

Phase-Based Application-Driven Hierarchical Power Management on the Single-chip Cloud Computer*

Nikolas Ioannou^{†‡}, Michael Kauschke[‡], Matthias Gries[‡], and Marcelo Cintra[†]

[†] School of Informatics
University of Edinburgh

{nikolas.ioannou|mc@staffmail}.ed.ac.uk

[‡] Germany Microprocessor Lab
Intel Labs Braunschweig

{michael.kauschke|matthias.gries}@intel.com

ABSTRACT

To improve energy efficiency processors allow for Dynamic Voltage and Frequency Scaling (DVFS), which enables changing their performance and power consumption on-the-fly. Many-core architectures, such as the Single-chip Cloud Computer (SCC) experimental processor from Intel Labs, have DVFS infrastructures that scale by having many more independent voltage and frequency domains on-die than today's multi-cores.

This paper proposes a novel, hierarchical, and transparent client-server power management scheme applicable to such architectures. The scheme tries to minimize energy consumption within a performance window taking into consideration not only the local information for cores within frequency domains but also information that spans multiple frequency and voltage domains.

We implement our proposed hierarchical power control using a novel application-driven phase detection and prediction approach for Message Passing Interface (MPI) applications, a natural choice on the SCC with its fast on-chip network and its non-coherent memory hierarchy. This phase predictor operates as the front-end to the hierarchical DVFS controller, providing the necessary DVFS scheduling points.

Experimental results with SCC hardware show that our approach provides significant improvement of the Energy Delay Product (EDP) of as much as 27.2%, and 11.4% on average, with an average increase in execution time of 7.7% over a baseline version without DVFS. These improvements come from both improved phase prediction accuracy and more effective DVFS control of the domains, compared to existing approaches.

Keywords: phase detection, message passing interface, DVFS, resource management

*Work mainly done while N. Ioannou was an intern at Intel Labs. M. Cintra is currently on sabbatical leave at Intel Labs. M. Kauschke is now affiliated with Intel Corp. and the ExaCluster Laboratory at FZ Jülich, Germany.

1 INTRODUCTION

Due to process scaling and the capability of integrating more transistors on a die, chip designs are becoming increasingly constrained by thermal limits. This in turn means that power has become a scarce resource that needs to be carefully allocated. One way of manufacturers addressing the need for reducing power consumption is by offering on-die infrastructure for Dynamic Voltage and Frequency Scaling (DVFS). In traditional multi-chip multiprocessors and multi-core systems, DVFS has been shown to decrease power consumption of individual cores when it is deemed that this will not adversely affect the performance of the programs running on the system.

Many-core processors, such as Intel Labs' Single-chip Cloud Computer (SCC) experimental processor [10], a 48-core "concept vehicle" created as a platform for many-core software research, consist of a very large number of individual cores and are capable of running workloads with high degrees of multi-programming or multi-threading. However, the following many-core trends impose challenges to the effective use of power control through DVFS and call for novel approaches:

- Although the number of voltage and frequency domains on a chip increases, they may be outpaced by the increase in the number of cores. The required circuitry for decoupling and converting voltages and frequencies on silicon may not scale as quickly as logic and memory transistors [14]. In the SCC several small processing cores are clustered in voltage and frequency domains and all cores on a domain must agree on a common voltage (frequency) setting. This clustered approach to DVFS domains is a way forward for many-cores [20].
- Changing the frequency state of a domain remains inexpensive as changes occur almost immediately. However, changing voltage requires regulators to output a stable target voltage. This remains a relatively costly process and needs to be scheduled carefully.

- Many-core architectures become attractive for highly parallel workloads that require more involved DVFS control than multi-programmed workloads because of the frequent interactions (communication and synchronization) across processes. As a result, DVFS control must be coordinated among all cores running the application.

Prior work has applied DVFS to single cores [11, 17, 23], clusters of multi-chip multiprocessors [7, 15, 22, 25], and shared-memory multi-cores [1, 3, 4, 12]. These studies have the underlying assumption that each core has full, independent control over its voltage and frequency settings, whereas the design space for several cores sharing one DVFS domain has not been explored much.

We propose a novel, hierarchical, and transparent DVFS power management scheme to accommodate this new architectural opportunity. The scheme applies a client-server approach and attempts to minimize energy consumption within a performance window. The key idea is to separate the power manager into a *local* “client” component, which identifies the best setting for a core, and a *domain* “server” component, which performs the actual DVFS changes taking into consideration not only the local information within a frequency domain, but also information that spans multiple frequency domains.

As a front-end to the power manager, we employ a new run-time and application-level phase detection algorithm to drive DVFS decisions. This is based on our observation that phases defined by MPI [19] calls in highly parallel scientific applications provide good coverage for predicting the run-time behavior of these workloads. Program phases have been extensively studied in prior work [2, 5, 6, 24]. Our phase detector is based on a novel phase detection and prediction approach for MPI applications. The key idea is to track recurring phases at an MPI-event granularity by detecting and encoding maximal repeating sequences. Finally, our approach to power management separates the DVFS manager from the phase detector, such that different phase detection modules (front-end) and DVFS power managers (back-end) can be seamlessly plugged-in.

We evaluate our scheme on Intel Labs’ experimental Single-chip Cloud Computer (SCC). The 48-core SCC provides voltage scaling at the granularity of 8-core domains and frequency scaling at the granularity of 2-core domains. In addition, workloads using the Message Passing paradigm are a natural choice on the SCC with its fast on-chip network and its non-coherent memory hierarchy. Finally, the SCC conveniently exposes control knobs of voltage and frequency to user software. Experimental results with SCC hardware show that our approach provides significant improvement of the Energy Delay Product (EDP) of as much as 27.2%, and 11.4% on average, with an increase in execution time of 7.7% on average. Furthermore, our scheme performs well within the 3:1 ratio of *power savings* : *performance degradation* that is nominally expected in the field [12].

This paper makes the following contributions:

- We present a novel hierarchical scheme for power management that accommodates current trends in many-core architectures. The scheme allows for coordinated power management for large-scale parallel applications running on several cores with voltage and frequency control at the granularity of multiple core domains.
- We present a novel and highly-accurate phase detection and prediction algorithm for MPI applications that is able to detect phases at an MPI-event granularity – the SuperMaximal Repeat phase Predictor (SMRP). This predictor is used as an application-driven front-end to our power management scheme.
- We present experimental results for MPI phase detection and DVFS on the SCC. The results discussed in this paper represent real-world numbers taken on a real system with a working implementation of the proposed hierarchical scheme.

The rest of the paper is organized as follows: Section 2 presents the SCC architecture and necessary background information; Section 3 introduces our hierarchical power management approach; Section 4 elaborates on phases in MPI applications and introduces our phase detection algorithm; Section 5 presents the experimental setup used; Section 6 presents quantitative results; Section 7 discusses the related work; and Section 8 concludes the paper.

2 BACKGROUND

DVFS Dynamic power P_{dy} dissipated by a chip is strongly dependent on supply voltage V_{dd} and operating frequency f : $P_{dy} \propto V_{dd}^2 f$. By reducing the voltage by a small amount, dynamic power is reduced by the square of that factor. However, reducing the voltage means that transistors need more time to switch on and off, which forces a reduction in the operating frequency. Dynamic Voltage and Frequency Scaling (DVFS) [16] exploits this relationship by reducing the voltage and the clock frequency when this can be done without experiencing a proportional reduction in performance. Reducing frequency can usually be done quickly, whereas for changing voltage the regulators have to settle their output voltage. Changes in voltage must, thus, be carefully scheduled in advance to align ramping up voltage with activity in the chip.

