REPP-H: Runtime Estimation of Power and Performance on Heterogeneous Data Centers

Rajiv Nishtala and Xavier Martorell
Universitat Politécnica de Catalunya &
Barcelona Supercomputing Center, Spain
firstname.lastname@bsc.es

Vinicius Petrucci
Universidade Federal da Bahia, Brazil
petrucci@dcc.ufba.br

Daniel Mossé
University of Pittsburgh, USA
mosse@cs.pitt.edu

***Abstract*—One of the main challenges in data center systems is operating under certain Quality of Service (QoS) while minimizing power consumption. Increasingly, data centers are adopting heterogeneous server architectures with different power-performance trade-offs. This requires careful understanding of the application behavior across multiple architectures at runtime so as to enable meeting specified power and performance requirements. In this work, we present and evaluate REPP-H (Runtime Estimation of Performance and Power on Heterogeneous data centers). REPP-H leverages hardware performance counters available on all major server architectures to ensure a highly responsive power capping mechanism and delivering a minimum performance in a single step. We experimentally show that REPP-H can successfully estimate power and performance of several single-threaded and multiprogrammed workloads. The average errors on ARM, AMD and Intel architectures are, respectively, 7.1%, 9.0%, 7.1% when predicting performance, and 6.0%, 6.5%, 8.1% when predicting power on those heterogeneous servers.**

I. Introduction

Modern data centers increasingly demand improved performance with minimal power consumption. Managing the power and performance requirements of the applications is challenging because these data centers, incidentally or intentionally, have to deal with server architecture heterogeneity [19], [22]. One critical challenge that data centers have to face is how to manage system power and performance given the different application behavior across multiple different architectures.

Dynamic voltage and frequency scaling (DVFS) is a widely used technique to optimize constraints such as performance and power trade-offs [27], power capping [13], [23] and minimum performance [21]. To satisfy these *constraints*, iterative algorithms periodically monitor system behavior and change the frequency and voltage of cores.

Iterative algorithms are P-State based (i.e, DVFS): they monitor the application behavior history [11] and set the DVFS configuration for the next quantum. However, these approaches require multiple iterations before power-performance criteria are satisfied and can lead to massive violations in power-performance budgets for data centers [19], [27].

Fast and reactive prediction based algorithms provide the benefit of quick response and reaction for a small fraction of computation costs. The broad spectrum of computation and memory behavior can be understood by monitoring the hardware performance monitoring counters (PMCs), available on most commercial processors. For instance, applications that are memory bounded, do not take benefit of higher P-State as they stall for memory, thus using a lower P-State might be sufficient to satisfy the performance constraint while saving power. At the same time, the converse is true for compute bounded workloads. Using PMCs, we demonstrate that our prediction algorithm can determine if the application is scalable with DVFS at runtime and can select the most appropriate configuration given a constraint. Such techniques are also important to enable energy-efficient cluster management in data centers (which thread is assigned to each core and what frequency to set each core when).

Most prior works have focused on building specialized performance and power models for individual server architectures [21], [23], [24], [30]. In this paper, we extend prior works to scale power and performance prediction techniques for a variety of server architectures. We specifically designed a Runtime Estimation of Performance and Power across Heterogeneous architectures (REPP-H) that uses two hardware knobs, namely: processor performance states (P-States) and idle cycles (Cl-States), to estimate and control power-performance on server architectures running multiple workloads using statistics gathered on real processors. Cl-States introduce synchronized idle injections across CPU threads to provide forced and controllable C-State residency. The changes in Cl-States are governed using Intel powerclamp driver [5].

We show that REPP-H scales well (requiring under 100 cycles per prediction) and is effective on multicore architectures that require meeting service level agreements. REPP-H uses P-States, Cl-States to understand the complicated mapping between the low-level power management features and constraints like minimum performance (Millions of Instructions per Second (MIPS)) and power (in Watt).

Our major contributions are below:

- We extend the models built in [23] from single-threaded workloads to multiprogrammed workloads to predict performance and power on three different architectures.
- We evaluated REPP-H on each architecture for single-threaded and multiprogrammed applications with mulitple constraints and show that power capping and minimum performance requirements are met.
- We show that the power capping mechanism built with REPP-H can perform much faster (3.6X) than a commonly used iterative approach.

After extensive experiments on real systems to understand the architectural influence on prediction techniques, our results show that REPP-H performs well across architectures with

TABLE I: Events and *perf* raw event numbers used on AMD, ARM and Intel machines

Component	AMD	AMD (*perf* raw event)	ARM	ARM (*perf* raw event)	Intel	Intel (*perf* raw event)
FE	retired_uops	r5000c1:u	-	-	uops_retired:any	r1c2
INT	dispatch_stall_for_int_sched_queue_full	r0d6	inst_spec_exec_simd	r074	(uops_dispatched_port:(port_0 + port_1 + port_5) - fp_comp_ops_exe:x87 - br_inst_retired)	r1a1 + r2a1 + r80a1 - r110 - r0c4
FP	retired_mmx_and_fp_instructions:x87	r5001cb:u	inst_spec_exec_vfp	r075	fp_comp_ops_exe:x87	r110
BPU	branch_retired	r5000c3:u	inst_spec_exec_soft_pc	branches	br_inst_executed	rff88
L1	perf_count_hw_cache_l1d	-	l1d_cache	r04	perf_count_hw_cache_l1d	-
L2	requests_to_l2	r037d	l2d_cache	r16	l2_rqsts:0xc0	rc024
L3	perf_count_hw_cache_references	r080	no L3	no L3	last_level_cache_references	r4f2e
MEM	perf_count_hw_bus_cycles	r0062	perf_count_hw_bus_cycles	bus-cycles	perf_count_hw_bus_cycles	bus-cycles

TABLE II: Machines used in this study.

