PoweRock: Power Modeling and Flexible Dynamic Power Management for Many-core Architectures

Zhiquan Lai, King Tin Lam, *Member, IEEE,* Cho-Li Wang, *Member, IEEE,* and Jinshu Su *Member, IEEE*

***Abstract*—Energy-efficient design for green high performance computing (HPC) is a formidable mission for today's computer scientists. Dynamic power management (DPM) is a key enabling technology involved. While DPM can have various goals for different application scenarios (e.g. enforcing an upper bound on power consumption or optimizing energy usage under a performance-loss constraint), existing DPM solutions are generally designed to meet only one goal and not adaptable to changes in optimization objectives. This paper proposes a novel flexible DPM approach based on a profile-guided dynamic voltage/frequency scaling (DVFS) scheme to meet the different goals. Our contributions include: (1) a new profiling method for (distributed) shared-memory parallel applications to flexibly determine the optimal frequency and voltage for different phases of the execution; (2) a power model based on hardware performance counters; and (3) a hierarchical domain-aware power control design boosting the DPM system scalability for many-core chips. We implement the approach into a working library dubbed PoweRock and evaluate it on the Intel SCC port of the Barrelfish operating system. Experimental results obtained from several well-known benchmarks show that PoweRock attains significant energy and energy-delay product (EDP) improvements (average: 37.1% and 25.5%; best-case: 64.0% and 65.4%, respectively) over a static power scheme.**

***Index Terms*—Dynamic power management, dynamic voltage and frequency scaling, DVFS, power modeling, many-core processors**

I. Introduction

In this computing era, it is not an overstatement to say transistors are "free" but power is expensive [1]. The top-ranked supercomputer, China's Tianhe-2, consumes 17.81 MW (excluding the cooling system) or 156 GWh for a year, which is equivalent to the average annual domestic electricity consumption of a midsize town (over 312,800 persons) in China. The other top ten supercomputers on the the latest TOP500 List [2] are having similar power efficiencies ranging around 1,900 to 2,700 Mflops/watt, implying the commonplace "huge power" concerns in the current HPC landscape. It is a pressing research challenge to boost energy efficiency of HPC or cloud computing data centers for curbing the ever-increasing running expenditure and environmental impact.

To ease the tension between high performance and low energy, the concepts of supercomputing and green computing are closely intertwined nowadays. *Dynamic power management (DPM)*, encompassing a set of control strategies aimed at adapting the power/performance of a system to its workload, is an important vehicle for green supercomputing. DPM refers to selective shutdown or slowdown of idle or underutilized system components. Clock gating and power gating are canonical techniques for idle power saving. *Dynamic voltage and frequency scaling (DVFS)*, including the so-called *multiple energy gears* [3] and *dynamic overclocking* [4], extends the dynamic power efficiency by scavenging opportunistic energy benefits during active program runs (e.g. tuning down the clock of the CPU during long memory access or I/O events).

A DPM policy can be defined with different goals. Some DPM schemes aim to maximize the performance or throughput within a fixed power budget. This is known as *power capping* [5]–[8], which is useful for financial or thermal management. There are others aimed at minimizing the power consumption under a maximal performance loss constraint [9], among which a few performance-savvy schemes are doing this under tight constraints of vanishingly small performance loss [10]. For quantitative analysis of tradeoffs between speed and energy, various energy efficiency metrics like the most widely used *energy-delay product (EDP)* (a.k.a. *energy-performance ratio*) [11] have been proposed for evaluating DPM schemes.

To the best of our knowledge, there is however not yet a DPM solution capable of flexibly meeting different goals. For example, the solution proposed by Ioannou *et al.* [9] for saving as much energy as possible below a threshold of maximum performance loss, without a top-to-bottom overhaul, cannot be used to cap the power instead. A universal DPM design and implementation with selectable policies for varying application needs is an important research gap to fill. For instance, an enterprise can deploy the same application with different DPM policies to data centers with different electricity costs—high-performance policies for low-cost areas and power capping policies for high-cost ones—to cost-effectively meet service demands. With such flexible per-application DPM infrastructure, changes in optimization goal can be easily entertained by redeploying the application with a new DPM option.

To make this approach viable, rigorous power-performance models are required to support DPM policy adaptation. There is rich literature on multiprocessor power modeling [11]–[17]. These works however either did not establish analytical power models or missed out some key elements determining the chip power in their models. Accurate power modeling for emerging many-core CPUs is also lacking. When using DVFS to scale the power, the biggest challenge is not about deciding if the power state (frequency/voltage pair, a.k.a. *energy gear*) should be scaled up or down, but about how much the scaling should be made (in other words, which state is regarded the optimal).

Z. Lai and J. Su are with the College of Computer, National University of Defense Technology, Changsha, China (e-mail: zqlai@nudt.edu.cn; sjs@nudt.edu.cn).

K.T. Lam and C.-L. Wang are with the Department of Computer Science, the University of Hong Kong, Hong Kong, China (e-mail: ktlam@cs.hku.hk; clwang@cs.hku.hk).

Prior work employs "trial-and-error" online profiling [9] or offline training with lots of experiments [3], [11] to estimate optimal power states for a certain goal of power-performance tradeoff. However, once the goal is changed, these profiling approaches and DPM implementations designed for the original purposes will cease working.

These concerns lead us to develop the *Power Optimization Controller Kit (PoweRock)* [pronounced "power rock"]—a new DPM solution to solve the power/performance modeling, profiling and DVFS scheduling problems tailored to tiled many-core processor architectures such as the Intel SCC [18]. We propose a new power model based on the memory access and synchronization patterns of the program. Our power model advances the state of the art by taking the program execution pattern and voltage/frequency (V/f, henceforth) settings into account. With this model, we can accurately estimate the total chip power consumption and execution time in different V/f settings for programs with different patterns. Hence, we are able to quickly and precisely determine the optimal power level(s) for executing the program (or phases of the program). Based on the power model, this work also designed and implemented a hierarchical domain-aware DPM framework for tiled many-core chips with multiple clock/voltage domains.

In summary, the contributions of this work include:

- a performance/power prediction model considering both V/f states and program execution patterns to accomplish accurate and flexible power management;
- a novel profiler design to facilitate the model for phase-based power management of parallel applications following a shared-memory programming style;
- a flexible DPM solution tailored to tiled many-core architectures (we can optimize the V/f setting for a given DPM goal using the power and performance model, and update the optimal setting easily for a changed goal);
- a working implementation of a profile-guided DVFS controller and power management library (for the Barrelfish multikernel operating system [19] on the Intel SCC), and thorough experimental evaluation using several well-known benchmarks to show significant energy saving or EDP improvement (up to 65%) of our solution.

The rest of the paper is organized as follows. We explain the power and performance model in Section II. Section III presents our profile-guided DVFS scheme for flexible power management. We describe the design and system implementation of our PoweRock library in Section IV. Section V and Section VI detail the evaluation methodology and experimental results of our implemented models respectively. Finally, Section VIII concludes the paper.

II. Power Modeling

A. Power Model

Many-core architectures to date usually have much more complex hardware design than uniprocessor or multicore chips. Apart from the plenty of cores, non-trivial network-on-chip (NoC), multiple memory controllers, multilevel on-chip caches and programmable buffers, fine-grained DVFS regulator units, etc. are present. The total chip power equals the consumptions by every component. For simplicity, our model focuses on the main power contributor—the CPU cores—and the NoC. First of all, the power of a single core i (denoted as P_i) is formulated as (2) where k is a constant coefficient.

$$P_i = P_{i,dyn}(v_i, f_i) + P_{i,sta} \tag{1}$$

$$P_{i,dyn}(v_i, f_i) = k \cdot A_i \cdot f_i \cdot {v_i}^2 \tag{2}$$

P_i consists of two parts—static power ($P_{i,sta}$) and dynamic power ($P_{i,dyn}$). Static power depends on the voltage and the CPU's thermal design power (TDP) whereas the dynamic core power is proportional to frequency (f_i), squared supply voltage (${v_i}^2$) and the (program-level) activity factor (A_i). The activity factor generally has a functional relationship with the program execution patterns (to be explained). To simplify the power model, we assume the static core power $P_{i,sta}$ and the activity factor A_i for a certain program are both constant. So we can estimate the power of a core when a trunk of program is executed with any V/f settings.

For current many-core chip designs like Intel's SCC, dynamic power state tuning of the NoC is generally unsupported. So it is a fair assumption that the network power, denoted by P_{noc}, is a constant. Let N be the number of cores. The formula for the total chip power (denoted by P) can be expressed as

$$\begin{aligned} P &= \sum_{i=0}^{N-1} (P_i) + P_{noc} \\ &= \sum_{i=0}^{N-1} (P_{i,dyn}(v,f)) + \sum_{i=0}^{N-1} (P_{i,sta}) + P_{noc} \\ &= \sum_{i=0}^{N-1} (k \cdot A_i \cdot f_i \cdot v_i^2) + (\sum_{i=0}^{N-1} (P_{i,sta}) + P_{noc}). \end{aligned} \tag{3}$$

Prior works [14], [20], [21] revealed that power dissipation is strongly correlated to #instructions per cycle (IPC). Thus we assume the activity factor A_i of core i is a function of IPC at a specific power setting (V/f pair). As k is a constant, $k \cdot A_i$ is also a function of IPC (denoted as $g(IPC_i)$). Let $P_s = \sum_{i=0}^{N-1}(P_{i,sta}) + P_{noc}$, then (3) is rewritten as

$$P = \sum_{i=1}^{N-1} (g(IPC_i) \cdot f_i \cdot v_i^2) + P_s. \tag{4}$$

The next and important part of the power modeling is to parameterize $g()$ and P_s.

