, and core-level gating preferable, at high power allocations.

As the power constraint increases, core-level gating blindly treats all applications the same in terms of hardware allocation, while Flicker's lane-level configuration more precisely matches the hardware to individual application characteristics. Moreover, as shown earlier, Flicker's accuracy increases as the power constraint becomes more stringent. At a 55% power cap, Flicker outperforms core-level gating by an average of 27%.

6.4.1 Adapting to Parallel Workloads

For a parallel application in which identical homogeneous threads are operating in parallel, global optimization is simplified, since core configurations for the threads should be identical. Thus, an optimization algorithm with knowledge of the application structure need only consider one thread of the application during these periods of homogeneous parallel execution. For example, a 32 core Flicker system running four applications, each with eight homogeneous threads, makes similar decisions as a four core system running four single-

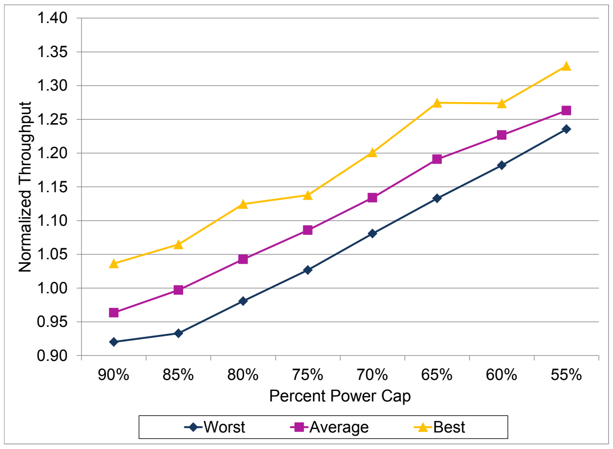


Figure 9: Normalized throughput of Flicker relative to core-level gating for different levels of allocated power. For each power setting, minimum, average, and maximum improvements are shown.

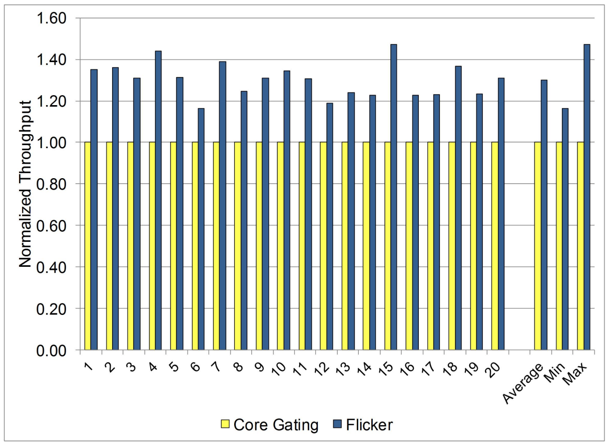


Figure 10: Performance of Flicker relative to core-level gating at 55% power allocation for each of the 20 workloads that approximate a multiprogrammed environment of parallel applications.

threaded applications. Once the configuration is determined for one of the application's threads, the cores for the remaining threads are identically configured. The decision space for a multiprogrammed system of parallel workloads is simplified, making the search more efficient compared to workloads of single-threaded applications.

To approximate a multiprogrammed workload consisting of parallel applications, four benchmarks are randomly selected and each replicated eight times. Twenty such workloads are created, for which the optimization algorithm samples and builds response surfaces for only the four unique benchmarks at one-eighth of the global power constraint, which simplifies the search. The performance of each of the 20 workloads at a 55% power budget relative to core-level gating is shown in Figure 10. The average performance improvement at this power cap is 30%, which is an improvement over the sequential application workloads.

In summary, Flicker's adaptive lane-based core microarchitecture and efficient "black box" global optimization algorithm significantly outperform core-level gating under strict power caps.

7. CONCLUSIONS

As microprocessors become increasingly heterogeneous and power constrained, the general-purpose cores must efficiently adapt to changes in power allocation in a way that maximizes overall performance. The Flicker general-purpose multicore architecture dynamically adapts to varying and potentially stringent limits on allocated power. The lane-based core microarchitecture permits tailoring an individual core to the running application without the high overheads of microarchitecture adaptation, but with greater flexibility than core-level power gating.

Exploiting the full potential of Flicker's flexible microarchitecture is challenging. The global controller must determine within a reasonably small fraction of a time slice the combination of powered lanes within the different cores that maximizes performance given the current power budget. To address this problem, a new online optimization approach is proposed that combines reduced sampling techniques, application of response surface models to online optimization, and heuristic online search. The algorithm efficiently finds a near-global-optimum configuration of lanes without requiring offline training, microarchitecture state, or foreknowledge of the workload. Flicker significantly outperforms core-level gating under the stringent power constraints expected in future systems.