MPI The Message Passing Interface (MPI) [19] is a programming abstraction that provides a virtual topology and mechanisms for synchronization and communication between a set of ranks (usually mapped one-to-one to cores). The programmer explicitly specifies communication and synchronization patterns in the algorithm implementation by instantiating MPI calls (such as sends, receives, and collective operations). In most systems, and in the SCC alike, MPI

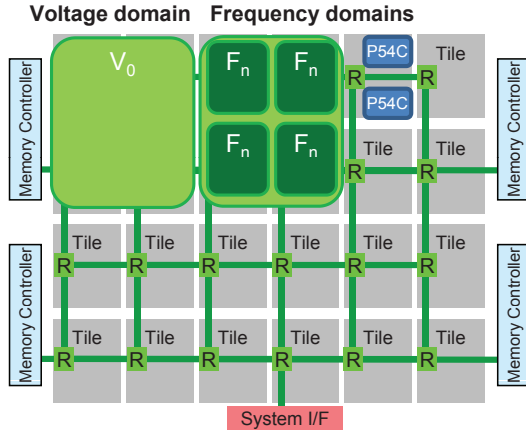


Figure 1: Voltage and frequency domains on the SCC.

calls are executed in the same core as the user application and are, thus, amenable to and affected by voltage and frequency changes in the same way as the rest of the user application.

SCC by Intel Labs The Single-chip Cloud Computer (SCC) experimental processor [10] is a 48-core concept vehicle created by Intel Labs as a platform for many-core software research. Each core is an independent Pentium® core with local, non-coherent, memory subsystem. The SCC is thus a natural target for MPI-based applications, as explored in this paper. The cores are organized in pairs (tiles) for frequency scaling and in groups of four tiles for voltage scaling (see Figure 1). Each voltage domain thus controls eight cores, whereas each tile (two cores) can run at a different frequency. Each tile has a specific register for frequency control and status. Voltage regulators are controlled through a register interface. SCC has no restrictions on which cores are allowed to trigger voltage and frequency changes. Voltage changes can happen in the order of 1ms, whereas frequency changes can happen within a few clock cycles. However, in SCC only a single voltage change can be effected at any given time.

3 HIERARCHICAL POWER MANAGEMENT

Our hierarchical power management scheme is tailored to work with many-core architectures that have DVFS support based on frequency and voltage domains. Our scheme is based around the existence of repeatable phases within a program and tries to exploit possible amenability of these phases to different voltage and frequency operating points. Any predictor that is able to detect repeatable behavior and project future phases can be used as the front-end to our power management scheme (Figure 2a). In this work we evaluate our scheme on the SCC many-core platform using our new phase

Frequency (MHz)	Voltage (V)
800	1.1
533	0.9
400	0.8
320	0.7

Table 1: Frequency and Voltage pairs of operation points for the SCC.

predictor described in Section 4 for MPI applications.

The requirement to control the frequency and voltage of the cores at the granularity of domains led us to choose a hierarchical power management scheme. At the lower level a local power management system, one per core, employs a controller algorithm based on which it places requests for desired voltage and frequency values. A domain-level power management system services these requests and makes a final decision. A high level block diagram of this two-level scheme is presented in Figure 2b.

3.1 Local Power Management

Each core has a power management controller that takes as input the sub-phases identified by the phase predictor module¹. After the predictor has identified recurring phases in the program execution, it provides the power manager with reference points for DVFS scheduling decisions (Section 4.2). In order to amortize the non-negligible voltage transition costs, these reference points should be far away. Moving them too far apart, however, can lead to the inability to exploit finer grain program phases. Thus, a balance has to be achieved and we empirically found that a threshold of at least 20ms² should be employed between reference points on the SCC.

3.1.1 Determining DVFS Operating Points

Leveraging the repetitiveness of the phases, the power manager assigns to each sub-phase a frequency/execution time (f/t) table (see Figure 2b). The table is populated using a dynamic tracking technique similar to [11, 22]. We focus on execution time and frequency settings. As a first step for each sub-phase the highest possible frequency (here 800 MHz, see Table 1) is exercised and the respective execution time monitored and stored in the table. Afterwards, at the next occurrence of the said sub-phase the next lower frequency is employed and the execution time is again recorded in the table. Filling the table terminates when the resulting execution time penalty with respect to the maximum frequency setting exceeds a certain limit, δ . Due to the large gap between the frequency steps of the SCC, the performance threshold δ was empirically set to 10% (see discussion in Section 6.4). The

¹As explained later in Section 4, the phase predictor we developed identifies maximal repeating macro phases as well as sub-phases within them.

²On SCC, a DVFS decision affecting the whole chip can take up to 6ms.

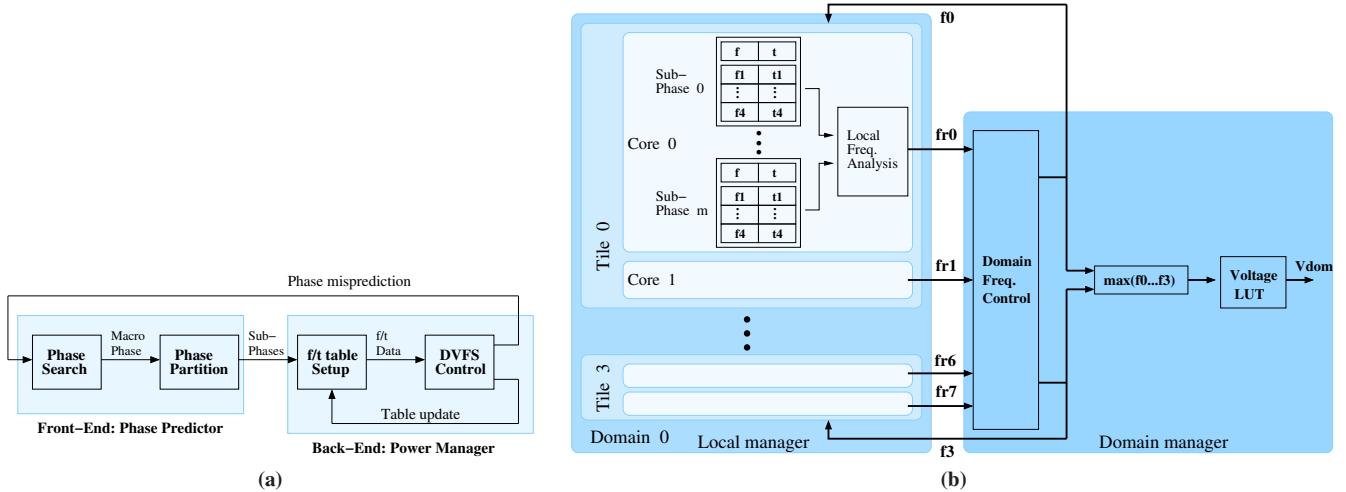


Figure 2: High level view of proposed hierarchical power management scheme: (a) State diagram of power manager and phase predictor. (b) Two level power management scheme.

tables are still dynamically updated while in operation. This may entail an update of already existing table entries but also population of empty entries in case the domain level power manager decides to choose frequencies for the entire domain (Section 3.2) not yet considered locally. In some cases this operation may lead to a new locally optimal frequency setting. This greedy algorithm employed by the local power manager is depicted in Figure 3a.