B. Parameterization of the Power Model

Here we describe how to parameterize the model (i.e. specify $g()$ and P_s) with a case study—the Intel SCC platform [22]. Intel SCC is a 48-core experimental many-core chip with fine-grained DVFS mechanism [18]. The IPC pattern of each core can be derived by reading the *performance monitor counters (PMCs)* provided by the CPU chip. The instantaneous chip power can be measured by reading the power sensors provided by the SCC board. Thus the energy consumption is obtainable by integrating the instantaneous power over time.

To derive $g()$ and P_s, we conduct microbenchmarking using two primitive compute kernels, INT and FP, which perform arithmetic computations on an array of integer and floating-point variables respectively. We run the programs with various

TABLE I
MICROBENCHMARK SETTING FOR POWER MODELING

Frequency/Voltage *(MHz/V)*				
800/1.1	533/0.9	400/0.8	320/0.8	267/0.8
229/0.8	200/0.8	178/0.8	160/0.8	145/0.8
133/0.8	123/0.8	114/0.8	107/0.8	100/0.8

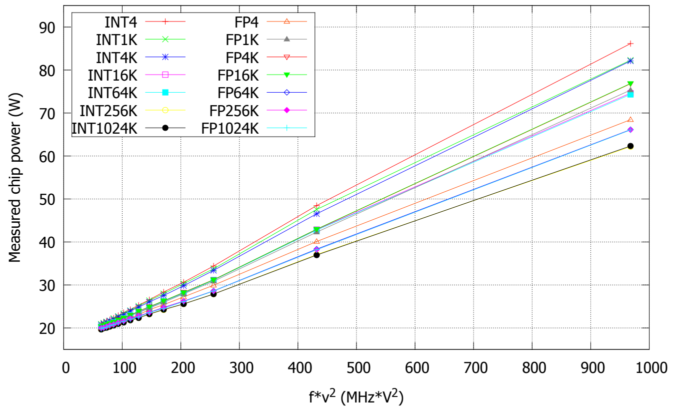


Fig. 1. Microbenchmarking results for parameterizing the power model

problem sizes (array length) to produce various IPC patterns. They are launched with 15 V/f settings (as shown in Table I) and seven problem sizes (4, 1K, 4K, 8K, 64K, 256K and 1024K), giving 2×15×7=210 data points plotted as Fig. 1.

As we cannot measure the individual power of each core directly, we exploit a trick to simplify the chip power function for specifying g and P_s by launching the same program on all the 48 cores of the SCC and setting the V/f levels of all cores to be the same. Hence, the activity factors of all per-core program instances are the same, denoted by $g(IPC)$. So the chip power P can be formulated as

$$P = N \cdot g(IPC) \cdot f \cdot v^2 + P_s. \tag{5}$$

Fig. 1 shows the microbenchmarking results. The x-axis and y-axis represent $f \cdot v^2$ (unit: MHzV^2) and the chip power P respectively. We can observe that the power consumption of the chip is linearly related to $f \cdot v^2$. For each program, we use a linear function $Y = \alpha X + \beta$ for curve-fitting their data points, where X denotes $f \cdot v^2$, Y denotes P, α and β denote the fitting values of $N \cdot g(IPC)$ and P_s respectively. The regression analysis results are showed in Table II.

In Table II, the power for each program is linearly related to $f \cdot v^2$ with almost the same P_s (i.e. β). We can hence take the average of coefficients β as the static chip power:

$$P_s \approx 16.185. \tag{6}$$

On the other hand, to get the specified analytic function of $g(IPC)$, we use a function to fit the values of α with respect to IPC. The values of α with respect to IPC are plotted in Fig. 2. Finally, using the least-squares regression algorithm, we get the parameterized function of $N \cdot g(IPC)$ with R^2 coefficient = 0.812, expressed as (7).

$$N \cdot g(IPC) = 0.068 \cdot IPC^{0.09} \tag{7}$$

TABLE II
RESULT OF LINEAR REGRESSION ANALYSIS FOR THE POWER MODEL

Test case[a]	**IPC@800MHz**	$\boldsymbol{\alpha}$	$\boldsymbol{\beta}$
INT4	0.9130	0.072	16.14
INT1K	0.7335	0.068	16.39
INT4K	0.7281	0.068	16.17
INT16K	0.4310	0.060	16.07
INT64K	0.4222	0.060	16.09
INT256K	0.0642	0.047	16.45
INT1024K	0.0640	0.047	16.41
FP4	0.1239	0.053	16.37
FP1K	0.5883	0.061	15.86
FP4K	0.2382	0.062	15.91
FP16K	0.2380	0.062	15.91
FP64K	0.0309	0.051	16.30
FP256K	0.0309	0.051	16.33
FP1024K	0.0310	0.051	16.19
	Average $\boldsymbol{\beta}$		16.185

[a]Test case name is made up of program name and problem size.

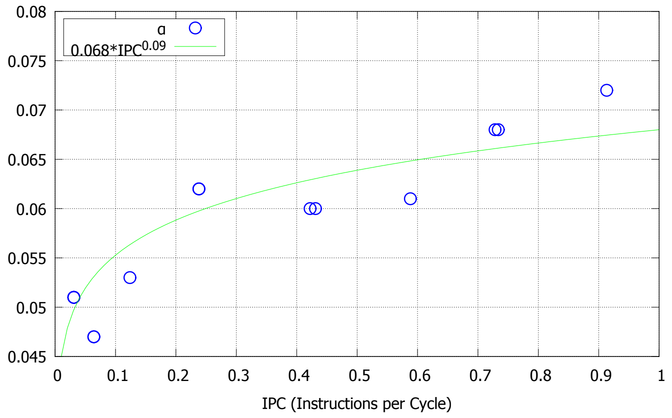


Fig. 2. Fitting analysis for $N \cdot g(IPC)$

Since N equals 48 for the SCC case, we can derive $g(IPC_i)$ for each core i, as in (8).

$$k \cdot A_i = g(IPC_i) = 1.417e^{-3} \cdot {IPC_i}^{0.09} \tag{8}$$

In (8), IPC equals #instructions per clock at a profiling frequency of 800MHz. We substitute the parameters of $g(IPC_i)$ and P_s into (4), giving the chip power model for Intel's SCC as follows:

$$P = \sum_{i-0}^{N} 1.417e^{-3} \cdot {IPC_i}^{0.09} \cdot f_i \cdot {v_i}^2 + 16.185. \tag{9}$$

The units of the variables are: watts (W) for P, MHz for f_i, and volts (V) for v_i. With this chip power model, we can predict the SCC chip power in any power states for various program execution patterns, as shown in Fig. 3. As we never see an IPC value larger than two instructions/cycle, we present the figure with IPC ranging from 0 to 2.

III. MODEL-DRIVEN FLEXIBLE DPM APPROACH

A. Phase-based Profiling and Profile-guided DVFS

In this work, we target applications following a lock-based shared-memory programming model. Non-coherent shared-memory machines like Intel's SCC or a cluster can rely on some shared virtual memory (SVM) solutions to virtualize

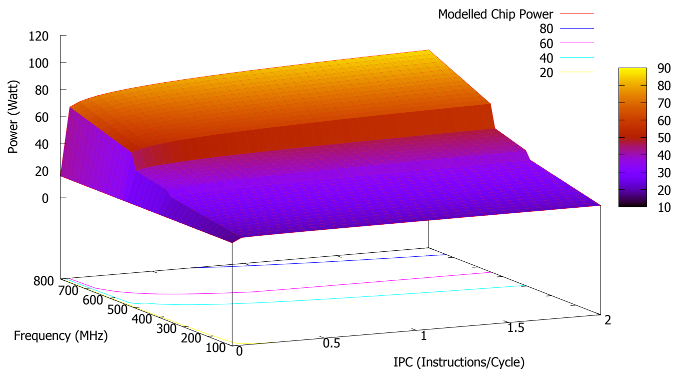


Fig. 3. The estimated chip power of Intel SCC with respect to frequency settings (i.e. power states) and IPC (i.e. program execution patterns)

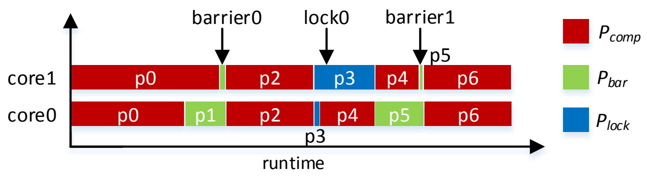


Fig. 4. Phases partitioned by barrier and lock operations (P_{comp}: computation phase, P_{bar}: barrier phase, P_{lock}: lock phase)

a coherent shared memory space to stay conformable to this assumption. For ease of study and execution profiling, we exploit a parallel program's pattern of synchronization among multiple cores to split its runtime up into logical segments known as *phases*. In a (distributed) shared-memory programming environment, the parallel program execution is generally partitioned by lock and barrier operations, which are the likely turning points across which the program may vary in computational or memory-access patterns. Therefore, we define a *phase* as an execution window delimited by locks or barriers, including the busy-waiting time interval in the locks/barriers.