Processor	Linux Kernel	Power Meter	gcc	DFS (in GHz)	L2 (size in KB)	L3 (size in MB)
Intel Core i7-2760QM (4 cores)	3.14.5, perf events	RAPL	4.8.1	0.8-2.4	256	6
AMD Phenom II X4 B97 (4 cores)	3.13.0, perf events	WattsUp Pro	4.9.2	0.8-3.2	512	6
ARM Juno 64bit (ARMv8 – 2 Cortex A57 and 4 Cortex A53)	4.3.0, perf events	Native energy meter	4.9.2	0.6-1.15	2048 (Cortex A57)	no L3

average power prediction errors of 6.0% on ARM, 6.5% on AMD, and 8.1% on Intel, and, similarly, an average error of 7.1% on ARM, 9.0% on AMD and 7.1% on Intel when predicting performance.

II. REPP-H

Our previous work, REPP [23], is a modeling and prediction technique on server multicores that run single-threaded workloads. REPP predicts performance and power for single-threaded workloads for all combinations of P-States and Cl-States, with high accuracy on an Intel architecture.

REPP is divided into two phases: offline and online. In the **offline phase**, single core models are built by extensive profiling of three floating point benchmarks and three integer benchmarks at random [23] (training dataset) from representative benchmark suites such as SPECcpu2006 [17], PARSEC [10], etc., at various P-States and Cl-States, to build multilinear regression models; one multilinear regression model per P-State for each architecture.

In the offline phase, we profile the activity in the microarchitectural components, namely: Front End (FE); Integer units (INT); Floating Point units (FP); branch predictor unit (BPU); L1 (L1 cache); L2 (L2 cache); LLC (L3 cache); and memory subsystem (MEM). Since these components do not have PMCs to directly record their activity, the **activity ratio** (AR) of these individual components is computed using PMCs activity formulas. The activity ratio is defined as the component's average number of uops per cycle (uops/cycle). Table I summarizes the microarchitectural components, activity formulas for AMD, ARM and Intel, and the raw perf event registers [1]–[3]. To compute the activity ratio, the activity in each component, for each architecture, is divided by `cpu_clk_unhalted (r076)`, `cpu_cycles (cycles)` and `cpu_clk_unhalted:0x01 (r13c)`, respectively.

The offline design phase is attractive because (a) it eliminates the overhead of learning and tuning the performance and power model at numerous P-States and Cl-States; (b) it does not rely on using power sensors/meters to estimate power and performance at runtime; and (c) it is a one-time effort.

In the **online phase**, we showed the prediction accuracy of REPP by evaluating the power and performance models for all combinations of P-States and Cl-States.

REPP-H: Dealing with Multiple Architectures — REPP-H, in contrast to REPP, is a modeling and prediction technique for both single-threaded and multiprogrammed workloads in data centers. The single-core models for the prediction technique are built such that the co-efficients are not dependent on the number of applications. Instead they are based on the activity in each of the microarchitectural components. Therefore, it can scale across multi-node, multi-core data centers and is aimed to ensure power capping or minimum performance requirements. In contrast to prior work [34], REPP-H is not based on application signature or similarities between application. Instead, REPP-H leverages online monitoring of basic PMCs that provide application behavior at runtime, which require a one-time, offline profiling effort per architecture,and not per application. Such data is then fed into statistical tools to predict performance and power, making REPP-H fast, accurate, and architecture-agnostic.

The **offline phase** in REPP-H is imported from REPP. We build multilinear regression models for each architecture at each P-State and Cl-State. In building the multilinear regression models, we specifically profile the activity in the aforementioned microarchitectural components because our results using microbenchmarks on each architecture have shown that these microarchitectural components have a high dynamic power. We define dynamic power as the difference between the current power consumption and power consumption when idle. The activity formulas were built using carefully selected PMCs that are highly correlated with the dynamic power and performance. For instance, on the Intel platforms' pipeline functionality, the scheduler which is a part of the out-of-order engine has six issue ports, out of which ports 0, 1 and 5 are shared for integer, branch instructions and floating point instructions. However, there are counters for each port, branch instructions and floating point instructions. Therefore, we subtract the instructions issued using ports 0, 1 and 5 and the branch instructions and number of floating point instructions

to get the total number of integer instructions. By contrast, on AMD and ARM, the microarchitectural components have unique counters for total number of integer instructions.

The **online phase** is divided into two stages: (1) Similar to REPP, we first validate the models that predict performance and power for each architecture, across a combination of P-States and Cl-States. (2) In REPP-H, the results from each of the single cores models are `aggregated` to estimate the system performance and power consumption in multi-core architectures. Then, we evaluate REPP-H across a wide range of performance and power constraints. Contrary to prior works [13], [21], [27], [28], [30], which predict power and performance at a system level, our work actually predicts at a system level on a per core basis.

Workload Measurement — We perform workload characterization by executing a combination of compute-bound and memory-bound applications. We ran the workloads on three 64-bit architectures, Intel, AMD and ARM, as shown in Table II. We note that the ARM Juno architecture does not allow PMCs to be read for Cortex A53, the in-order processor, as the `CPTR_EL3` (Architectural Feature Trap Register) is not implemented in the Juno chip. Table II shows the architecture, processor, Linux kernel version, the power meters, the compiler (gcc) version used for compiling the benchmarks, the range of core frequencies (min-max) when using DVFS, the L2 and L3 cache sizes (ARM does not have L3). We ran the experiments in multiple institutions, and had no control over the kernel and gcc versions; regardless, the results for each architecture are consistent within that set of experiments.