The local power manager employed in each of the cores does not directly control the voltage and frequency settings. It merely sends a request to the domain level power manager (the f_r signals in Figure 2b) that then makes a coarser grain decision and sets the frequency of the tiles (the f signals in Figure 2b) and the domain voltage (V_{dom} in Figure 2b). The current frequency setting for each tile is easily read with a single access to a virtual memory location, different for each tile.

3.1.2 Example

Let us assume that the local power manager requests a reduction of frequency from 800MHz to 533MHz for a specific sub-phase and is granted this frequency. Assume that the execution times of this sub-phase for these frequencies are 100ms and 105ms, respectively. In this case, our algorithm will fill the execution time table entry of 533MHz with 105ms and decide that the increase in execution time of 5% is lower than the selected δ . The next time this particular sub-phase is encountered, the power manager will again request a decrease in frequency, this time to 400MHz. Assume that once granted by the domain level power manager a further 10% decrease in performance is observed. Now the power manager will observe that the decrease in performance is greater than δ and, thus, not tolerable, fixing 533MHz as the optimal frequency for this sub-phase. Thus, each time it re-encounters this sub-phase, it will request a frequency of 533MHz. Note that, in

this example, the lower setting of 320MHz is never exercised, since we expect performance impact to be even larger than at 400 MHz.

3.2 Domain Level Power Management

The domain level power manager is responsible for controlling the frequencies of the tiles and the voltage levels of the domains. It receives requests from the local power managers and it has to decide on how to service these requests. There are two choices to be made when implementing the domain level power management scheme. The first choice is how to implement the communication between the local and the domain level management schemes along with the placement of the domain level power manager(s). The second one is regarding the policy that this domain level scheme would apply to decide on the final voltage and frequency settings to be applied.

3.2.1 Communication Between the Managers on the SCC

Bearing in mind the fast on-chip network and the SCC's limitation of a single outstanding request for voltage change, a natural first choice would be to use the SCC's fast Message Passing Buffers (MPB) for communication and to have a single core acting as a global power manager. In such a scheme, each of the local power managers would write its request to a shared location on the MPBs, different for each core, and the global power manager would poll these locations in a busy loop. If a new request is identified, the global power manager would then try to service that request. This choice has three drawbacks: it (a) reduces the number of usable cores by one, (b) incurs non-negligible power overhead even if running in the lowest performance state (320MHz @ 0.7V) and (c) is not scalable. The non-negligible power overhead comes

```

local_power_manager()
/*called once at each DVFS reference point*/
{
    /*read the timestamp counter*/
    time = read_tsc() - prev_tsc;
    if(opt_freqs[cur_subphase] != 0) {
        /*we have already settled to an optimal setting*/
        new_freq_request = opt_freqs[cur_subphase];
    } else {
        /*we are still trying to settle*/
        if((time/freq_time[cur_subphase][base_freq])
            > 1+δ) {
            /*the increase in execution time was bigger than*/
            /*the threshold so we settle at a freq one step up*/
            opt_freqs[cur_subphase] = cur_freq + step_up;
            new_freq_request = cur_freq + one_step_up;
        } else {
            /*the increase in execution time was tolerable ,*/
            /*try one step down, or settle iff at the lowest*/
            if(cur_freq == lowest) {
                opt_freqs[cur_subphase] = lowest;
                new_freq_request = lowest;
            } else
                new_freq_request = cur_freq + step_down;
        }
    }
    /*the first time we have observed this frequency and*/
    /*we already had an optimal, force re-evaluation*/
    if(freq_time[cur_subphase][cur_freq] == 0 &&
        opt_freqs[cur_subphase] != 0) re_evaluate_opt;
    freq_time[cur_subphase][cur_freq] = time; /*update*/
    /*send our new freq request to the domain manager*/
    sendto();
}

```

(a)

```

domain_power_manager()
{
    init();
    /*main loop*/
    while(true) {
        /*1. stall here until a packet arrives*/
        recvfrom();
        /*compare new request with core's previous one*/
        if(new_freq_request[core] == old_freq_request[core])
            continue;

        /*2. compute the new average for the domain ,*/
        /*quantized to the 4 frequency settings*/
        new_domain_freq = avg(core_freq_requests);
        if(new_domain_freq == old_domain_freq) continue;

        /*3. if required voltage is lower than the current*/
        /*one , apply the frequency change instantly and*/
        /*then lower the voltage*/
        new_domain_voltage = lookup_vf(new_domain_freq);
        if(new_domain_voltage < old_domain_voltage)
            set_domain_freqs(new_domain_freq);

        /*lightweight lock with sleep using the hardware*/
        /*test-and-set registers on the SCC*/
        enter_critical_section();
        /*4. voltage request, one write to a register*/
        do_voltage_change(new_domain_voltage);
        exit_critical_section();

        /*5. it is now safe to increase the frequency*/
        if(new_domain_voltage > old_domain_voltage)
            set_domain_freqs(new_domain_freq);
    }
}

```

(b)

Figure 3: Power management algorithms: (a) Local power management. (b) Domain power management.

from the fact that the power manager needs to constantly poll the MPBs for new requests in a busy while loop.

We opted for a client-server scheme based on the User Datagram Protocol (UDP) for the communication between the local power managers and the domain power manager. The additional latency of less than 0.1ms in the communication of requests is minor compared to the voltage transition overhead that ranges from 1ms to 6ms depending on the number of outstanding voltage change requests. This client-server approach only executes code on a per request basis, thus bringing down the power overhead of the domain level management scheme to practically zero. Since this server can be employed as a daemon service running in the background, it does not require a full core.

To tackle the scalability issue we have decided to employ a hierarchical scheme, where one core in each voltage domain, the domain *leader*, is responsible for requests within that domain. For this hierarchical scheme to work, however, synchronization has to be exercised between the domain *leaders* to make sure that only one voltage change request is outstanding at one point in time. This is efficiently implemented with the hardware test-and-set registers available on the SCC platform.

3.2.2 Power Management Policies

We now look at the policy that our domain power managers (Figure 3b) employ in order to service the frequency requests they receive. Policies investigated are:

- *Simple*: the domain power manager simply services the incoming requests, granting the local power managers the frequency they desire. However, since the frequency can only be changed on a per tile basis, a decision has to be made in case the two cores within a tile request different frequencies. *Simple* chooses the highest of the two frequencies to be applied to the tile. This is a straightforward policy to deal with the restrictions imposed by the DVFS control at coarser granularity than an individual core.
- *Allhigh* runs all the cores within a voltage domain at the highest requested frequency.
- *Alllow* runs all the cores within a voltage domain at the lowest requested frequency.
- *Amean* runs all the cores within a voltage domain at the arithmetic mean of the requested frequencies.
- *Chip-Wide* runs the cores of the *whole* chip at the arithmetic mean of the requested frequencies.