Fig. 4 illustrates our phase definition with two processes running on two cores. They synchronize via two barriers and one lock in the program. The runtime of each core is divided into multiple phases as shown. In general, we have three basic types of phases: P_{comp}, P_{bar} and P_{lock} (the red, green and blue portions), which denote the local computation time, barrier time and lock time respectively. A P_{comp} phase—usually the useful computation portion of the program—happens between two busy-waiting phases (P_{bar} or P_{lock}). According to this definition, in the case of Fig. 4, we could partition core0's execution time into seven phases, including four P_{comp}'s (p_0, p_2, p_4, p_6), two P_{bar}'s (p_1, p_5) and one P_{lock} (p_3).

There are two benefits with this phase partitioning scheme. First, it respects the shared-memory programming model naturally. Second, this allows profiler calls and DPM functions to be hooked onto or implemented into the internal of synchronization routines of the parallel programming library, making the concerns of power management transparent to the application level. In this work, we provide our own parallel programming library—Rhymes SVM [23] for developing and running parallel programs on the SCC platform. The implementation of lock and barrier functions contains calls to the PoweRock profiler and DPM API.

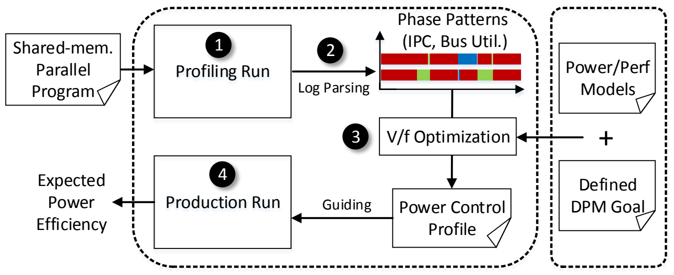


Fig. 5. Processing flow of profile-guided DVFS for flexible DPM

Fig. 5 shows the processing flow of our profile-guided power management scheme. For a given program, an offline profiling run (step 1) is used to derive its phased execution pattern. During the profiling run, our custom-made profiler, triggered by calls from barrier and lock routines of the programming library, records those relevant performance monitor counters (PMCs) of the CPU, e.g. IPC and bus utilization, along with other runtime information. The profiling data, including the PMC values, phase type and phase time, are dumped and parsed (step 2) to generate an execution profile for the program—a sequence of phases with per-phase execution pattern information like IPC. Meanwhile, we measure the instantaneous chip power during the execution. The next and key step (step 3) is to compile the *power control profile (PCP)* which contains the optimal voltage and frequency for each phase towards the given DPM goal quantified by some corresponding indexes like the power capping upper bound, the maximum performance loss or the least EDP target. In this step, we need the power model (in Section II) and the performance model (to be explained) to predict the power and runtime in different V/f settings. Finally, the derived PCP can be applied to production runs (step 4), which are guided for the targeted power efficiency.

B. Performance Model

The total execution time of a program is generally made up of computation, memory access, I/O and synchronization. The computation part is localized to the processor cache while the others involve the use of the front-side bus. In other words, the sum of CPU computation time (t_{cpu}) and bus-related processing time (t_{bus}) approximates to the total execution time on a single core.

Suppose a phase p_i gives runtime t_0 (seconds) and bus utilization BU (percentage of the bus cycles over the total execution cycles read from the PMCs) when running at frequency f_0 (MHz). We can predict the runtime t of phase p_i at a different frequency f as follows:

$$t(f, BU) = t_{cpu} + t_{bus} = (1 - BU) \cdot t_0 \cdot \frac{f_0}{f} + BU \cdot t_0. \quad (10)$$

With this performance model, we can estimate the execution time of each phase in different power settings, and hence the performance degradation (the ratio of t/t_0) due to V/f scaling in the 2D space of frequency versus bus utilization, as shown

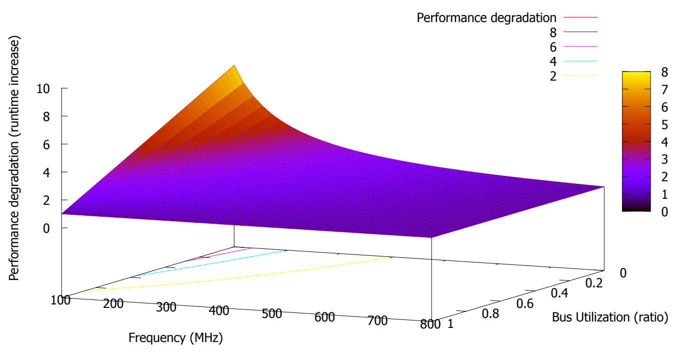


Fig. 6. Estimation of the performance degradation (runtime increase) vs. frequency and bus utilization (compared with maximum frequency of 800MHz)

in Fig. 6. The x- and y-axis represent the frequency in the range of 100 to 800MHz (the spectrum of Intel SCC) and the bus utilization in the range from 0 to 1, respectively. The z-axis indicates the performance degradation in the condition of f_0 set to 800MHz. Agreeing with our intuition, we can observe that the smaller the bus utilization ratio, the larger the performance impact of tuning down the frequency.

C. Power Efficiency Optimization Using the Models

By leveraging the power and performance models, we can estimate the consumed power consumption and execution time for each phase in different V/f settings. Then we can choose an optimal power state[1] for each phase towards the DPM goal. To summarize, the process can be outlined as follows:

- For each program or each phase (denoted as p_i) executed in a static power state (f_0), besides the execution time, we get two metrics related to power and performance, instructions per cycle (IPC) and bus utilization (BU), derivable from the PMCs of the underlying CPU core.
- We estimate the power consumption and execution time in any power state f using the power-performance model.
- Towards an optimization goal (e.g. minimal EDP or minimal energy), we search for an optimal power state. As the power states available in the hardware are usually limited, the complexity of the search could be ignored.

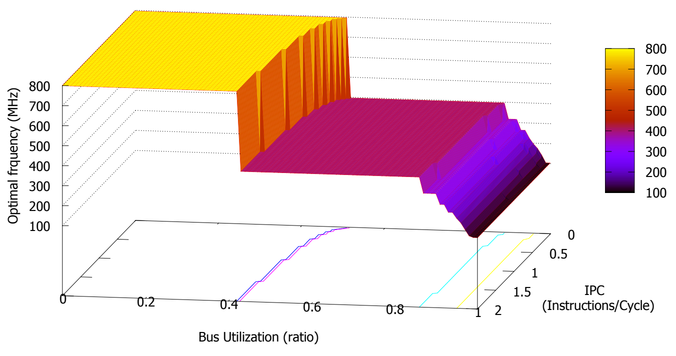
For instance, assuming the DPM goal is to minimize the EDP, we can predict the EDP of each phase in a certain power setting using the adapted power-performance model as in (11).

$$\begin{aligned} EDP(f) &= Energy(f) \cdot Runtime(f) \\ &= (t(f, BU) \cdot P(f, IPC)) \cdot t(f, BU) \\ &= P(f, IPC) \cdot t(f, BU)^2 \end{aligned} \tag{11}$$

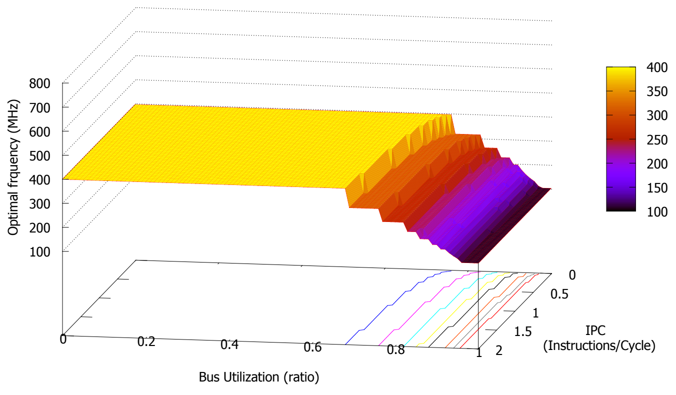
Then we can determine the optimal power setting for each phase to meet the minimal EDP target.

The most powerful aspect of our model is its ability to determine the optimal frequency setting (hence the power state) for any program or phase, given the IPC and BU pattern (uncovered by our profiler). As shown in Fig. 7(a), we plot the variation of the optimal frequency (z-axis) towards the

[1]Assuming we always use the least supplied voltage for a certain frequency, it suffices to use the frequency alone to represent the power state in the paper.