We use `perf` [6], a performance monitoring tool, on all architectures to collect the `perf_events`. We initialize one session of perf per thread to characterize the thread behavior and collect PMCs every 250ms (control/mapping interval). We have explored mapping intervals of 50ms, 100ms, and upto 950ms, and 1000ms, but 250ms was chosen empirically as most SPEC benchmarks have less than 1% change in performance or power at a particular P-State and allowed us to predict in the same phase [23]. In practice, since the performance and power of the applications varies with time, larger periods of prediction *may* violate the power or performance target, whereas frequent predictions *may* increase the overhead, while delivering minor improvements in predictions. The number of Cl-States are 50, and they can be controlled independently for each core on Intel. ARM and AMD do not have Cl-States. We do not enable any other internal thermal/power management algorithms. Because there is no hyper-threading on AMD and Intel, we disable it only on Intel.

Power Measurement — To gather the power measurements, REPP-H uses the machine's power sensor (in case of Intel and ARM) or an external power meter (in case of AMD power sensors are not available) periodically during execution. To minimize interference, the process collects power data every 250ms. For building the models offline, we measure the power of the system in idle state, where no user initiated tasks are running. This allows us to model the power of the application and in turn the total dynamic system power (subtracting the idle power) for a given frequency.

The power meters used are as follows. First, the *RAPL* (running average power limit) register in the Intel architecture records power of core, uncore, and DRAM controller [24]. We collect idle power only at 2.4GHz because it implements C-States, which allow for sleep/deep-sleep state of the processor to save power when idle, thereby having the same power consumption at all the P-States when *idle*. Second, *WattsUP pro* for AMD: external device that records power of the entire system [4]. We collect idle power at each frequency available since AMD does not implement C-States. Third, *four native energy meters* for ARM [2]: records the power of the GPU, the Cortex A57, Cortex A53, and the DRAM. We collect idle power at 1.15GHz because ARM implements CPUidleness (comparable to C-States on Intel) [26].

REPP-H configuration selector — Modern data centers have power constraints (e.g., power capping) and run multiple application instances with different performance constraints. This requires an algorithm to select a configuration per core such that the performance and power constraints are met per core and/or per application. To ensure that these constraints are met, REPP-H dynamically selects a configuration per core every 250ms by performing a linear search for the P-State that is the nearest to the given constraint; next, REPP-H selects the Cl-State for the given P-State that satisfies the constraint. This selected configuration, P-State, Cl-State, is used to ensure that the constraint is met for each interval. This process is repeated across all cores at the same time.

In our study, the performance and power constraints are given at a system level. These constraints are distributed homogeneously across all cores. For example, if an AMD server can only consume 600W, that power constraint is distributed 25% per core, allowing each core to consume 150W (it would be 50% on ARM). At runtime, for the spawned applications, REPP-H samples application behaviour periodically and uses the models built to predict performance and power at all configurations. REPP-H then selects the configuration for each interval to satisfy the local constraint.

This ability to satisfy constraints per core makes REPP-H better than previous works [13], [21], [27], [28], [30], allowing for multi-node, multi-core data centers running numerous application to satisfy a wide-range of performance and power requirements.

III. Evaluation

We evaluate REPP-H by considering a wide range of single threaded and multiprogrammed workloads. For single-threaded workloads, we ran 22 SPECcpu2006 [17], 9 PARSEC 3.0 [10], 11 SPLASH2x [31] and 6 NAS benchmarks [7]. The number of benchmarks in a multiprogrammed workload is equal to the number of cores: 35 workloads of 4 benchmarks on AMD and Intel, and 10 workloads of 2 benchmarks on ARM, based on the methodology described by Sanchez and Kozyrakis [25]. We could not compile SPECcpu2006 on ARM. For all workloads, we select the native input set size to make it realistic.

The benchmarks are divided in four categories [25]: Insensitive (**N**), Cache-friendly (**F**), Cache-fitting (**T**), and Thrashing/Streaming (**S**). Since each architecture has different cache sizes and frequency ranges, benchmarks in

TABLE III: Categorization of workloads.

Label	Description	Intel-Benchmarks	AMD-Benchmarks	ARM-Benchmarks
N	Insensitive	bzip2, blackscholes, bodytrack, calculix dedup, ep.C, freqmine, gobmk gromacs, h264ref, hmmer, is.C lu_cb, namd, omnetpp, povray raytrace, tonto, vips	blackscholes, bzip2, calculix, ep.C, facesim freqmine, gobmk, gromacs, h264ref hmmer, is.C, lu_cb, namd, omnetpp povray, radiosity, raytrace, raytrace, sjeng tonto, volrend, water_nsquared, water_spatial	blackscholes, ep.C, fft fluidanimate, freqmine is.C, lu_cb, lu_ncb
F	Cache Friendly	cactusADM, cg.C, lbm, libquantum lu_ncb, ocean_cp, sjeng water_spatial, x264, zeusmp	bodytrack, cactusADM, canneal, cg.C fmm, lu_ncb, ocean_cp, streamcluster vips, wrf, x264, zeusmp	facesim, radiosity streamcluster, volrend water_nsquared
T	Cache Fitting	astar, bwaves, canneal, gemsFDTD radix, soplex	astar, bwaves, dedup, radix soplex	canneal, radix, raytrace, vips water_spatial
S	Thrashing	lu.C, mcf, milc, mg.C, sp.C streamcluster, xalancbmk	gemsFDTD, lbm, libquantum, lu.C mcf, mg.C, milc, sp.C, xalancbmk	bodytrack, cg.C, dedup, fmm lu.C, mg.C, ocrean_cp, sp.C

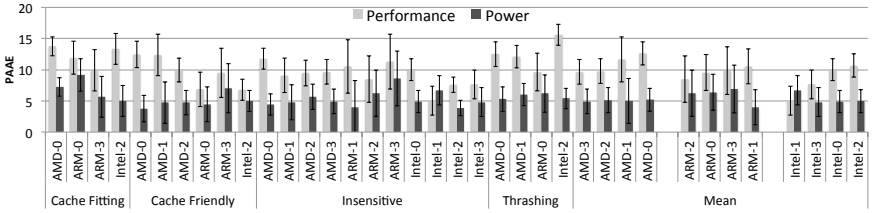


Fig. 1: Average PAAE when predicting performance and power on a single core for across a combination of P-, Cl-States and architectures. The error bars represent the STDEV

.