The highest frequency within a voltage domain dictates the voltage at which the whole voltage domain has to operate at, based on the frequency-voltage pairs shown in Table 1. This could lead to cores running at frequencies lower than what the voltage level allows, leading to higher power consumption than necessary. Thus, apart from the *Simple* policy, all the other policies evaluated try to apply a uniform frequency across the cores of each voltage domain, running at the highest frequency the voltage setting can support.

	Size	Instances	Total MPI Calls	Region Coverage
BT.B.36	133	200	26755	99.6%
CG.B.32	789	75	59965	98.4%
EP.B.48	-	-	5	-
FT.B.32	2	19	44	87.5%
IS.C.32	3	10	36	97.8%
LU.B.32	406	250	101538	99.4%
MG.C.32	440	20	9412	85.4%
SP.B.36	103	400	41324	99.6%
lammgs.48	762	95	76340	97.7%
tachyon.48	1	160	162	70.0%

^a The numbers presented in this table represent statistics from MPI rank 0.

Table 2: Phase information for the benchmarks evaluated.

3.3 Discussion

Complexity of Power Management The dynamic table used to store frequency and execution time information has a size proportional to the maximum number of sub-phases within a macro phase, which never exceeded 3K entries for all the evaluated benchmarks and was much smaller for most cases (Table 2). The overhead to take the domain power management decisions (Figure 3b) boils down to issuing a `recv()` call for a UDP socket, reading one received integer, computing the new output function (e.g., for *Amean* four additions and one division), and writing to five global registers (four for the frequency and one for the voltage). This happens only if a packet is received, a poll of which is done by the kernel driver at a kernel quantum (1ms) granularity, and is negligible in a multitasking OS like Linux (Figure 3a). Additionally, as shown in Figure 3a the local power managers only perform a few reads and comparisons and send a udp packet when in steady state, and only do so once every at least 20ms.

Extensions The hierarchical power management scheme we propose does not make use of coordination among the domain power managers. Such additional level in the hierarchical management could be employed, for instance, to handle applications with more irregular and intertwined communication and synchronization patterns or to support management of peak power. For the applications we evaluated and due to our focus on energy optimization we did not find the need for this additional level.

4 ADAPTIVE PHASE PREDICTION

Many highly scalable MPI based scientific applications show recurring communication and execution phases (e.g., a numerical iteration) that occur at an MPI-event granularity. We build a predictor that detects such phases and is able to project future program behavior. This predictor is used as a front-end to our power management scheme, providing

it with the necessary information, as previously discussed in Section 3.

Detailed information about recurring phases in the benchmarks evaluated is shown in Table 2. The *Size* column represents the size of the main phase in number of MPI calls, the *Instances* column shows how many times this phase occurs within the application, the *Total MPI Calls* column depicts the number of all the MPI calls for each application, and the *Region Coverage* column shows the percentage of the application covered by all the recurrences of this main phase. The results underpin how the analysis of MPI call patterns can provide very good coverage for the run-time behavior of these workloads. This motivates our use of phases based on MPI calls for predicting workload behavior.

4.1 Phase Detection

We intercept every MPI call using our own library exploiting PMPI, the profiling interface of MPI. The Program Counter (PC) at the calling site is obtained using the *backtrace* call of GNU `glibc` and is used to uniquely identify the MPI call. We execute code before and after each call that first tries to identify phases and then predicts the next MPI call. We also measure the call’s execution time, as well as the execution time of the program in-between calls.

At the heart of the SuperMaximal Repeat phase Predictor (SMRP) lies a phase detection implementation based on the *supermaximal repeat string* algorithm [9]. This algorithm takes linear time and space with respect to the length of the input sequence. We modified the algorithm to adapt it to the problem at hand. More specifically, in our case a character is a PC and a string is a sequence of PCs. Moreover, we added the following restrictions: i) the supermaximal repeat needs to be non-overlapping, i.e., part of an instance of a supermaximal repeat cannot lie within another instance of it, and ii) it has to be recent, that is, it has to occur until the end of the sequence. A *macro phase* in our terminology is, therefore, a supermaximal repeat in the sequence of PCs that also meets these restrictions.

Initially, the algorithm enters a *record* mode, where it searches for a macro phase in the sequence of PCs. Within each entry associated with a PC, we also store information about the execution time of the MPI call, as well as for the program regions between MPI calls. Once a macro phase is identified, the algorithm enters the *replay* mode, during which the found macro phase is replayed and predictions are enabled. At each MPI call, the current PC is compared against the expected PC based on the macro phase. If they match we stay in *replay* mode, otherwise we go back to *record* mode.

To better understand the phase detection algorithm, let us go through an example. In Fig 4a we see a small portion of the execution trace for the NAS NPB *MG* benchmark. The program starts off at the beginning of the trace, with the detection algorithm in *record* mode. The algorithm starts mon-

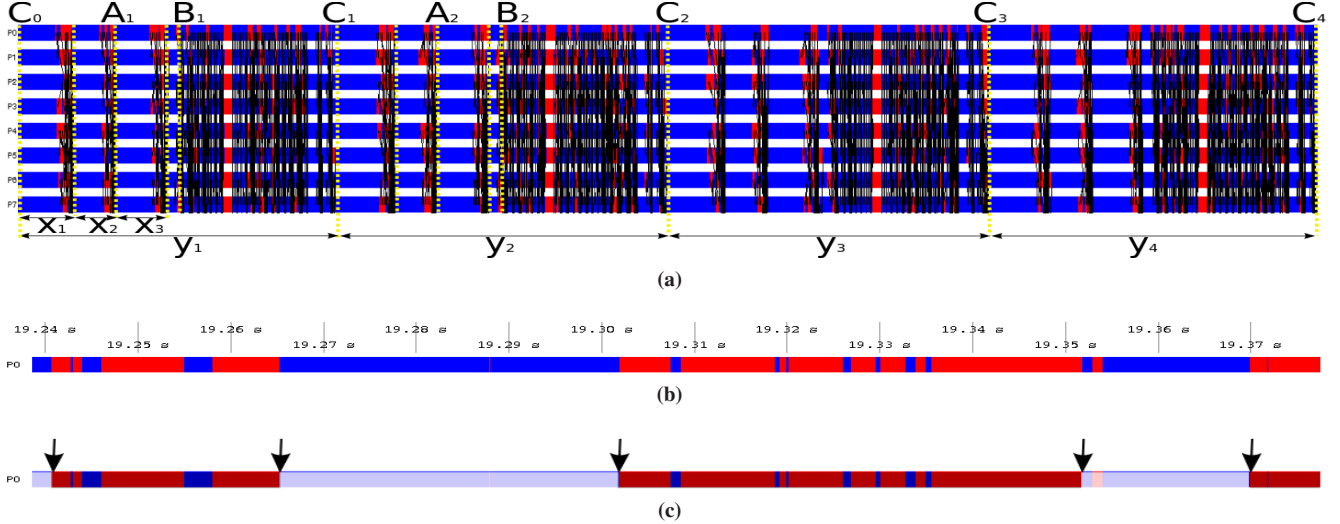


Figure 4: Example of recurring phases in the NAS NPB MG benchmark. The blue regions represent program execution, the red ones show MPI calls, and the black lines represent messages. (a) Sample of the program’s trace obtained with the Intel Trace Analyzer. The trace for only eight of the cores is shown, but all the cores exhibit similar behavior. We can identify the macro phase recurring four times (y_1 to y_4) and some sub-phases they subsume (x_1 to x_3). (b) Zoom inside one of the instances of the recurring macro phase. (c) Partitioning of the macro phase into sub-phases of appropriate size for the power controller to handle. The arrows represent DVFS scheduling decision points.

itoring the sequence of calls and at point A_i is able to make a macro phase detection, because sub-phases x_1 and x_2 have identical sequences of PCs. At this point, therefore, the algorithm transitions to *replay* mode and predictions are enabled. However, at point B_i the algorithm detects that the program no longer follows the detected phase, hence falling back to *record* mode. The same interplay of events is repeated at points A_2 and B_2 , respectively. The algorithm is not able to detect the biggest recurring phase until we reach point C_2 , where the first two instances of the macro phase have occurred, namely regions y_1 and y_2 . From point C_2 onwards, the algorithm enters *replay* mode, and stays in this mode until the end of the program, enabling predictions along the way.