(a) Optimal frequency settings for minimal EDP



(b) Optimal frequency settings for minimal energy

Fig. 7. Optimal frequency settings for different program patterns in 2D space

minimal-EDP optimization goal, with respect to the two dimensions (x- and y-axis) representing the program patterns—instructions per cycle (ranging from 0 to 2, the maximum observed in our experiments) and bus utilization (ranging from 0 to 1). Likewise, Fig. 7(b) presents the optimal frequency towards the minimal-energy optimization goal.

This optimization method does not consider the V/f scaling latency. If the power level before the phase is different from the predicted optimal one for the phase, we have to scale the power level first, probably introducing non-negligible voltage/frequency switching time and extra power dissipation. So methods without considering such latency could make inaccurate DVFS schedules. Our DPM scheme has been made latency-aware when determining the optimal power levels (via the methodology detailed in our earlier publication [24]).

IV. PoweRock: Design and Implementation

In view of the potential of the multikernel approach to scalable OS design, PoweRock was designed and implemented on the Barrelfish OS [19]. As another support tool for Barrelfish, we developed the Rhymes Shared Virtual Memory (SVM) system [23], which leverages software virtualization to restore cache coherence on the SCC machine with non-coherent memory architecture. The PoweRock APIs are intended for use by the SVM layer instead of the application layer for not compromising the programmability for power saving.

Fig. 8(a) shows the system architecture of PoweRock. There is a runtime instance of PoweRock spawned on each core for handling the calls from the Rhymes library and determining the optimal power state for the local CPU core. On many-core systems, per-core DVFS is a costly design, so cores are generally grouped into frequency and/or voltage domains where V/f scaling can only happen in domain-level granularity. Each PoweRock instance has a *rank* and domain-level *role*, both defined upon OS bootstrap. The rank is the instance identifier (like core id); the role differentiates if the instance is the domain master. By synchronizing the locally optimal V/f settings with other instances in the same domain, a master instance determines an optimal V/f level for the domain, and performs the V/f scaling action. PoweRock is composed of four major functional components—OPS, PPM, DVFSC and RPRS—which are detailed as follows.

A. Offline Profiling Subsystem (OPS)

The OPS refers to the subsystem encompassing the profiler and power monitor designed for execution profiling and power measurement before applications get deployed to production. Similar to the work of Magklis *et al.* [25], we need a profiling run to generate the power control profile (PCP) for each application. In the profiling run, the profiler makes use of kernel-level PoweRock system calls to retrieve IPC and bus utilization (BU) statistics from the hardware performance monitor counters while the power monitor obtains the chip power readings from the BMC unit. The outputs are combined to generate the PCP for the application. In our current implementation, the PCP will be transformed to a C header file to be included by the application in production runs.

We trigger the profiler by modifying the barrier and lock routines in the Rhymes SVM library. The beginnings and the ends of barrier and lock routines are augmented with code to record the timestamp and PMC readings. The timestamp is read from a wall-clock of 125MHz on the SCC board [22]. We implement a kernel-level module to monitor the PMCs of the CPU cores. For P54C architecture [26], three counters—local time stamp counter (TSC), instruction executed and clocks while a bus cycle is in progress—are useful for deriving IPC and BU of each phase. Although the patterns within a phase may keep varying in some ways, we use the average to represent the overall pattern of the phase.

As shared-memory parallel programs usually follow a SPMD model (single program, multiple data), the execution patterns of all processes are alike. However, if the application follows a master/slave design pattern, it is common that the execution pattern of the master process (presumably running on core0) is somehow different from that of the slave processes running on the rest of cores. Thus, the PCP includes the profiles of the "master core" and one of the other "slave cores".

B. Profile-guided Power Manager (PPM)

When the execution enters or exits a barrier/lock, it means a new phase begins (refer to Section III-A). At this moment, the Rhymes library calls the PoweRock API. Then the PPM handles this event and determines the DVFS action for the coming phase by referencing the power control profile attached to the application. Due to program dynamics, the actual phase pattern of the production run may not be equivalent to that of the profiling run. So the PPM should be able to tolerate phase pattern discrepancies. This is the key problem the PPM has to resolve. For an SVM-based application, the sequence of barrier phases (P_{bar}) is generally fixed. In our current design, we take the barrier phases as the reference points to tolerate the phase pattern difference between the profile and the runtime execution. The next step the PPM does is to request a power state transition to the optimal V/f setting in the profile. For a phase which does not fit the profile, we request a conservative frequency of 400MHz without voltage change. The request is then sent to the DVFS controller.

TABLE III
POWEROCK API FOR DVFS CONTROL

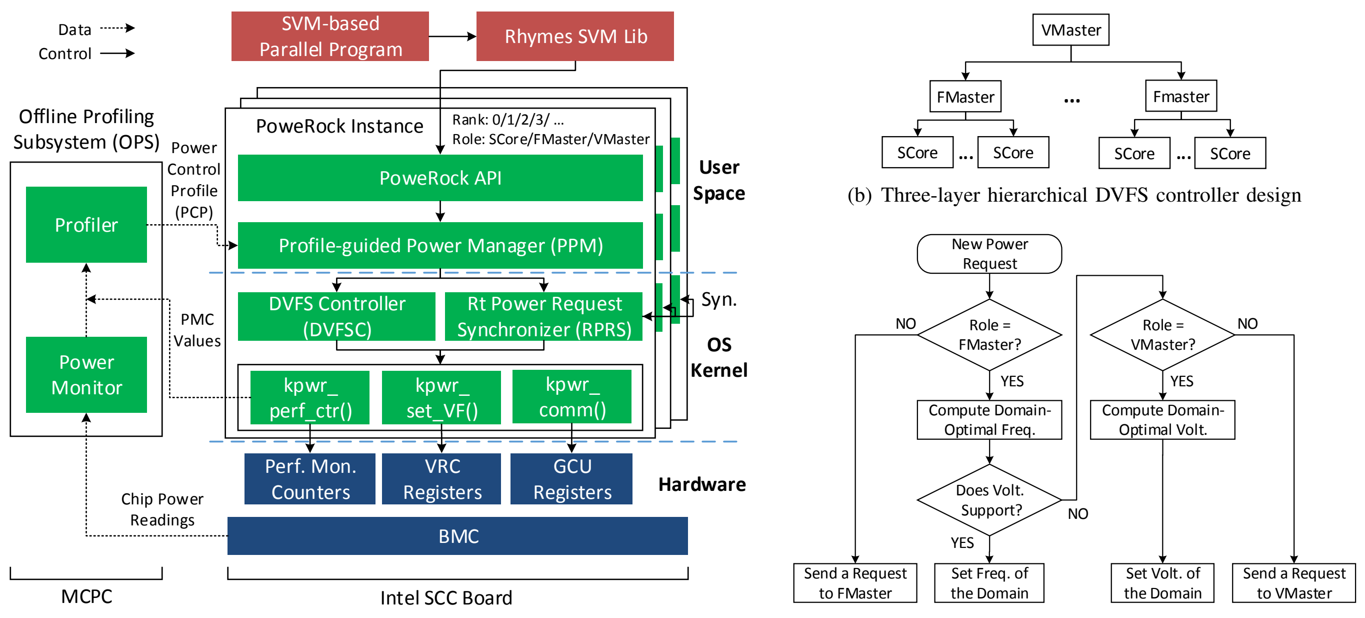
API Functions and Descriptions
Parameter specification: • `Fdiv` (input) - the requested value for the frequency divider • `Vlevel` (input) - the requested value for the voltage level • `new_Fdiv` (output) - the returned value of the new frequency divider • `new_Vlevel` (output) - the returned value of the new voltage level
`pwr_local_power_request(Fdiv,new_Fdiv,new_Vlevel)` A non-blocking function for the caller core to make a power request to the low-level power management system. The voltage setting is assumed to be the least voltage value. But the exact V/f of a domain will be decided by the domain master by negotiation among all the cores in the domain.
`pwr_local_frequency_request(Fdiv,new_Fdiv)` A non-blocking function that explicitly scales the frequency of the cores in the local frequency domain. The function takes effect only if called by frequency domain masters.
`pwr_local_voltage_request(Vlevel,new_Vlevel)` A conditional blocking function that explicitly sets the voltage level of the local voltage domain. The function takes effect only if called by voltage domain masters. In scale-down cases, the function is non-blocking; in scale-up cases, it blocks until the voltage reaches the expected level.

C. DVFS Controller (DVFSC)

The DVFSC provides an interface for other components to adaptively configure the power settings of CPU cores. It adopts a domain-aware design tailored to the common DVFS design of most multicore or many-core chips with multiple clock/voltage domains. As shown in Fig. 10, each two-core tile forms a frequency domain, and the 48 cores of the SCC chip are partitioned into six four-tile voltage domains. As shown in Fig. 8(b), we design a three-layer hierarchical DVFS controller for this type of many-core chips. Each core plays a role of stub core (SCore). One core in each voltage (or frequency) domain is chosen as the domain master called VMaster (or FMaster). The DVFSC is accessible from the user space through the PoweRock API functions listed in Table III.