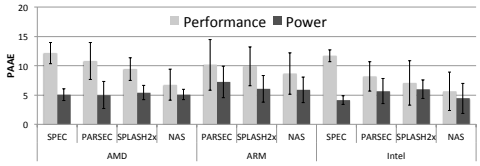


Fig. 2: Average PAAE when predicting performance and power per suite across architectures. The error bars represent the STDEV.

each category are different for each architecture. Table III shows the categorization of the workloads. This method of categorizing multiprogrammed workloads provides significant variability in the size of the shared memory footprint, working set sizes and number of stall cycles.

A. Experiments

We perform two types of experiments: one for validating the power capping mechanism and the other for delivering a minimum performance. We define two input parameters: (a) frequency of change, and (b) x, which represents load or power. The average load offered by the applications is constant between two load changes, which can occur every **load change interval** (1, 6, or 9 seconds), based on a *change_factor*, as follows. Load starts at a minimum, and varies by multiplying load by **change_factor** P until it reaches a maximum load x_{max}; thereafter, the load is multiplied by the negative value of change_factor until it reaches the minimum x_{min}. The values of change_factor tested were 20% (**Low**), 35% (**Mid**), and 50% (**High**). The minimum load is defined as the sum of smallest IPS (instructions per second) for all 4 applications running at minimum frequency; similarly, maximum load is the sum of highest IPS for all 4 applications at maximum frequency. In another set of experiments, we change the power consumed by the workload similar to the load offered by the workload. Mathematically, the experiment conducted for a load_change interval 1 can be represented as follows: In Equation 1, ρ represents the number of datapoints before the maximum load/power (x_{max}) and in Equation 2, $x(n)$ shows the datapoint n in the sequence.

$$\rho = \left\lfloor \frac{\log(x_{\text{max}}/x_{\text{min}})}{\log(1+P)} \right\rfloor \qquad Eq(1)$$

$$x(n) = \begin{cases} x_{\text{min}} \times (1+P)^n & 0 \leq n \leq \rho \\ x_{\text{min}} \times (1+P)^{2\rho - n} & \rho < n \leq 2\rho \end{cases} \quad Eq(2)$$

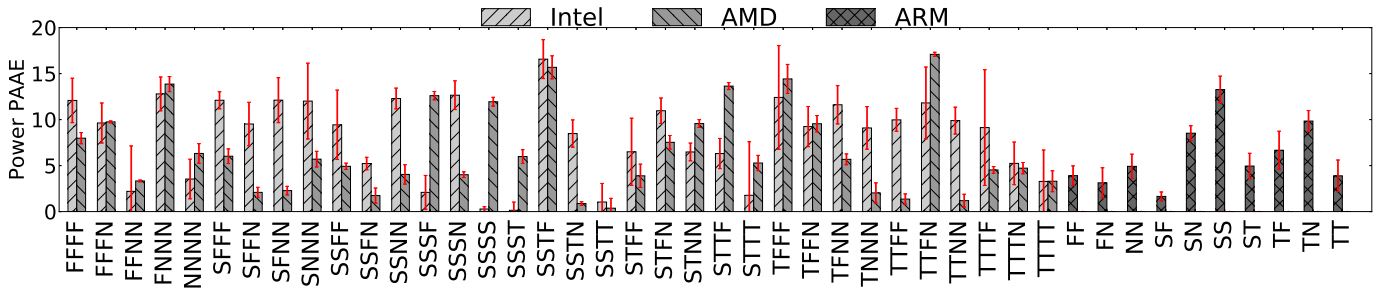
The error occurs when REPP-H selects a configuration that makes the application fall short of the minimum required performance (or exceed the maximum power requirement) in a given mapping interval.

We ran ten experiments for power and ten for performance. Nine of them come from the combinations of the 2 parameters (frequency of change and load/power) described above; the tenth comes from a **Random** setting within fixed boundaries of either power or performance. Selecting a broad spectrum of load (or power) and frequency of change allowed us to validate REPP-H across multiple combination of configurations at runtime. Each experiment was run three times for each power and performance constraint. The deviation over multiple runs for each workload was low (<2%).

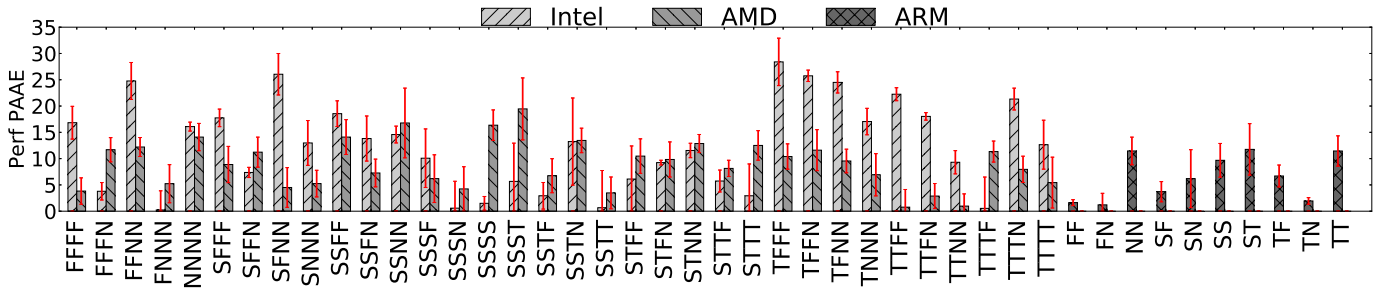
B. Quantifying error in meeting constraints using REPP-H