4.2 Phase Predictor

With a mechanism to detect recurring phases that occur at an MPI-event granularity, it is now possible to build a predictor front-end for our power management scheme. As a first step, the predictor has to decide which points within the macro phase are going to act as reference points for DVFS scheduling decisions, respecting the power controller requirements (Section 3.1). To this end, we apply the principle of “reducible regions” [15] to our macro phases to amortize the cost of DVFS switches. If a sub-phase consisting of a single MPI call or a program execution between two MPI calls is longer than the imposed threshold then this choice is trivial. If they are shorter, however, we *merge* together sub-phases of the program that i) are longer than the threshold of 20ms (Section 3.1) and ii) have denser MPI or program sub-phases. Hence, the result is a macro phase partitioned into MPI and program sub-phases. The reference points are

then put around these sub-phases.

At each reference point the predictor provides the controller with the predicted next sub-phase along with its projected execution time (based on its previous occurrence). Using the same example as the one in the previous paragraph, the identified macro phase would be partitioned into sub-phases with DVFS reference points placed as shown in Figures 4b and 4c.

4.3 Complexity of Phase Prediction

The fact that the phase detection algorithm takes linear time and space with respect to the length of the input sequence of PCs translates to negligible overhead at run-time of less than 1% on average. Furthermore, the dynamic structure used to store the predictor information has a size within the same order of magnitude as the size of the maximum occurring pattern (Table 2), and did not exceed 3K entries in our experiments. After the predictor has settled, its overhead boils down to obtaining the PC, doing a comparison with the expected PC and incrementing the index to the next expected PC.

5 EVALUATION METHODOLOGY

Each of the 48 cores of the SCC runs an instance of Linux with a modified 2.6.16 kernel. We use the NAS Parallel Benchmark suite MPI version 2.4, except for the EP benchmark which only has 5 MPI calls clustered at the end of the program, thus not providing enough potential at an MPI-event granularity. Data set Class B is used for benchmarks FT, LU, CG, BT and SP and class C is used for IS and MG. CG, FT, IS, LU, and MG require the number of cores to be

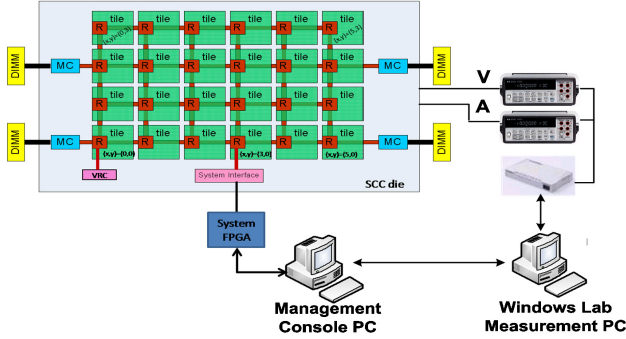


Figure 5: Intel SCC power measurement setup.

a power of two, utilizing 32 cores of the SCC. BT and SP require a square number of cores, utilizing 36 cores. Two benchmarks from the SPEC MPI 2007 v1.1 are also used, namely *lammps* (a complex molecular dynamics simulation code) and *tachyon* (a parallel ray trace application). The other SPEC MPI 2007 benchmarks are written in *Fortran90* and were not supported by our compiler infrastructure. The *mref* data set is used for *lammps* and the *train* data set for *tachyon*. These two additional benchmarks are able to utilize all 48 cores on the SCC. All the applications were built with GCC version 3.4.5 using the *mpich2* MPI library version 1.2.1p1. The object files were linked against our PMPI wrapper library. The results presented are based on the average out of three runs and represent full program execution, including initialization.

To evaluate energy consumption we measure the chip’s supply current. Our experimental setup (Figure 5) consists of: i) SCC board, ii) 2 Agilent 34401A Digital Multimeters and iii) a National Instruments GPIB-ENET 100 for transferring the measurements over the network. One of the multimeters is clipped onto an on-board shunt resistor for the main SCC socket supply voltage to track current. The measurements are transferred through ethernet to a Lab Measurement PC that runs a TCP server application that communicates with the SCC infrastructure and receives commands to initiate and stop monitoring.

6 EXPERIMENTAL RESULTS

In this section we present experimental results for a range of benchmarks and evaluate the different power management policies. In addition, we compare our SMRP predictor presented in Section 4 against a state-of-the-art history predictor, the Global History Table Predictor (GHTP) [13].

6.1 Summary of DVFS Results

We evaluate the proposed two-level, hierarchical power management with our SMRP predictor and the *Amean* policy (SMRP+Amean) against the baseline *Simple* policy and

the GHTP predictor as well as a chip-wide scheme. The chip-wide alternative uses the SMRP and *Amean* policy but applies voltage and frequency pairs to the whole chip. The results are depicted in Figure 6 and show Energy Delay Product (EDP), execution time, and average power consumption. Results are normalized against a no DVFS setting with all the cores running at a constant 800MHz. Our scheme using SMRP with *Amean* on individual domains provides significant EDP improvements of as much as 27.2% for BT, and 11.4% on average, with an increase in execution time of 7.7% on average, and less than 12.0% in the worst-case (for SP), over the no DVFS setting. It also performs favorably against the alternative schemes, outperforming the best one (GHTP+Amean) by 5.4% in EDP and by 11.6% in execution time. Moreover, it performs well within the 3:1 ratio of *power savings* : *performance degradation* that is nominally expected in the field [12].

The source of the improvements is twofold. First, the SMRP predictor provides significant boost in accuracy compared to the GHTP (Section 6.2), allowing the local power managers to perform tighter control by having more accurate timing data and enables power reduction with small performance impact. This applies to CG, LU, and, especially, *lammps*. MG, SP, and BT also enjoy improvement due to this, but to a lesser extent. Second, the *Amean* power management policy performs significantly better energy-wise than the baseline *Simple* policy (see Section 6.3). This applies to all the benchmarks, except for *tachyon* where all the schemes settle at the highest frequency. *tachyon* is a compute-bound application which is highly sensitive to frequency changes – almost linear increase of 70% in execution time by going from 800MHz to 400MHz – making the highest frequency most profitable. The chip-wide scheme performs similarly to our proposal for benchmarks exhibiting more uniform behavior, like FT, IS, BT, and SP. However, for benchmarks that show more unbalanced workloads between the threads chip-wide shows significant increase in execution time. This applies to LU, CG and, to a lesser extent, MG.