8. REFERENCES

[1] D. H. Albonesi et al. Dynamically Tuning Processor Resources with Adaptive Processing. *IEEE Computer*, December 2003.

[2] R. I. Bahar and S. Manne. Power and Energy Reduction Via Pipeline Balancing. In *Proceedings of the International Symposium on Computer Architecture*, July 2001.

[3] R. Balasubramonian, D. H. Albonesi, A. Buyuktosunoglu, and S. Dwarkadas. Memory Hierarchy Reconfiguration for Energy and Performance in General-Purpose Processor Architectures. In *Proceedings of the International Symposium on Microarchitecture*, December 2000.

[4] R. Bergamaschi et al. Exploring Power Management in Multi-Core Systems. In *Proceedings of the Asia and South Pacific Design Automation Conference*, January 2008.

[5] R. Bitirgen, E. Ipek, and J. F. Martinez. Coordinated Management of Multiple Interacting Resources in Chip Multiprocessors: A Machine Learning Approach. In *Proceedings of the International Symposium on Microarchitecture*, November 2008.

[6] F. A. Bower, P. G. Shealy, S. Ozev, and D. J. Sorin. Tolerating Hard Faults in Microprocessor Array Structures. In *Proceedings of the International Conference on Dependable Systems and Networks*, June 2004.

[7] G. E. Box and D. W. Behnken. Some New Three Level Designs for the Study of Quantitative Variables. *Technometrics*, November 1960.

[8] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A Framework for Architectural-Level Power Analysis and Optimizations. In *Proceedings of the International Symposium on Computer Architecture*, June 2000.

[9] A. Buyuktosunoglu, T. Karkhanis, D. H. Albonesi, and P. Bose. Energy Efficient Co-Adaptive Instruction Fetch and Issue. In *Proceedings of the International Symposium on Computer Architecture*, June 2003.

[10] J. Chen, L. K. John, and D. Kaseridis. Modeling Program Resource Demand using Inherent Program Characteristics. *SIGMETRICS Perform. Eval. Rev.*, June 2011.

[11] D.-S. Chiou, D.-C. Juan, Y.-T. Chen, and S.-C. Chang. Fine-Grained Sleep Transistor Sizing Algorithm for Leakage Power Minimization. In *Proceedings of the Design Automation Conference*, June 2007.

[12] K. Deb. An Efficient Constraint Handling Method for Genetic Algorithms. In *Computer Methods in Applied Mechanics and Engineering*, June 2000.

[13] G. Derringer and D. Suich. Simultaneous Optimization of Several Response Variables. *Journal of Quality Technology*, October 1980.

[14] S. Dropsho et al. Integrating Adaptive On-Chip Storage Structures for Reduced Dynamic Power. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, September 2002.

[15] C. Dubach, T. M. Jones, E. V. Bonilla, and M. F. P. O'Boyle. A Predictive Model for Dynamic Microarchitectural Adaptivity Control. In *Proceedings of the International Symposium on Microarchitecture*, December 2010.

[16] Y. Eckert, S. Manne, M. J. Schulte, and D. A. Wood. Something Old and Something New: P-states Can Borrow Microarchitecture Techniques Too. In *Proceedings of the International Symposium on Low Power Electronics and Design*, September 2012.

[17] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger. Dark Silicon and the End of Multicore Scaling. In *Proceedings of the International Symposium on Computer Architecture*, June 2011.

[18] D. Folegnani and A. González. Energy-Effective Issue Logic. In *Proceedings of the International Symposium on Computer Architecture*, July 2001.

[19] D. Gibson and D. A. Wood. Forwardflow: A Scalable Core for Power-Constrained CMPs. In *Proceedings of the International Symposium on Computer Architecture*, June 2010.

[20] H.-M. Gutmann. A Radial Basis Function Method for Global Optimization. *Journal of Global Optimization*, March 2001.

[21] J. Harrington. The Desirability Function. In *Industrial Quality Control*, April 1965.

[22] M. C. Huang, J. Renau, and J. Torrellas. Positional Adaptation of Processors: Application to Energy Reduction. In *Proceedings of the International Symposium on Computer Architecture*, 2003.

[23] E. Ipek, O. Mutlu, J. F. Martínez, and R. Caruana. Self-Optimizing Memory Controllers: A Reinforcement Learning Approach. In *Proceedings of the International Symposium on Computer Architecture*, June 2008.

[24] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. 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*, December 2006.

[25] H. Jiang, M. Marek-Sadowska, and S. R. Nassif. Benefits and Costs of Power-Gating Technique. In *Proceedings of the International Conference on Computer Design*, October 2005.