We present the results of REPP-H when predicting performance and power on single and multicore processors. The prediction error is computed over a period of 400 seconds for all P-States, and Cl-States in terms of percentage absolute average error (PAAE) and the standard deviation (STDEV) [9], [30]. The STDEV over PAAE determines how far the prediction is from the constraint (optimal point). The error metric indicates that the constraint was violated by that amount, which occurs when REPP-H selects a configuration that makes the application fall short of the minimum required performance (or exceed the maximum power requirement) in a given mapping interval.

Single Core Evaluation — To evaluate the results on a single core model, we separate the benchmarks used in validation into four `clusters`, to present the results for



(a) Power



(b) Performance

Fig. 3: Average PAAE when predicting power (3a) and performance (3b) for all workloads under Low, Mid, High and Randomized change_factors in multicore architectures. The error bars represent STDEV across change_factors. The x-axis shows multiprogrammed workloads generated based on methodology described in Section III.

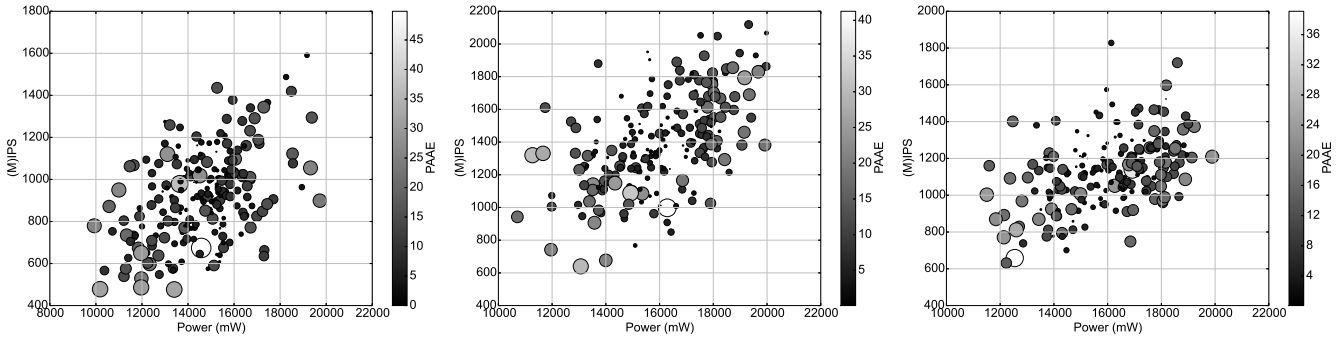
over 150 applications, using K-Means [33] with parameters FE, INT, FP, MEM, BPU, L1, L2, L3. The number of clusters (four) was chosen empirically based on the silhouette coefficient. We narrow the number of parameters to two using principal component analysis for keeping the most singular vectors to project the data in a lower dimensional space. Clusters are named with the architecture and cluster number, such as ARM-0, Intel-3, etc. Each cluster has results from all four categories. The PAAE on a single core is computed using the error between value measured from PMCs performance (and power meters for power) and predicted performance or power (using REPP-H).

Figure 1 shows the average PAAE over all applications in a cluster on a single core when predicting power and performance across combinations of P-States and Cl-States at runtime, and the error bars represent the STDEV. We analyze the data points with the higher error and also pointed out the sources of error below. (a) Average PAAE when predicting performance for Intel-2 for *thrashing* benchmarks is 15.8% because *Mcf* has 22.5% error as it is a pointer-chasing benchmark [17] and generates more than 41000 LLC misses per million instructions retired and the models are not trained for that range. On the other hand, applications like *Lbm – memory intensive – floating point benchmark* generate 3000 LLC misses per million instruction retired has an error 11.2%; (b) The average PAAE for performance for *Cache-Friendly* on AMD-0 and AMD-1 are 12.4% and 12.3%, respectively; this is because both clusters contain applications such as *Canneal* and *dedup*. The possible sources of error are: (1) Both applications have a high dynamic variability in application phases [12], which leads to erroneous counters due to PMCs multiplexing. (2) In contrast to the other applications across suites, these benchmarks have a shorter execution time. (3) Observe that Canneal is a cache fitting benchmark on Intel, by contrast it is a cache-friendly benchmark on AMD. This is because of the aggressive hardware prefetcher on Intel causing a higher miss rate [16], thereby leading to fewer dynamic phase changes and relatively smaller error of 6.5%.

We also observe that application `radix` is a cache-fitting, integer radix sorting algorithm, has very high activity in FE, across three different architectures, even though other benchmarks across four suites do not show this behavior.

Figure 2 shows average PAAE over all applications in each benchmark suite across architectures, with error bars representing STDEV. Across architectures, we observe performance PAAE is higher for SPEC benchmarks, which have high variability at runtime, and low for NAS benchmarks, which have less variability after the initialization phase. We conclude that the models to predict performance and power are accurate enough to capture the real behavior, and since the computational complexity at runtime is low, they can be used for fine-grain power/performance management. The models to predict power can be built using standardized power meters and the models are built using PMCs that are available across architectures.