6.2 Prediction Evaluation

We now evaluate the SMRP predictor proposed in Section 4, which is used as a front-end to the proposed power management scheme. We compare SMRP against a state-of-the-art Global History Table Predictor (GHTP), based on the one presented in [13]. Such a predictor also tries to detect recurring phases by taking into account a window of last seen sequences. This window was chosen to have a length of 8, as in [13], coupled with a history table of 1024 entries. Figure 7 depicts the results of the prediction evaluation. The SMRP outperforms GHTP by 17% on average with a coverage decrease of only 3%. The coverage decrease for most of the benchmarks is due to the slightly longer training that SMRP requires. For benchmarks IS, FT, and *tachyon* where

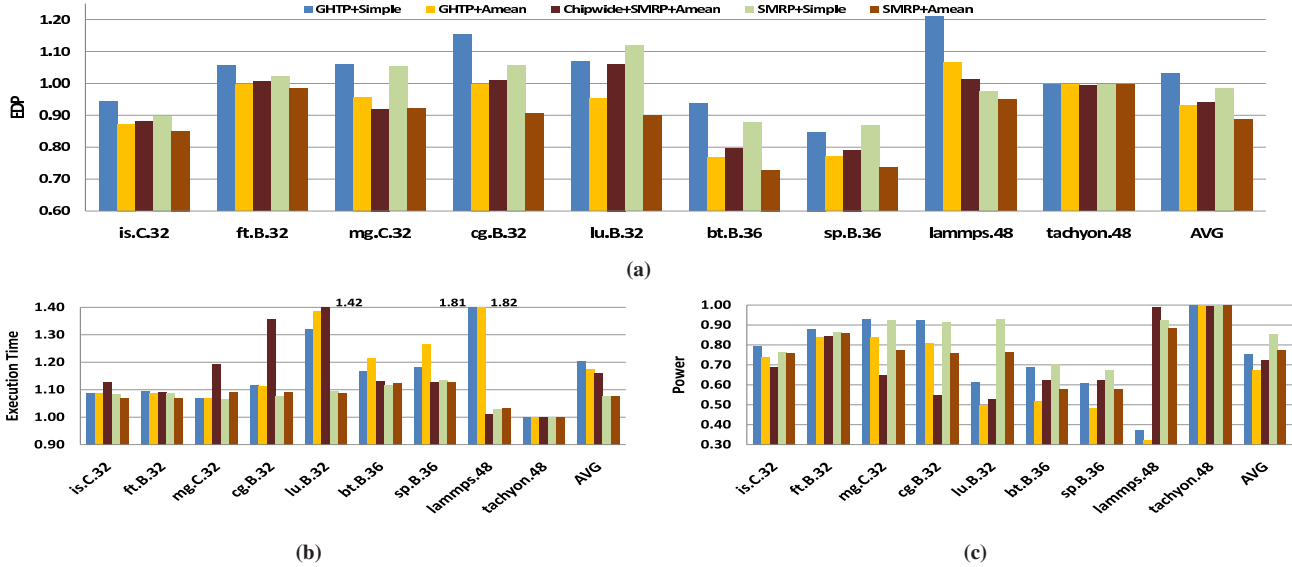


Figure 6: Overall results comparing our overall proposed scheme (SMRP+Amean) against the baseline Simple policy and the GHTP predictor: (a) Energy Delay Product. (b) Execution Time. (c) Power.

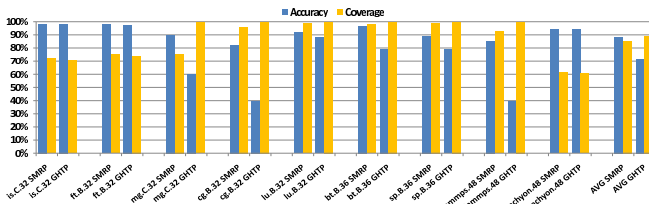


Figure 7: Predictor accuracy and coverage.

the macro phase is trivial (ranging from 1 to a few MPI calls), the two predictors perform identically. In workloads exhibiting more complex macro phases, SMRP significantly outperforms GHTP. This is true for MG, BT and especially true for CG and *lammmps*.

The reason for the significant increase in accuracy of the SMRP over the GHTP is that it i) is able to detect patterns of variable length, and ii) is guaranteed to settle on the longest recurring pattern (the macro phase). These two advantages give SMRP a significant boost in accuracy, especially for benchmarks with non-trivial macro phases like MG, CG and *lammmps*. These complex macro phases are significantly longer running and exhibit smaller patterns within them, tricking the GHTP into false-positives.

6.3 Power Management Policies

We now evaluate in more detail the power management policies employed by the domain managers. We compare them based on their execution time and EDP, and the results are presented in Figure 8. *Amean* is clearly the most beneficial policy in terms of EDP improvements. *Simple* performs well in terms of execution time but fails to provide signifi-

cant energy benefits. *Allow* is the worst performing policy with a substantial increase in execution time, and *Allhigh* is the faster policy but provides small energy improvements.

To better understand the policies and their repercussions in benchmark energy performance, we provide a representative trace from the CG benchmark, showing different voltage and frequency decisions for each policy (Figure 9). We chose CG because of its non-uniform behavior which results in threads requesting different frequencies. If we follow the decisions made by the *Simple* policy we can see that, although it chooses frequencies ranging from 400MHz to 800MHz for the shown tile, the voltage is always at the highest value. This happens because at least one core in the voltage domain has requested the highest frequency setting. Thus, *Simple* fails to provide noticeable reductions in power consumption. *Allhigh* settles for the highest voltage frequency pair for the entire evaluated trace, behaving identically to a static policy. The *Allow* policy settles for the lowest settings of all the policies. Finally, *Amean* settles for settings that lie between the *Allow* and *Allhigh* policies, resulting in a compromise between power consumption and execution time. This explains the behavior of the evaluated policies for the CG benchmark in Figure 8 and can be extrapolated for benchmarks MG and LU.

6.4 Discussion

The hierarchical power management we propose is intended as an approach to provide more effective power management in many-core architectures that present voltage and frequency control domains, as opposed to at each individual core. Similarly, the phase-based application driven policy we propose can be generally applied to parallel applications that

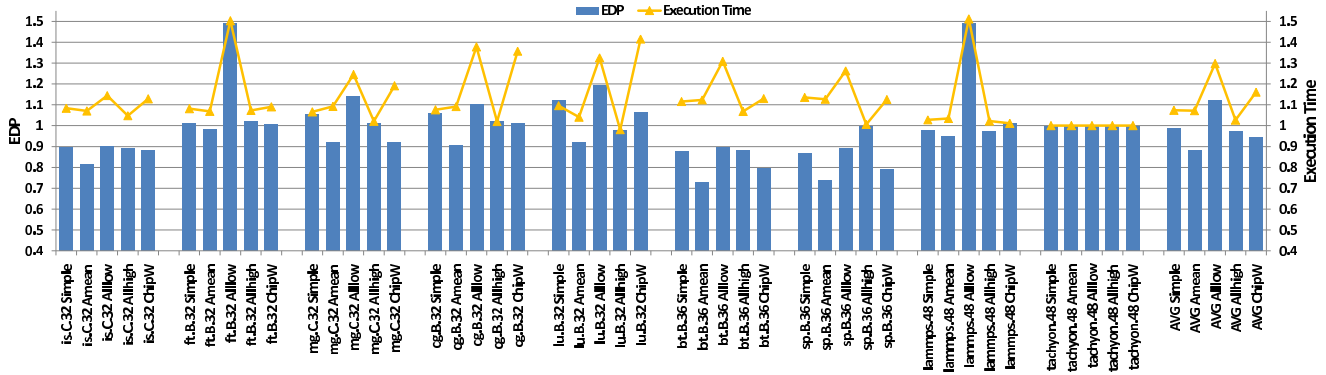


Figure 8: Execution time and total energy consumption for the different power management policies evaluated with the SMRP predictor.