[26] Khubaib, M. Suleman, M. Hashemi, C. Wilkerson, and Y. N. Patt. MorphCore: An Energy-Efficient Microarchitecture for High Performance ILP and High Throughput TLP. In *Proceedings of the International Symposium on Microarchitecture*, December 2012.

[27] S. Kim, S. Kosonocky, D. Knebel, and K. Stawiasz. Experimental Measurement of A Novel Power Gating Structure with Intermediate Power Saving Mode. In *Proceedings of the International Symposium on Low Power Electronics and Design*, August 2004.

[28] R. Kumar and G. Hinton. A Family of 45nm IA Processors. In *International Solid-State Circuits Conference*, February 2009.

[29] B. C. Lee and D. Brooks. Efficiency Trends and Limits from Comprehensive Microarchitectural Adaptivity. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, March 2008.

[30] B. C. Lee and D. M. Brooks. Accurate and Efficient Regression Modeling for Microarchitectural Performance and Power Prediction. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, October 2006.

[31] B. C. Lee, J. Collins, H. Wang, and D. Brooks. CPR: Composable Performance Regression for Scalable Multiprocessor Models. In *Proceedings of the International Symposium on Microarchitecture*, November 2008.

[32] J. Li and J. Martinez. Dynamic Power-Performance Adaptation of Parallel Computation on Chip Multiprocessors. *Proceedings of the International Symposium on High-Performance Computer Architecture*, February 2006.

[33] J. Mueller, C. Shoemaker, and R. Piche. SO-MI: A Surrogate Model Algorithm for Computationally Expensive Nonlinear Mixed-integer Black-box Global Optimization Problems. *Computers and Operations Research*, May 2013.

[34] P. Petrica, J. A. Winter, and D. H. Albonesi. Dynamic Power Redistribution in Failure Prone CMPs. In *Workshop on Energy Efficient Design*, June 2010.

[35] D. Ponomarev, G. Kucuk, and K. Ghose. Reducing Power Requirements of Instruction Scheduling through Dynamic Allocation of Multiple Datapath Resources. In *Proceedings of the International Symposium on Microarchitecture*, December 2001.

[36] R. G. Regis and C. A. Shoemaker. Local Function Approximation in Evolutionary Algorithms for the Optimization of Costly Functions. *IEEE Transactions*

on Evolutionary Computation, October 2004.

[37] R. G. Regis and C. A. Shoemaker. A Stochastic Radial Basis Function Method for the Global Optimization of Expensive Functions. *INFORMS Journal on Computing*, Fall 2007.

[38] R. G. Regis and C. A. Shoemaker. Combining Radial Basis Function Surrogates and Dynamic Coordinate Search in High-dimensional Expensive Black-box Optimization. *Engineering Optimization*, May 2013.

[39] J. Renau, B. Fraguela, J. Tuck, W. Liu, M. Prvulovic, L. Ceze, S. Sarangi, P. Sack, K. Strauss, and P. Montesinos. SESC Simulator. http://sesc.sourceforge.net. 2005.

[40] J. Sharkey, A. Buyuktosunoglu, and P. Bose. Evaluating Design Tradeoffs in On-Chip Power Management for CMPs. In *Proceedings of the International Symposium on Low Power Electronics and Design*, August 2007.

[41] K. Shi and D. Howard. Sleep Transistor Design and Implementation - Simple Concepts Yet Challenges To Be Optimum. In *International Symposium on VLSI Design*, April 2006.

[42] D. Tarjan, S. Thoziyoor, and N. Jouppi. Cacti 5.3. *HP Laboratories Palo Alto Technical Report*, 2005.

[43] R. Teodorescu and J. Torrellas. Variation-Aware Application Scheduling and Power Management for Chip Multiprocessors. In *Proceedings of the International Symposium on Computer Architecture*, June 2008.

[44] G. Venkatesh et al. Conservation Cores: Reducing the Energy of Mature Computations. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, March 2010.

[45] Y. Watanabe, J. D. Davis, and D. A. Wood. WiDGET: Wisconsin Decoupled Grid Execution Tiles. In *Proceedings of the International Symposium on Computer Architecture*, June 2010.

[46] T. F. Wenisch, R. E. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, and J. C. Hoe. SimFlex: Statistical Sampling of Computer System Simulation. *IEEE Micro*, July/August 2006.

[47] C. F. J. Wu and M. S. Hamada. *Experiments: Planning, Analysis, and Optimization.* John Wiley and Sons, Inc., 2009.

[48] J. Yi, D. Lilja, and D. Hawkins. A Statistically Rigorous Approach for Improving Simulation Methodology. In *Proceedings of the International Symposium of High-Performance Computer Architecture*, February 2003.

[49] Y. Zhang, D. Parikh, K. Sankaranarayanan, K. Skadron, and M. Stan. HotLeakage: A Temperature-Aware Model of Subthreshold and Gate Leakage for Architects. In *University of Virginia Technical Report CS-2003-05*, 2003.