D. Real-time Power Request Synchronizer (RPRS)

As V/f switching is done in the granularity of a domain, the RPRS is designed to synchronize different power requests in the same domain. Fig. 8(c) describes the RPRS workflow for a stub core when it receives a power request. If the stub core is not a FMaster, RPRS is responsible for issuing the power request to its FMaster in the domain. If it is the case, the FMaster computes the domain-wide optimal frequency setting



(a) Component-wise system architecture of PoweRock

(b) Three-layer hierarchical DVFS controller design

(c) Workflow of real-time power request synchronizer (RPRS)

Fig. 8. The layered design and system architecture of PoweRock for many-core chips with multiple clock/voltage domains

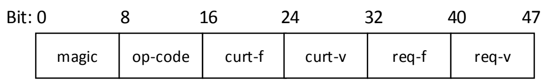


Fig. 9. Format of the inter-core power request message

and sets it, provided that the current voltage is high enough to support the new frequency. If the current voltage is not high enough, we have to scale up the voltage first. The workflow of voltage scaling is similar to frequency scaling. Meanwhile, if the VMaster finds that the current voltage appears too high for the current frequency settings in the voltage domain, it will scale down the voltage for power saving. The domain-wide power setting adopts an "arithmetic mean" policy just as Ioannou *et al.* [9] did. That means the power of a domain is set as the arithmetic mean of the frequencies or voltages requested by all the cores in the domain.

To synchronize power requests as soon as possible without waiting for the system to schedule, RPRS implements a real-time, interrupt-based, on-chip communication mechanism leveraging the on-die *message passing buffer (MPB)* and *inter-processor-interrupt (IPI)* provided by the SCC hardware. Each power request is packed as a 48-bit message and written to the local MPB of the destination core. Fig. 9 shows the message format. The 8-bit magic number is used to distinguish power requests from other MPB messages. The op-code denotes the request's operation type—frequency scaling or voltage scaling. The curt-f and curt-v (req-f and req-v) fields denote the current (requested) voltage and frequency settings of the source core.

V. Evaluation Methodology

A. Experimental Setting

To evaluate our PoweRock system implementing the power model and profile-guided DVFS scheme, we port several well-known benchmarks to our Rhymes SVM. We use gcc version 4.4.3 to compile programs with the O3 optimization level. The execution performance (runtime), power consumption, energy and EDP of the benchmark executions are inspected for experimental comparison. During the experiments, the clock frequencies of the network-on-chip (NoC) and memory controllers (MCs) are both fixed at 800MHz. As the temperature of the SCC board is kept around 40 °C, we do not consider the temperature's impact on the chip power in this paper. In our experiments, energy consumption by a program is the product of runtime and the average chip power during the execution. EDP is the product of energy and runtime. The efficiency of our DPM solution is evaluated by setting the optimization goal towards the least EDP and comparing with a static power case (800MHz/1.1V for all cores) in which DVFS is disabled.

We evaluate our approach using seven benchmarks with their problem sizes described in Table IV. Graph 500 [27] and Malstone [28] are data-intensive (or memory-bound) computing benchmarks. LU and SOR are ported from the SPLASH2 benchmark suite [29] whereas IS and EP are from the NPB suite [30]. Genome is a genetic analysis program that we develop. By comparing two arrays representing DNA sequences, the program logic is to locate a disease gene pattern on a human. All programs were run on 48 cores of the SCC.

B. Power Measurement

To avoid overhead introduced by power measurement, we measure the power consumption of the SCC outside the chip. We view the chip as a "black box" and read the current and voltage values from the board management controller (BMC) using the management console PC (MCPC) during the application run. As shown in Fig. 10, the BMC is connected via Ethernet to the MCPC, from which the power values are read. PCIE bus connects the SCC to the MCPC. The phase

TABLE IV
BENCHMARKS FOR EVALUATION AND THEIR PARAMETERS

Benchmark	Problem Size
Graph 500	Scale: 18 (262144 vertices); Edgefactor: 16 (4194304 edges)
Malstone	300000 records
LU	1024 × 1024 matrix 16 × 16 block
SOR	2048 × 1024 matrix
IS	Class A
EP	Class A
Genome	Human DNA length: 67.1MB; disease gene length: 8KB

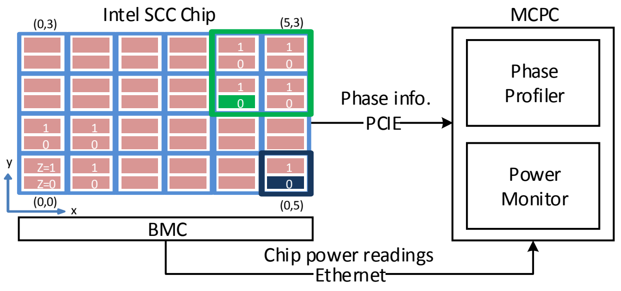


Fig. 10. The chip power measurement setting for Intel SCC

profiling information is obtained from the PCIE connection. The power states of the chip will be recorded into a log file on the MCPC. Although the maximum power sampling rate is only about 3.3 samples per second, it suffices for our evaluation. The average power is estimated by taking arithmetic mean of all sampled power values. In this paper, we use energy consumption and energy-delay product (EDP), both are widely adopted, to evaluate our DPM approach.

VI. Experimental Results and Analysis

A. Evaluation of the Power Model

To evaluate our power model, we compare the estimated chip powers using our model and the power model proposed by Sadri *et al.* [14]. Sadri *et al.* proposed a single-core power modeling approach for Intel SCC. Their model also considered the CPU frequency and program execution pattern but not the supplied voltage. They split the core power P_i into idle and active parts, $P_i = P_{i,idle} + P_{i,active}$, and propose the fitting functions for them as (12) and (13).

$$P_{i,idle} = p + q \cdot f_i \tag{12}$$

$$P_{i,active} = (a+b\times CPI^{c})\times f_i+(a'+b'\times CPI^{c'})\times f_i+d \tag{13}$$

In (12) and (13), p, q, a, b, c, d, a', b' and c' are constant coefficients, while f_i (unit: GHz) is the clock frequency of core i. They use CPI (clock per instruction)—the multiplicative inverse of IPC—to represent the program pattern and measure the power of a single core in different frequency and program pattern settings. Based on the measurements, they use a least-square optimization algorithm to find the coefficients of the fitting power function, giving the core power model as (14).

$$P_i = (0.38-0.24\cdot CPI^{0.085}+0.22\cdot CPI^{0.02})\cdot f_i+0.41 \tag{14}$$

According to their modeling approach, we can assume the power consumed by the rest of the chip, apart from the cores, equals the chip power when no core is active. By the measurement in their paper, we figure out this idle power is about 34.0W. So the chip power (P) can be estimated by summing the power of every core and this 34.0W as in (15).

$$P = \sum_{i=0}^{N-1} P_i + 34.0 \tag{15}$$

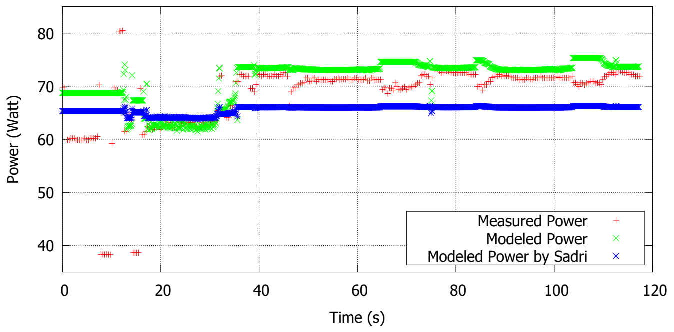


Fig. 11. Estimated and measured chip power of Graph 500 running on 48 cores at static CPU frequency of 800MHz

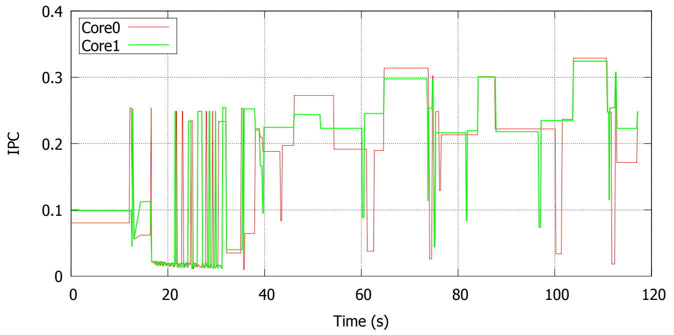


Fig. 12. The IPC of core0 and core1 during Graph 500 execution

We conduct the following experiment to compare our power model with Sadri's one. As we can only get measured power of the entire chip, the comparison between the models is also at chip level. We obtain the real and estimated chip powers as follows. First, we run Graph500 on 48 cores at static 800MHz/1.1V, and profile each core's PMCs for CPI (or IPC) during the execution. The actual chip power is monitored using the mechanism provided by the Intel SCC platform [22]. Then the estimated chip powers are derived from the profiled CPI (or IPC) using our model (9) and Sadri's model (15).

The measured chip power and the modeled ones are plotted as Fig. 11. Comparing with the measured power, our modeled power is obviously closer to it than Sadri's one. During the Graph 500 execution, the chip power changes due to varying execution pattern. But the estimated power given by Sadri's model seems invariant despite the pattern variation.