Multicore Evaluation — We show the effectiveness of REPP-H on multicore architectures by selecting a configuration in a single step to meet power and performance requirements (see Section III-A) for 35 multiprogrammed workloads across 20 experiments (ten for power and ten for performance). The PAAE is computed using the error between the performance (or power) requirement and the value



(a) SSTN (b) FFFN (c) STFN

Fig. 4: PAAE when predicting performance and power for workloads SSTN, FFFN and STFN on Intel with change_factor High.

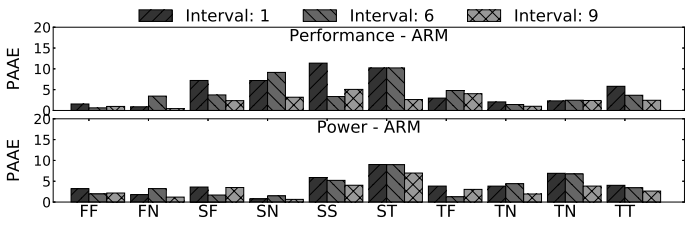


Fig. 5: Average PAAE when predicting power and performance on ARM for all workloads under different load_change `intervals` for change_factors Low, Mid and High. The x-axis shows multiprogrammed workloads generated based on methodology described in Section III.

measured from PMCs for performance (or power consumption as read from power meters).

Figure 3 shows the average PAAE for each workload when meeting the performance (ARM 7.1%, AMD 9.02%, and Intel 7.1%) and power (ARM 6.0%, AMD 6.6%, and Intel 8.1%) requirements in ten experiments across architectures. The error bars represent the STDEV across ten experiments for power and performance, which is less than 5.3% for each workload. We focus on analyzing those workloads with an error greater than 10% in both architectures. When predicting power, we observe an error of 13.8% (654.86 uJ) on AMD for workload `FNNN` that contains `ep.C`, which has very high activity ratio in BPU (4.2542) and FE (13.752). Similarly for workload `TTFN`, Intel has a higher error 11.8% (11.34uJ) than AMD, because on Intel we run two instances of `radix`, whereas on AMD we run a single one. Recall that the multiprogrammed are generated based on the methodology described in Section III, therefore we can not control the number of instances in a given workload. When predicting performance, we observe an error of 26.0% for `SFNN` on Intel because we run `lu_ncb`, which has non contiguous blocks of memory and the activity ratio in LLC is higher relative to other benchmarks. Similarly, workloads `TTTF` on AMD and `TTFN` on Intel have a power prediction error of 11.3% and 18.0%, respectively, because `radix` is part of the multiprogrammed workloads. The max. performance error on AMD, ARM and Intel are 19.4% (769 MIPS for `SSST`), 11.7% (5389 MIPS for `ST`) and 28.4% (13611 MIPS for `TFFF`). Similarly the max. power error AMD, ARM and Intel are 17.0% (101.72uJ for `TTFN`), 13.3% (10uJ for `SS`) and 16.6% (37.24uJ for `SSTF`), respectively.

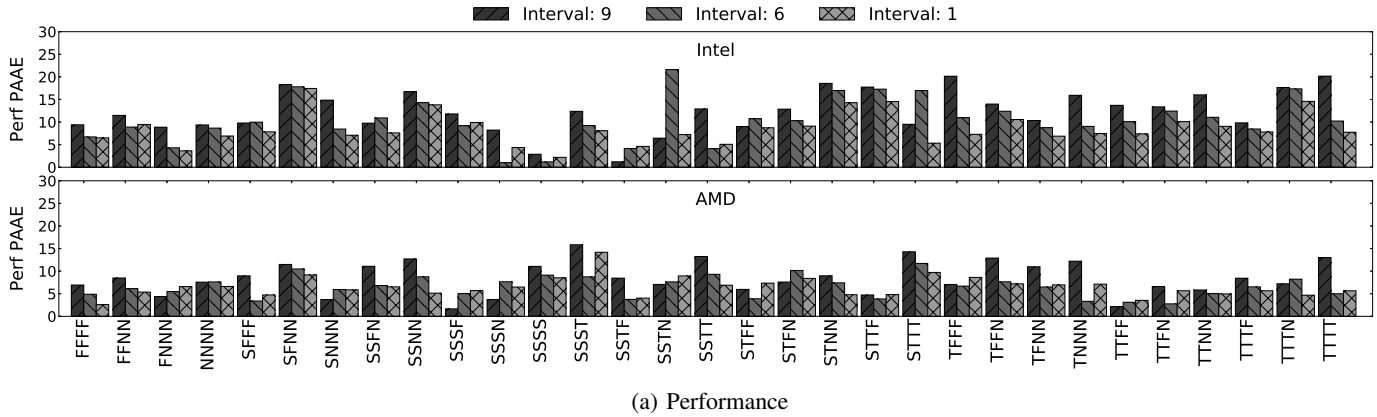
Figure 4 represents the performance on y-axis and power on x-axis for multiprogrammed workloads SSTN, FFFN, STFN on Intel architecture with *change_factor* High. The radius of each circle is the maximum prediction error, either power or performance (i.e, radius = max($PAAE_{power}$, $PAAE_{perf}$)). There exists multiple grayscale grading from low PAAE (*black*) to high PAAE (*white*). Although the maximum PAAE shown is at the 50% mark (the grayscale in Figure 4a is up to 50%), the number of error predictions with PAAE greater than 30% is less than ten. For SSTN, FFFN and STFN, the average error when predicting power (and performance) of 9.7% (3.3%), 8.7% (9.4%) and 8.4 (6.7%). This behaviour is observed across all workloads. However, due to space restrictions we only show selected workloads.