APPENDIX

A Radial Basis Function has form $\varphi(||x-c||)$ whose value depends only on the Euclidian distance from the center c. Assuming that there are n samples, there are $\underline{x}_1, \underline{x}_2, ..., \underline{x}_n \in \mathbb{R}^d$ centers, each with its corresponding Radial Basis Function, where $\underline{x}_n$ are the sampled points in a d-dimensional real space, and d is the dimension of the independent variables (*i.e.*, d=3 because there are three factors in the Flicker pipeline). Each point $\underline{x}_n$ is the n^{th} sampled core configuration of the three factor levels $(x_{1_n}, x_{2_n}, x_{3_n})$. The interpolating RBF response surface is of the form:

$$\hat{\mathbf{y}} = \sum_{i=1}^{n} \lambda_i \varphi(||\underline{x} - \underline{x}_i||) + p(\underline{x}) \tag{3}$$

where λ_i are the coefficients of the response function, $||\cdot||$ is the Euclidean distance between two d-dimensional points, and $p(\underline{x}) = \mathbf{b}^T \underline{x} + a$ is a polynomial tail. Without the polynomial tail, the n by n matrix $\mathbf{\Phi}$ with elements $\Phi_{ij} = \varphi(||\underline{x}_i - \underline{x}_j||)$ described below in Equation 4 might become singular [20]. A surface is built for each of the two responses (throughput and power).

Flicker uses a cubic RBF $\varphi(\underline{x}_{ij}) = \left(||\underline{x}_i - \underline{x}_j||\right)^3$ that needs a linear polynomial tail: $p(\underline{x}) = b_0 + b_1x_1 + b_2x_2 + b_3x_3$. Characterizing an application requires obtaining the set of coefficients $\boldsymbol{\lambda} = (\lambda_1, ..., \lambda_n)$ and $\mathbf{b} = (b_0, b_1, b_2, b_3)$, which is accomplished by solving the linear system of equations in 3.

$\hat{y} = (y_1(\underline{x}_1), ..., y_n(\underline{x}_n))$ is the value of the system responses sampled at the points $(x_1, ..., x_n)$. $y_n(\underline{x}_n)$ is either the measured throughput or the measured power of the core configured with pipeline regions $\underline{x}_n = [x_{1_n}\ x_{2_n}\ x_{3_n}]$. A surrogate RBF surface is built for the throughput response and a second RBF for the power response. The methodology to obtain both of them is identical, the only difference being the $y_n(\underline{x}_n)$ values. If

$$\mathbf{\Phi} = \begin{bmatrix} \Phi_{11} & \Phi_{12} & \dots & \Phi_{1n} \\ \Phi_{21} & \Phi_{22} & \dots & \Phi_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ \Phi_{n1} & \Phi_{n2} & \dots & \Phi_{nn} \end{bmatrix} \tag{4}$$

where $\Phi_{ij} = \varphi(||\underline{x}_i - \underline{x}_j||)$ and

$$\mathbf{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \mathbf{P} = \begin{bmatrix} \underline{x}_1 & 1 \\ \underline{x}_2 & 1 \\ \vdots & \vdots \\ \underline{x}_n & 1 \end{bmatrix} \boldsymbol{\lambda} = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_n \end{bmatrix} \mathbf{c} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_0 \end{bmatrix} \tag{5}$$

then the system of equations can be rewritten in contracted form and solved for the coefficients λ and c:

$$\begin{bmatrix} \mathbf{\Phi} & \mathbf{P} \\ \mathbf{P}^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\lambda} \\ \mathbf{c} \end{bmatrix} = \begin{bmatrix} \mathbf{Y} \\ \mathbf{0} \end{bmatrix} \tag{6}$$

$$\begin{bmatrix} \boldsymbol{\lambda} \\ \mathbf{c} \end{bmatrix} = \underbrace{\begin{bmatrix} \mathbf{\Phi} & \mathbf{P} \\ \mathbf{P}^T & \mathbf{0} \end{bmatrix}^T}_{\text{offline computation}} \begin{bmatrix} \mathbf{Y} \\ \mathbf{0} \end{bmatrix} \tag{7}$$

By adopting RBFs in this manner, the majority of the computation to obtain the coefficients can be performed offline (as noted in Equation 7), resulting in fast surface fitting (in < 1 ms for Flicker), which makes this a viable approach for online optimization. Equation 7 applies if the location of the sample points $x_1, ..., x_n$ stays the same so that the matrix involving $\mathbf{P}$ does not change, even though the vector $\mathbf{Y}$ will change in each sampling interval due to dynamic changes in the behavior of the running applications.