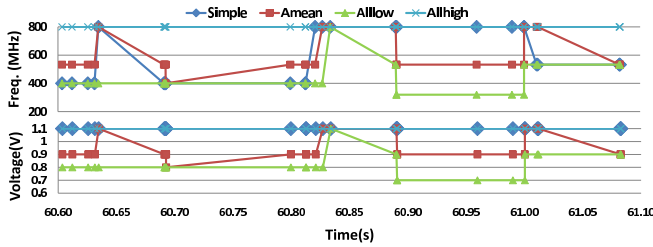


Figure 9: Representative trace from the CG.B.32 benchmark for one Voltage Domain. Frequency changes are shown for one of its tiles.

employ the message passing paradigm. On the other hand, some of the results we present are partially biased to the SCC system parameters and must be considered within this context. We briefly discuss some of these aspects.

Sensitivity to Performance Thresholds and Management Policies Before settling on the performance threshold of 10% presented in Section 3.1.1 we have evaluated different thresholds based on their average EDP and execution time, as depicted in Figure 10. The 10% was a natural choice given our goal to minimize EDP. Note that due to the large gap between the frequency steps of the SCC lower values offer little room for DVFS. Moreover, we have also experimented with additional power management policies, namely one computing the *geometric mean* of the requests and one computing a *weighted mean*, where each requestor sends a weight associated with its request. We have found, however, that these policies performed very close to *Amean*, and hence kept the latter for its simplicity.

Core and Memory Frequencies Compared to existing multi-core systems the SCC cores are relatively simple and run at a relatively low frequency compared to its state-of-the-art DRAM. The implication of this is that, even though the NAS NPB benchmarks often show memory bound phases, they were less so in our experiments. This, in turn, means that we had much less leeway in terms of reducing power consumption without penalizing execution time by reducing

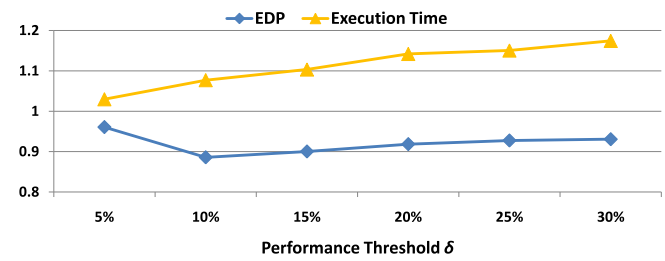


Figure 10: Evaluating different performance thresholds. Results represent the average over all the workloads.

frequency. In addition, since operating frequencies are derived by division from a reference clock, valid operating frequencies are non-uniformly distributed, as listed in Table 1. Overall, this means that the absolute energy savings numbers we report should not be directly compared to those previously reported in the literature, which reflect systems with more powerful cores relative to the memory system. Nevertheless, the relative energy and execution time results between our proposed approach and the alternatives are still indicative of the benefits of our approach.

Chip-wide DVFS Our results in Section 6.2 suggest that applying DVFS at the granularity of the entire chip leads to lower energy-efficiency than finer-grained approaches. In particular, while this approach leads to reasonable performance and energy consumption for balanced parallel applications (and likely multi-programmed workloads alike), the execution time of more imbalanced applications is significantly increased. In addition to being less able to cope with imbalanced applications, chip-wide DVFS is also penalized in the SCC because of its restriction of a single in-flight voltage change (which translates to a 6ms overhead for a whole-chip change).

Hardware Control Support On the SCC experimental processor all DVFS knobs are exposed to the software stack to allow experimenting with power management policies. On commercial processors, available CPU and system-wide

power states are abstracted by the ACPI open standard. Power states selected by software levels can be seen as guidance for the underlying hardware to select suitable parameters and control steps. With hardware support, even quicker response times and additional safety features can be achieved. Our distributed control policy could be implemented with hardware support in order to protect the controller and data structures (e.g., the operating tables for sub-phases) and enable synergies with non-visible resource management and protection policies, such as thermal protection by power capping.

7 RELATED WORK

Power management techniques to reduce energy consumption have been extensively studied in prior work. Our focus lies on application-driven power management for scientific MPI workloads, where DVFS can be employed to exploit memory and communication bound phases during active periods. These workloads are continuously active. Our approach is complementary to idle time policies often implemented at the OS level which switch cores to sleep states when idle periods are detected, such as PowerNap [18].

Isci et al. [12] evaluated several DVFS policies on multi-cores within a power constraint, also evaluating a simple chip-wide policy. They employ a hardware/software hierarchical power management scheme and their evaluation is simulation based. Our system is fully implemented in software and run on a real system. Wu et al. [26] proposed a dynamic compilation framework to detect memory-bound phases of sequential applications and exploit them for energy savings. We also exploit phases, but we target a many-core system and we do not require profile data. Huang et al. [11] also exploit repeatable program (subroutines) behavior employing a power management controller whose policy resembles the policy of our local controller. However, their scheme is purely hardware based, targets a single core system, and is evaluated on a simulator. A run-time system monitoring and prediction of program phases is used to guide power management in [13], in an approach similar to ours. Unlike our approach, however, they target a single core system, have full DVFS control over the core and exercise heavy performance counter usage³. PC-based pattern prediction for dynamic power management has been previously exercised by Gniady et al. [8] to detect and exploit I/O inactivity, as opposed to our full-chip, real-system DVFS.

DVFS management of MPI applications has been previously studied in [7, 15, 21, 22, 25]. Most of this work uses MPI events as the basic block of their power management proposals, similarly to our work. In [7], Freeh et al. attempt to

minimize energy consumption by exploiting memory boundness of phases in applications, and in [15] a similar energy goal is met by exploiting communication boundedness. Minimal execution time meeting power constraints is the goal of [25], where they develop an analytical model to this end. In [21], Rountree et al. use a linear programming model to determine upper bounds in energy savings by analyzing communication patterns. They use the same communication pattern detection in [22] to build a run-time system aiming at energy savings with minimal performance penalty, similar to our work. Our work differentiates itself from this prior work in: i) not assuming independent per-core power management, and i) not requiring offline profile data, unlike [7, 21, 25]. Moreover, we target SCC, a many-core system with all the communication happening on-chip, unlike the regular MPI clusters where the communication is off-loaded to a Network Interface Card (NIC). This imposes additional load on the SCC, limiting the idle time due to communication, thus making it harder for a power management scheme to achieve energy savings with low performance penalty.