There exist several periods (in the first 12s, and around 50s, 70s, 85s, 105s) during which our model shows more apparent deviations from the real power. This can be explained as follows. In the beginning 12 seconds, we can see both the models give big errors in the estimated powers. This is because this part of execution (performing data-race-free buffer allocation and edge list generation) was taken as a

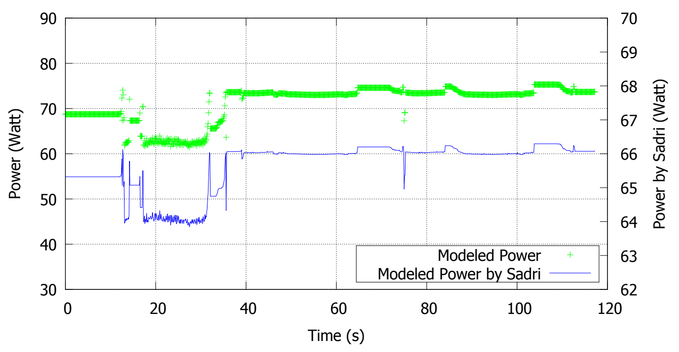


Fig. 13. Comparison with Sadri's model with its y-axis scale zoomed in

single phase (no locks in between for phase partitioning). The modeled power was taken as the average power of this phase while the actual patterns keep varying in this time range. After 17s passed, the estimated power becomes more accurate when the program becomes more fully-loaded. For the other four places (i.e. 50s, 70s, 85s and 105s), we find the clue by analyzing the IPC pattern of the execution. We plot the instantaneous IPC variation in Fig. 12 which shows that the IPCs of both master and slave cores are relatively high and all around 0.3 instructions/sec at all the four places. Referring back to Fig. 2, we do not have enough samples with IPC around 0.3 to specify $g(IPC)$. So we attribute the deviations to insufficient sampling for best-fitting the curve in Fig. 2 (not a design problem of our model). Our power model is still very accurate for low IPC range in the SCC case study.

As mentioned before, Sadri's modeled power was just "too stable". But zooming in on its power-time curve, we observe some interesting facts. Fig. 13 shows our modeled power and Sadri's modeled power measured in the left and right y-tics respectively. The scale on the right is zoomed in by 7.5 times. Then we can see the variation patterns or trends of both curves are actually very similar and fit the measured power well. The only difference is the granularity. This observation implies two concerns. On one hand, it is definitely necessary to take the program pattern into account. As we can see in Fig. 11, even in a static power setting, the instantaneous chip power keeps changing in fairly large range. But both models can fit the trend of changes well, implying IPC is a good metric to characterize the program execution pattern. On the other hand, our power modeling approach is more accurate when modeling the chip power of many-core chips. Measuring through Sadri's approach, the single-core power does not include the static power, but they still take the measured single-core powers as if they were the sums of the static power and dynamic power (idle power and active power). This may be the reason why their model is found not accurate enough.

B. Evaluation of the Profile-guided DVFS Implementation

This part of evaluation is done by comparing application benchmarking results obtained from our profile-guided DVFS model and the baseline static power model. Table V presents the runtime, average power and the counts of phases with different types for each benchmark executed at 800MHz for all the 48 cores. Although the V/f settings are kept the same, Table V shows that the average powers consumed by the benchmarks are very different (mainly due to their different memory access patterns). Actually, the more compute-intensive the benchmark program, the larger the average power. For example, EP and Genome are the most compute-intensive and consume the largest average powers.

TABLE V
RUNTIME STATISTICS OF THE BENCHMARKS (#PHASES IS CORE0'S)

Benchmark	Runtime (s)	Avg_power (W)	$\#P_{comp}$	$\#P_{bar}$	$\#P_{lock}$
Graph500	28.09	62.90	137	34	102
Malstone	62.15	70.02	4	3	0
LU	8.90	66.71	135	134	0
SOR	21.22	64.82	5	4	0
IS	45.79	68.62	26	25	0
EP	13.11	88.82	15	2	12
Genome	158.37	99.45	5	4	0

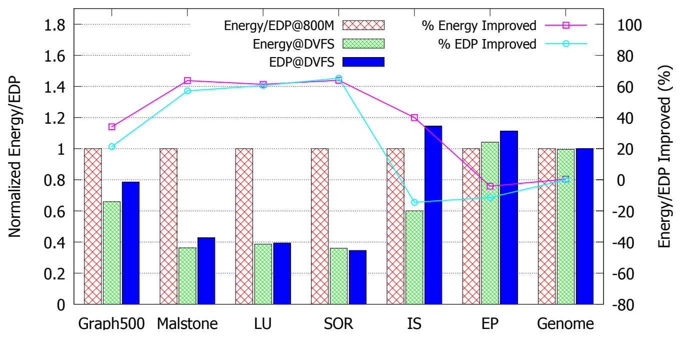
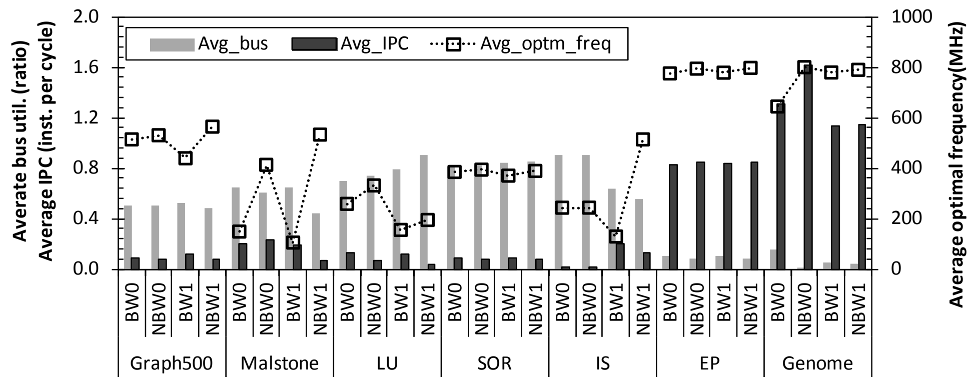


Fig. 14. Comparison of our profile-guided scheme and static power mode in terms of energy and EDP both normalized to the static-mode values.

Fig. 14 presents the energy/EDP improvement of different programs with our profile-guided DPM enabled. The energy/EDP values are normalized to the static power values (800MHz/1.1V). Most of the cases gain energy or EDP benefit. The average energy saving and EDP reduction are 25.5% and 37.1% respectively. SOR makes the best case—up to 64.0% energy and 65.4% EDP saved. Some cases like EP and Genome, which are the most CPU-intensive, do not gain any energy or EDP benefit; enabling profile-guided DVFS imposes adverse effect on them instead. The reason behind this is related to DVFS scaling overheads. EP is by nature a highly CPU-bound program, so scaling down the clock frequency just means seriously penalizing the runtime, hence augmenting the energy consumption or EDP. For Genome running at the specified problem size (see Table IV), the program behavior is also highly compute-intensive since the whole 8KB-long disease gene array can be totally cached in L1 most of the time during comparison with the human DNA data.

1) Analysis: For further understanding of the experimental results, we present in Fig. 15 the program patterns in terms of average bus utilization and IPC. Larger bus utilization implies more off-core activities including memory access whereas larger IPC means that instructions execute faster and consume more power. To wipe out the impact of "busy waiting" on the program pattern, we also present the average values without the P_{bar} and P_{lock} phases.

From Fig. 15, we can see that Graph 500, Malstone, LU,



- BW0 (BW1) refers core0's (core1's) data with busy waiting phases, i.e. P_{bar}'s and P_{lock}'s;
- NBW0 (NBW1) refers core0's (core1's) data without busy waiting phases.

Fig. 15. Average bus utilization, IPC and optimal frequency of benchmark programs

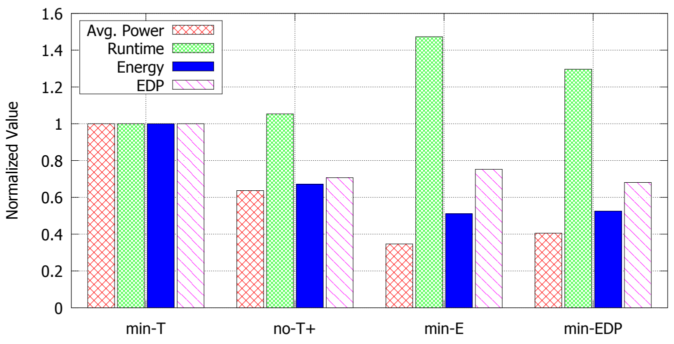


Fig. 16. Normalized measurements of Graph 500 under four DPM policies

SOR, and IS are memory-intensive because of their high "Avg-bus" values. All these four programs reported substantial energy saving under our DPM scheme (see Fig. 14). Both EP and Genome have low bus utilization and high IPC, implying they are highly CPU-bound and leave little opportunity for energy saving. In line with this conjecture, the average optimal frequencies for them are almost the highest frequency.