Figures 6 and 5 show the average PAAE for each workload under different load_change `intervals` on ARM and Intel/AMD when predicting performance (Figure 6a) and power (Figure 6b). Figures 5 and 6 are separated because ARM has different number of cores. Across the three architectures, faster (interval=1) load_changes have 3.5% higher error compared to slower (interval=9) load_changes because of aggressive changes in load in short burst cause numerous changes in configuration, thus leading to a higher error. On the other hand, slower load_changes lead to fewer changes in configuration, and have stable phases in application behaviour. Fortunately, such rapid changes in load seldom occur in data centers [24], [30]. Table IV summarizes the result obtained when predicting power and performance for load_change interval 1, 6 and 9.

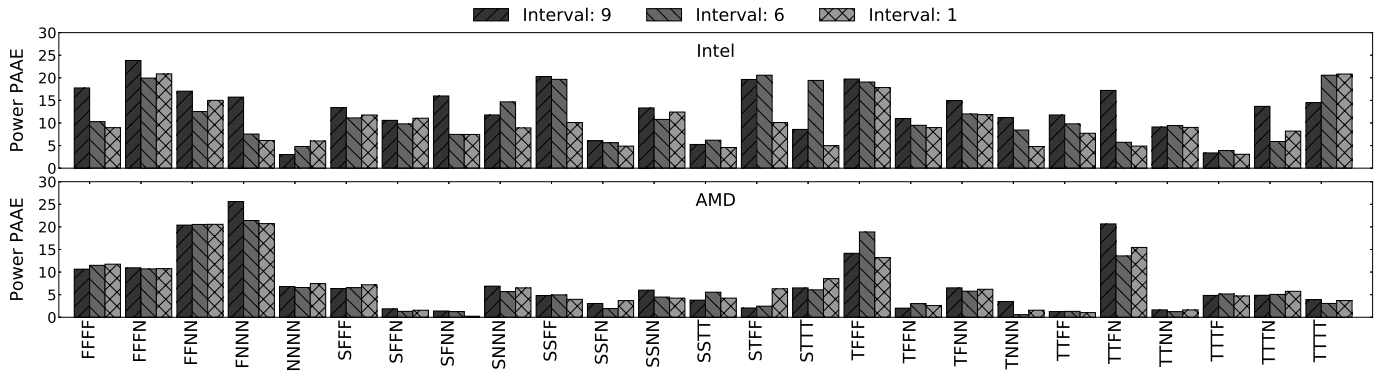
TABLE IV: Average PAAE for power and performance for load_change intervals 1, 6 and 9.

	Power			*Performance*		
Interval	1s	6s	9s	1s	6s	9s
Intel	11.1%	7.3%	5.6%	11.4%	10.6%	8.7%
AMD	7.2%	6.9%	6.7%	8.5%	6.5%	6.5%
ARM	4.2%	3.8%	2.9%	5.1%	4.6%	2.4%

Enabling Power/Performance Capping — Determining a configuration to meet the minimum performance (or not exceeding power consumption threshold) is usually done in



(a) Performance



(b) Power

Fig. 6: Average PAAE when predicting power and performance on Intel and AMD architecture for all workloads under different change `intervals` for change_factors Low, Mid and High. The x-axis shows multiprogrammed workloads generated based on methodology described in Section III.

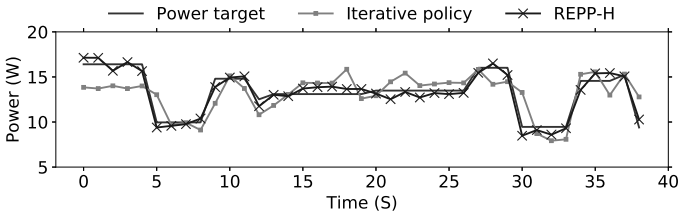


Fig. 7: Responsiveness to power capping for workload SSTT with constraint Random on Intel.

an iterative fashion (as in the RAPL driver on Intel processors [3]). A feedback control loop is often used to determine the configuration. If the power usage is above a certain threshold, the configuration is lowered. On the other hand, if the power usage is below a certain threshold, the configuration is increased to improve performance. In contrast, we provide a single-step mechanism to select configurations.

Figure 7 shows the responsiveness to (dynamic) power capping for workload `SSTT`, where the power capping limit is Random (see Section III-A) on Intel platform in the first 40 seconds. `SSTT` is composed of applications *streamcluster, lu.C, bwaves* and *soplex*. REPP-H changes P-States and meets the power target in 0.37 seconds on average, which is 3.6x faster than the iterative algorithm (used by Intel RAPL). This time includes sampling interval, latency to predict at all P-State and Cl-State and time to change P-State. Moreover REPP-H, provides 6% higher prediction accuracy.

IV. Related Work

Prior research works have focused on mapping applications to resources (mainly to CPUs/cores) to improve performance while saving power. In particular, Bellosa [8] used PMCs at run time to build a power-aware policy at OS level. Isci first showed that using PMCs it is possible to detect fine-grained application phases [14] and then show breakdown of power per component using multilinear models [15]. B. Rountree et al. [24] estimate performance (IPC, Instructions Per Cycle) across P-States by monitoring the number of leading load cycles. In [21] they predict performance on simulated architectures based on prefetch and variable memory access latencies.

In contrast to Mccullough et al. [20], who propose that linear regression of power tends only to work in restricted scenarios and will tend to over-fit based on application types. By contrast, our results have shown (Figure 1) that linear regression models built using a small training dataset do predict power and performance at runtime for a broad range of benchmarks, which are not a part of the training dataset, with high accuracy. Moreover, previous works [9], [15], [18], [27] show that linear regression techniques can be used to predict power with high accuracy.