8 CONCLUSIONS

Future many-core systems may provide DVFS control on a frequency and voltage domain granularity on-die, such as what we currently find in multi-chip multi-core architectures and clustered organizations. Intel Labs' Single-chip Cloud Computer (SCC) experimental processor provides an accessible infrastructure for exploring DVFS methods on different DVFS domain granularity.

To tackle the DVFS requirements of cores organized in DVFS domains, we have presented a novel, hierarchical, and transparent run-time client-server power management scheme that is applicable to many-core architectures. It exploits a new phase detector for MPI applications, and tries to find the "best" frequency-voltage pair for each repeatable phase, in order to minimize energy consumption within a performance window. Our phase detection and DVFS control techniques were implemented and applied on the SCC platform, a real system, running MPI applications resulting in significant EDP improvements of as much as 27.2%, and 11.4% on average, with an increase in execution time of 7.7% on average.

The predictor component plugged into the local controller is based on a novel phase detection algorithm that is able to dynamically detect recurring phases that occur at an MPI-event granularity and provide predictions for future program behavior. Our predictor provides a significant increase in accuracy of 17% over a state-of-the-art phase predictor.

Our work represents a first attempt in designing and implementing a run-time power management system for many-core systems. As such, it provides a reference point to future application-driven designs of DVFS management schemes on many-cores.

³Performance counter extraction from user space for all but the timestamp counter incurs non-negligible overhead in the current SCC software release.

REFERENCES

- [1] M. Annavaram, E. Grochowski, and J. Shen. "Mitigating Amdahl's Law through EPI Throttling". *Intl. Symp. on Computer Architecture (ISCA)*, June 2005.
- [2] M. Annavaram, R. Rakvic, M. Polito, J.-Y. Bouguet, R. Hankins, and B. Davies. "The Fuzzy Correlation Between Code and Performance Predictability". *Intl. Symp. on Microarchitecture (MICRO)*, December 2004.
- [3] A. Bhattacharjee and M. Martonosi. "Thread Criticality Predictors for Dynamic Performance, Power, and Resource Management in Chip Multiprocessors". *Intl. Symp. on Computer Architecture (ISCA)*, June 2009.
- [4] Q. Cai, J. González, R. Rakvic, G. Magklis, P. Chaparro, and A. González. "Meeting Points: Using Thread Criticality to Adapt Multicore Hardware to Parallel Regions". *Intl. Conf. on Parallel Architectures and Compilation Techniques (PACT)*, October 2008.
- [5] A. S. Dhodapkar and J. E. Smith. "Managing Multi-Configuration Hardware Via Dynamic Working Set Analysis". *Intl. Symp. on Computer Architecture (ISCA)*, June 2002.
- [6] E. Duesterwald, C. Cascaval, and S. Dwarkadas. "Characterizing and Predicting Program Behavior and its Variability". *Intl. Conf. on Parallel Architectures and Compilation Techniques (PACT)*, September 2003.
- [7] V. W. Freeh and D. K. Lowenthal. "Using Multiple Energy Gears in MPI Programs on a Power-Scalable Cluster". *Symp. on Principles and Practice of Parallel Programming (PPoPP)*, June 2005.
- [8] C. Gniady, Y. C. Hu, and Y.-H. Lu. "Program Counter Based Techniques for Dynamic Power Management". *Intl. Symp. on High-Performance Computer Architecture (HPCA)*, February 2004.
- [9] D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, New York, NY, USA, 1997.
- [10] J. Howard, S. Dighe, et al. "A 48-Core IA-32 Message-Passing Processor with DVFS in 45nm CMOS". *Intl. Solid-State Circuits Conf. (ISSCC)*, February 2010.
- [11] M. C. Huang, J. Renau, and J. Torrellas. "Positional Adaptation of Processors: Application to Energy Reduction". *Intl. Symp. on Computer Architecture (ISCA)*, June 2003.
- [12] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, and M. Martonosi. "An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget". *Intl. Symp. on Microarchitecture (MICRO)*, December 2006.
- [13] C. Isci, G. Contreras, and M. Martonosi. "Live, Runtime Phase Monitoring and Prediction on Real Systems with Application to Dynamic Power Management". *Intl. Symp. on Microarchitecture (MICRO)*, December 2006.
- [14] W. Kim, M. Gupta, G.-Y. Wei, and D. Brooks. "System Level Analysis of Fast, Per-Core DVFS Using On-Chip Switching Regulators". *Intl. Symp. on High-Performance Computer Architecture (HPCA)*, February 2008.
- [15] M. Y. Lim, V. W. Freeh, and D. K. Lowenthal. "Adaptive, Transparent Frequency and Voltage Scaling of Communication Phases in MPI Programs". *Supercomputing Conf. (SC)*, December 2006.
- [16] P. Macken, M. Degrauwe, M. Van Paemel, and H. Oguey. "A voltage Reduction Technique for Digital Systems". *Intl. Solid-State Circuits Conf. (ISSCC)*, February 1990.
- [17] G. Magklis, M. L. Scott, G. Semeraro, D. H. Albonesei, and S. Dropsho. "Profile-Based Dynamic Voltage and Frequency Scaling for a Multiple Clock Domain Microprocessor". *Intl. Symp. on Computer Architecture (ISCA)*, June 2003.
- [18] D. Meisner, B. T. Gold, and T. F. Wenisch. "PowerNap: Eliminating Server Idle Power". *Intl. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, March 2009.
- [19] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*, 2.2 edition, 2009.
- [20] E. Rotem, A. Mendelson, R. Ginosar, and U. Weiser. "Multiple Clock and Voltage Domains for Chip Multi Processors". *Intl. Symp. on Microarchitecture (MICRO)*, December 2009.
- [21] B. Rountree, D. K. Lowenthal, et al. "Bounding Energy Consumption in Large-Scale MPI Programs". *Supercomputing Conf. (SC)*, December 2007.
- [22] B. Rountree, D. K. Lowenthal, B. R. de Supinski, M. Schulz, V. W. Freeh, and T. Bletsch. "Adagio: Making DVS Practical for Complex HPC Applications". *Intl. Conf. on Supercomputing (ICS)*, June 2009.
- [23] G. Semeraro, G. Magklis, R. Balasubramonian, D. H. Albonesei, S. Dwarkadas, and M. L. Scott. "Energy-Efficient Processor Design Using Multiple Clock Domains with Dynamic Voltage and Frequency Scaling". *Intl. Symp. on High-Performance Computer Architecture (HPCA)*, February 2002.
- [24] T. Sherwood, S. Sair, and B. Calder. "Phase Tracking and Prediction". *Intl. Symp. on Computer Architecture (ISCA)*, June 2003.
- [25] R. Springer, D. K. Lowenthal, B. Rountree, and V. W. Freeh. "Minimizing Execution Time in MPI Programs on an Energy-Constrained, Power-Scalable Cluster". *Symp. on Principles and Practice of Parallel Programming (PPoPP)*, March 2006.
- [26] Q. Wu, M. Martonosi, D. W. Clark, V. J. Reddi, D. Connors, Y. Wu, J. Lee, and D. Brooks. "A Dynamic Compilation Framework for Controlling Microprocessor Energy and Performance". *Intl. Symp. on Microarchitecture (MICRO)*, December 2005.