C. Evaluation of the Flexibility of the DPM System

To evaluate our flexible DPM solution with different goals, we devise four DPM policies with different goals:

- Min-T: aims at the minimum runtime or best performance without considering power consumption; all cores are set to run in the highest power state of 800MHz/1.1V.
- No-T+: aims to save as much power as possible with no or little performance loss. To achieve this goal, we assign the highest frequency to all P_{comp} phases and the lowest frequency to all P_{bar} and P_{lock} phases.
- Min-E: aims at the least energy consumption. Optimal power levels are tuned towards the goal of minimizing energy for all P_{comp} phases using our models.
- Min-EDP: aims at the least EDP for the execution; optimal power levels are chosen for minimizing the EDP.

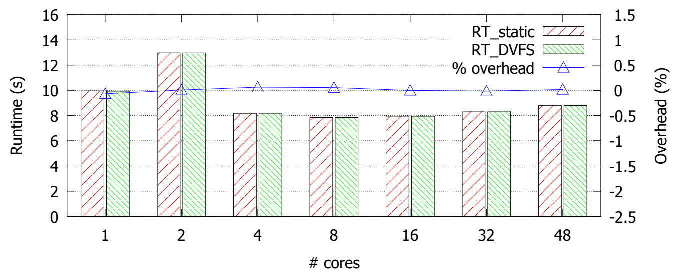
For each policy, we record the runtime and chip power of each execution of Graph 500. As shown in Fig. 16, our DPM system is capable of meeting the expected DPM goals flexibly—when the goal is adjusted, the system won't give noticeable drops in energy-performance indexes. Min-T gets the best performance as expected, but at the cost of the highest average power. No-T+ achieves good energy saving (32.9%) and EDP reduction (29.3%) with a slight slowdown (5.4%). Min-E gives the least energy consumption in all the cases (48.9% compared to min-T), yet thwarting the performance most seriously. Min-EDP achieves the least EDP with much less performance loss than min-E.

D. Overhead and Scalability Analysis

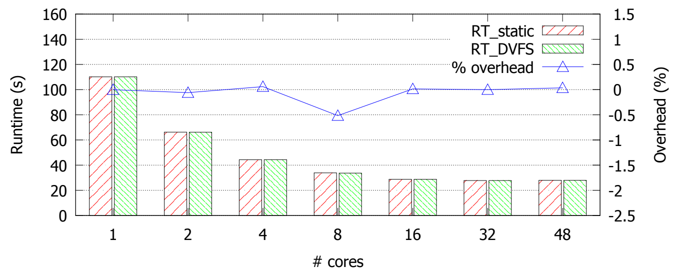
Using PoweRock may introduce performance overheads to the application. There are three major time costs with it: C1) the cost of context switching between user space and kernel space when DPM routines are called; C2) slowdown of domain masters which are being interrupted by power requests for making DVFS decisions; C3) V/f scaling latency of which the voltage switching time dominates. These costs determine the scalability of PoweRock. To measure the overhead of our system, experimental comparison between static power mode and profile-guided DVFS mode is performed. However, for the DVFS mode, we cannot really change the V/f setting or else the execution performance will be affected by frequency scaling. We choose to skip writing the frequency/voltage control registers. As a result, the measured overhead does not include C3, but the rest like C1 and C2 can be revealed.

Fig. 17 shows the results obtained from LU and Graph 500. We choose them for overhead analysis as they have up to 300 phases triggering lots of DPM actions for measurement. To assess the scalability, experiments are run on 1, 2, 4, 8, 16, 32 and 48 cores. The results show that our DPM implementation incurs almost imperceptible runtime overhead (all below 1%), and does not affect the scalability of the programs.

Memory overhead of PoweRock can also be said negligible. The system needs extra memory space mainly for storing the power control profiles. In our current design, we need to keep track of phase id, phase type and optimal power level for each phase—at least 4×3 bytes per phase is needed. The number of phases in the cases we tested did not exceed 300. Each master core and slave core require keeping 3,600 bytes at most.



(a) DPM overhead measured with blocked dense LU



(b) DPM overhead measured with Graph 500

Fig. 17. Performance overhead analysis for our DPM solution

VII. Related Work

A. Power Modeling for Multicore/Many-core Processors

Chip power modeling [11]–[15] is a challenging problem, particularly when processor chip designs are going many-core. A few prior works [16], [17] tried to characterize the power-performance behavior of the SCC processor in different power settings. For instance, Bartolini *et al.* [17] evaluated the impact of DVFS on the speed and power usage of MPI programs. However, these studies only presented power/performance results in different power settings rather than proposing any analytical power model for prediction purposes.

A handful of previous works [13]–[15] did build up analytic power models but sadly omitted some key elements for modeling the power consumption. Sadri *et al.* [14] proposed a power model to estimate the per-core power for thermal management on the SCC. However, they did not take voltage into account in their model. Cichowski *et al.* [15] also proposed a power model for the SCC, but they did not consider the impact of program pattern on the chip power consumption. Li *et al.* [13] presented a power model for multicore systems. Their model used IPS (instructions per second) rather than IPC to represent the activity factor when the frequency changes. Although IPS implies consideration of frequency in some ways, their model did not consider the voltage setting as was the case of Sadri's. After a thorough survey, our power model takes an all-round design considering all the key elements like clock frequency, supplied voltage and power-related patterns of the program execution (tracked via monitoring performance counters).

B. Power Management on Multicore/Many-core Systems

There is rich literature on power management for multicore or many-core systems [5]–[10]. Most of the previous works focused on power capping [5]–[8] or energy saving under a performance loss constraint [9], [10]. Pack & Cap [5], is a control technique designed to make optimal DVFS and thread packing control decisions for maximizing performance within a power budget. They built a classifier to accurately select the optimal power settings and pack threads with the same power-performance characteristics. Ma *et al.* [8] presented a scalable power control solution for many-core microprocessors that is specifically designed to handle realistic workloads. They focused on distributing the chip power budget to each core by a three-layer controller in order to optimize the overall performance. Kappiah *et al.* [10] made use of the slack time in MPI programs for reducing the frequency on nodes that are not in the critical path. Their scaling of frequency was based on the heuristic information of the program without considering the power consumption while scaling performance.

Our work, on the other hand, focuses on the flexibility of the DPM framework for different optimization goals (configurable at deployment time). Freeh *et al.* [3] and Ioannou *et al.* [9] were pursuing a line of research most relevant to ours, but they targeted MPI programs while we aimed at programmer-friendly shared-memory paradigms. Freeh *et al.* [3] presented a framework for executing a single MPI application in several V/f settings. Their basic idea is to divide a program into phases each of which is assigned a prescribed frequency. Our profiling method is quite similar to Freeh's, but our profiling is more lightweight and universal for most parallel applications. Freeh *et al.* used a brute force to explore the best per-phase V/f settings by multiple experimental runs. Our design enables us to easily choose the optimal V/f setting by one-off calculation. The DPM scheme proposed by Ioannou *et al.* [9] tried to minimize energy consumption within a performance window for the Intel SCC. They employed a trial-and-error approach to estimating the optimal power states and imposed limits on frequency scale-downs by setting an empirical performance threshold. This method is prone to exclusion of some perfect power-saving opportunities during the trial period.

VIII. Conclusion

Entering a many-core computing era, effective power management solutions have a vital role in reshaping today's power-hungry supercomputing. Advances in DPM/DVFS technologies should address several new challenges. First, they should be designed (scalable) for many-core processors since the state-of-the-art supercomputers can have much of the system power (over 60% in the case of Tianhe-2) dissipated in the abundance of compute cores. Second, an accurate and all-round chip power-performance model should be available for predicting the optimal power states for different parts of a program execution with varying compute/communication patterns. With this foundation, we can build a flexible DPM solution that has separation of mechanism and policy of power control, accommodating variable or reconfigurable DPM goals. This paper reports our latest research efforts making a complete solution—*PoweRock*—to cope with these challenges. We proposed new and practical prediction model linking power states (voltage/frequency settings) with performance characteristics (IPC and bus utilization). By using the models and a lightweight profiling approach, PoweRock can rapidly derive a power control profile containing per-phase optimal V/f settings for (virtual) shared memory applications. We designed a hierarchical domain-aware DVFS scheduling framework tailored to many-core architectures with multiple voltage/clock domains, and apply the concept to implement a scalable DVFS controller at the kernel level. Finally, we build a profile-guided power manager driving the DVFS controller, and ship with a user-space PoweRock API for easing the development of green applications or middleware systems. We implement and evaluate PoweRock on the Barrelfish operating system for Intel's SCC many-core processor. Experimental benchmarking results show that our solution models the power accurately across changes in DPM goals and gains up to 65% energy or EDP benefit over the static power scheme.

Acknowledgment

This work is supported by Hong Kong RGC grant HKU 716712E, Program for Changjiang Scholars and Innovative Research Team in University (PCSIRT, No. IRT1012) and Aid Program for Science and Technology Innovative Research Team in Higher Educational Institutions of Hunan Province (No. 11JJ7003). Special thanks go to Intel China Center of Parallel Computing (ICCPC) and Beijing Soft Tech Technologies Co., Ltd. for providing us with support services of the SCC platform in their Wuxi data center.

References

[1] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick, "The landscape of parallel computing research: A view from berkeley," EECS Depart., University of California, Berkeley, Tech. Rep. UCB/EECS-2006-183, 2006.

[2] "Top500 list - June 2014." [Online]. Available: http://www.top500.org/lists/2014/06/

[3] V. W. Freeh and D. K. Lowenthal, "Using multiple energy gears in MPI programs on a power-scalable cluster," in *Proc. 10th ACM SIGPLAN Symp. Principles and Practice of Parallel Programming (PPoPP'05)*, 2005, pp. 164–173.

[4] D. Lo and C. Kozyrakis, "Dynamic management of TurboMode in modern multi-core chips," in *Proc. 20th Int. Symp. High Performance Comput. Architecture (HPCA'14)*, 2014, pp. 603–613.

[5] R. Cochran, C. Hankendi, A. K. Coskun, and S. Reda, "Pack & Cap: adaptive DVFS and thread packing under power caps," in *Proc. 44th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO-44)*, 2011, pp. 175–185.

[6] Y. Liu, H. Zhu, K. Lu, and Y. Liu, "A power provision and capping architecture for large scale systems," in *Proc. 26th Int. Parallel and Distributed Process. Symp. Workshops & PhD Forum (IPDPSW)*, 2012, pp. 954–963.

[7] 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 *Proc. 39th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO-39)*, 2006, pp. 347–358.

[8] K. Ma, X. Li, M. Chen, and X. Wang, "Scalable power control for many-core architectures running multi-threaded applications," in *Proc. 38th Int. Symp. Comput. Architecture (ISCA'11)*, 2011, pp. 449–460.

[9] N. Ioannou, M. Kauschke, M. Gries, and M. Cintra, "Phase-based application-driven hierarchical power management on the Single-chip Cloud Computer," in *Proc. 20th Int. Conf. Parallel Architectures and Compilation Techniques (PACT'11)*, 2011, pp. 131–142.

[10] N. Kappiah, V. W. Freeh, and D. K. Lowenthal, "Just in time dynamic voltage scaling: Exploiting inter-node slack to save energy in MPI programs," in *Proc. ACM/IEEE Conf. Supercomputing (SC'05)*, 2005, pp. 33–41.

[11] A. Weissel and F. Bellosa, "Process cruise control: Event-driven clock scaling for dynamic power management," in *Proc. Int. Conf. Compilers, Architecture and Synthesis for Embedded Syst. (CASES'02)*, 2002, pp. 238–246.

[12] C. Isci and M. Martonosi, "Runtime power monitoring in high-end processors: Methodology and empirical data," in *Proc. 36th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO-36)*, 2003, pp. 93–104.

[13] D. Li, B. R. d. Supinski, M. Schulz, D. S. Nikolopoulos, and K. W. Cameron, "Strategies for energy-efficient resource management of hybrid programming models," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 1, pp. 144–157, 2013.

[14] M. Sadri, A. Bartolini, and L. Benini, "Single-Chip Cloud Computer thermal model," in *Proc. 17th Int. Workshop on Thermal Investigations of ICs and Syst. (THERMINIC'11)*, 2011, pp. 1–6.

[15] P. Cichowski, J. Keller, and C. Kessler, "Modelling power consumption of the Intel SCC," in *Proc. Many-core Appl. Res. Community Symp. (MARC'12)*, 2012, pp. 46–51.

[16] P. Gschwandtner, T. Fahringer, and R. Prodan, "Performance analysis and benchmarking of the Intel SCC," in *Proc. IEEE Int. Conf. Cluster Computing (Cluster'11)*, 2011, pp. 139–149.

[17] A. Bartolini, M. Sadri, J.-N. Furst, A. K. Coskun, and L. Benini, "Quantifying the impact of frequency scaling on the energy efficiency of the single-chip cloud computer," in *Proc. Design, Automation Test in Europe Conf. and Exhibition (DATE'12)*, 2012, pp. 181–186.

[18] J. Howard, S. Dighe, S. Vangal, G. Ruhl, N. Borkar, S. Jain, V. Erraguntla, M. Konow, M. Riepen, M. Gries, G. Droege, T. Lund-Larsen, S. Steibl, S. Borkar, V. De, and R. V. D. Wijngaart, "A 48-core IA-32 message-passing processor in 45nm CMOS using on-die message passing and DVFS for performance and power scaling," *IEEE J. Solid-State Circuits*, vol. 46, no. 1, pp. 173–183, 2011.

[19] A. Baumann, P. Barham, P.-E. Dagand, T. Harris, R. Isaacs, S. Peter, T. Roscoe, A. Schüpbach, and A. Singhania, "The multikernel: A new OS architecture for scalable multicore systems," in *Proc. ACM SIGOPS 22nd Symp. Operating Syst. Principles (SOSP'09)*, 2009, pp. 29–44.

[20] T. Li and L. K. John, "Run-time modeling and estimation of operating system power consumption," in *Proc. ACM SIGMETRICS Int. Conf. on Measurement and Modeling of Comput. Syst.*, 2003, pp. 160–171.

[21] W. Bircher, J. Law, M. Valluri, and L. K. John, "Effective use of performance monitoring counters for run-time prediction of power," Elect. and Comput. Eng. Depart., University of Texas, Tech. Rep. TR-041104-01, 2004.

[22] "SCC external architecture specification (EAS) (revision 0.94)," Intel Labs, Tech. Rep., 2010.

[23] K. T. Lam, J. Shi, D. Hung, C.-L. Wang, Y. Yan, and W. Zhu, "Rhymes: A shared virtual memory system for non-coherent tiled many-core architectures," in *Proc. 20th IEEE Int. Conf. Parallel and Distributed Syst. (ICPADS'14)*, 2014, (in press).

[24] Z. Lai, K. T. Lam, C.-L. Wang, J. Su, Y. Yan, and W. Zhu, "Latency-aware dynamic voltage and frequency scaling on many-core architectures for data-intensive applications," in *Proc. Int. Conf. Cloud Computing and Big Data (CloudCom-Asia'13)*, 2013, pp. 78–83.

[25] G. Magklis, M. L. Scott, G. Semeraro, D. H. Albonesi, and S. Dropsho, "Profile-based dynamic voltage and frequency scaling for a multiple clock domain microprocessor," in *Proc. 30th Int. Symp. Comput. Architecture (ISCA'03)*, 2003, pp. 14–25.

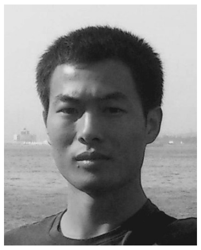
[26] Intel, *Pentium® Processor Family Developer's Manual Volume 3: Architecture and Programming Manual*, 1995.

[27] "Graph 500 benchmark." [Online]. Available: http://www.graph500.org

[28] C. Bennett, R. L. Grossman, D. Locke, J. Seidman, and S. Vejcik, "Malstone: towards a benchmark for analytics on large data clouds," in *Proc. 16th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, 2010, pp. 145–152.

[29] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," *SIGARCH Comput. Archit. News*, vol. 23, no. 2, pp. 24–36, 1995.

[30] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga, "The NAS parallel benchmarks—summary and preliminary results," in *Proc. ACM/IEEE Conf. Supercomputing (SC'91)*, 1991, pp. 158–165.



Zhiquan Lai received his M.S. and B.S. degrees in Computer Science from National University of Defense Technology (NUDT) in 2010 and 2008 respectively. He is currently a Ph.D. student at NUDT with Prof. Jinshu Su being his advisor. This work was mainly done while he worked as a research assistant at Department of Computer Science, the University of Hong Kong during Oct. 2012 to Oct. 2013. His current research interests include high-performance system software, power modeling, power-aware computing and many-core computing.

King Tin Lam received his Ph.D. and M.Sc. degrees in Computer Science and B.Eng. degree in Electrical and Electronic Engineering from the University of Hong Kong in 2012, 2006 and 2001 respectively. He is currently a postdoctoral research fellow in the Department of Computer Science at the University of Hong Kong. His research interests include parallel and distributed computing systems, cluster and cloud computing, memory coherence protocols, GPGPU and many-core computing.

Cho-Li Wang received his Ph.D. degree in Computer Engineering from University of Southern California in 1995. He is a full professor in the Department of Computer Science at the University of Hong Kong. His research interests include parallel architecture, software systems for cluster and grid computing, and virtualization techniques for cloud computing. He has served on the editorial boards of several international journals, including IEEE Transactions on Computers (2006-2010), Journal of Information Science and Engineering, and Multiagent and Grid Systems.

Jinshu Su received the Ph.D. and M.S. degrees, both in Computer Science, from National University of Defense Technology (NUDT) in 2000 and 1988 respectively, and the B.S degree in Mathematics from Nankai University in 1983. He is a full professor in the College of Computer, National University of Defense Technology, and serves as a head of the Institute of Network and Information Security, NUDT. He has led several national key projects of China, including 973, 863 and NSFC Key projects. His research interests include high performance routers, internet routing, high performance computing, wireless networks and information security. He is a member of the IEEE and ACM, and a senior member of China Computer Federation (CCF).