Power control can also be done by mapping threads-to-cores more efficiently [13], [32]. For example, Pack & Cap [13] predicts performance with an offline analysis trained using multi-nomial logistic regression classifier. When a change in configuration is required at runtime, this classifier returns the best candidate operating configuration. S. Srinivasan et al. [28] predict the performance of threads running on heterogeneous cores, that is from one core type to another, using closed expressions. These expressions, however, do not suffice for a generic approach. In contrast, REPP-H builds power and performance models using PMCs, giving a more generic approach with low complexity. Since, REPP-H can predict performance-power for a distinct combination of P-States and Cl-States simultaneously. This makes REPP-H a good black-box for a single step fine grained per-core power or performance online optimization problem solver without external power meters or using application signatures [34].

The recent work on PEPP [30] (Performance, Energy and Power Predictor) proposes a system-level performance [29] and power model by taking advantage of the hardware PMCs available in AMD processors to estimate the total number of leading loads, and in turn, predicts performance and power across DVFS states. In contrast to PEPP, REPP-H improves in three distinct ways: (a) REPP-H builds performance and power models based on basic PMCs available across all architectures (Intel, AMD and ARM). However, the leading loads counter used in PEPP is exclusive to AMD processors [21]. (b) In addition to providing system level performance and power management techniques, REPP-H also facilitates thread level power and performance control. This is especially useful in multi-node, multi-core data centers consisting of numerous applications having different performance and power requirements. (c) Finally, REPP-H provides fine grained, in terms of time and precision, power and performance control because it also consider Cl-State.

V. Conclusion

We presented REPP-H, a scalable power and performance modeling and prediction technique to meet power caps and minimum performance requirements in a single step across heterogeneous architectures with processors' voltage and frequency controllers. REPP-H is built on statistical models and works by profiling a small set of applications on modern multicore systems leveraging hardware performance counters. We validate REPP-H using several single threaded and multiprogrammed workloads with average errors, respectively, on ARM, AMD and Intel architectures of 7.1%, 9.0%, 7.1% when predicting performance, and 6.0%, 6.5%, 8.1% when predicting power consumption. We argue that REPP-H can enable operators to better control power and performance in modern data centers that include server architectures with heterogeneous processing capabilities.

Acknowledgements — This work has been supported by the EU FP7 program (Mont-Blanc 2, ICT-610402), by the Ministerio de Economia (CAP-VII, TIN2015-65316-P), and the Generalitat de Catalunya (MPEXPAR, 2014-SGR-1051). The material herein is based in part upon work supported by the US NSF, grant numbers ACI-1535232 and CNS-1305220.

References

[1] *AMD64 Architecture Prog. Manual Volume 2: System Prog.*
[2] *Infocenter for ARM Cortex-A57*.
[3] *Intel 64 and IA-32 Architecture Software Developers' Manual*
[4] Electronic Educational Devices, Watts Up PRO, 2010
[5] *The PowerClamp driver*.
[6] Perf: Linux profiling with performance counters. https://perf.wiki.kernel.org/index.php/Main_Page
[7] D. H. Bailey, E. Barszcz et al. The NAS Parallel Benchmarks – Summary and Preliminary Results. In *Proc. of SC 1991*.
[8] F. Bellosa. The Benefits of Event: Driven Energy Accounting in Power-sensitive Systems. In *Proc. of 9th Workshop on ACM SIGOPS European Workshop: Beyond the PC*, 2000.
[9] R. Bertran, M. Gonzalez, et al. A Systematic Methodology to Generate Decomposable and Responsive Power Models for CMPs. *IEEE Transactions on Computers*, 62(7):1289–1302, 2013.
[10] C. Bienia, S. Kumar, et al. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In *Proc. of PACT 2008*.
[11] S. Blagodurov. *Addressing Shared Resource Contention in Datacenter Servers*. PhD thesis, Simon Fraser University, August 2013.
[12] D. Chasapis, M. Casas, et al. ParsecSs: Evaluating the Impact of Task Parallelism in the Parsec Benchmark Suite. In *ACM TACO*, 2015.
[13] R. Cochran, C. Hankendi, et al. Pack & cap: Adaptive dvfs and thread packing under power caps. In *Proc. of MICRO 2011*
[14] C. Isci and M. Martonosi. Phase characterization for power: evaluating control-flow-based and event-counter-based techniques. In *Proc. of HPCA 2006*.
[15] C. Isci and M. Martonosi. Runtime Power Monitoring in High-End Processors: Methodology and Empirical Data. In *Proc. of MICRO 2003*.
[16] H. Kang and J. L. Wong. To hardware prefetch or not to prefetch?: A virtualized environment study and core binding approach. *SIGPLAN Not.*, 48(4):357–368, Mar. 2013.
[17] T. Kumar Prakash and L. Peng. Performance Characterization of SPEC CPU2006 Benchmarks on Intel Core 2 Duo Processor. pages 36 –41, 2008.
[18] A. Lewis, J. Simon, et al. Chaotic Attractor Prediction for Server Runtime Energy Consumption. In *Proc of HotPower 2010*.
[19] J. Mars, L. Tang, et al. Heterogeneity in homogeneous; warehouse-scale computers: A performance opportunity. *Computer Architecture Letters*, 10(2):29–32, July 2011.
[20] J. C. McCullough, Y. Agarwal, et al. Evaluating the Effectiveness of Model-based Power Characterization. In *Proc. of the USENIX ATC 2011*
[21] R. Miftakhutdinov, E. Ebrahimi, et al. Predicting Performance Impact of DVFS for Realistic Memory Systems. In *Proc. of MICRO 2012*.
[22] R. Nathuji, C. Isci, et al. Exploiting platform heterogeneity for power efficient data centers. In *Proc. of ICAC 2007*
[23] R. Nishtala, M. G. Tallada, et al. A methodology to build models and predict performance-power in CMPS. In *Proc. of ICCPW 2015*
[24] B. Rountree, D. Lowenthal, et al. Practical performance prediction under Dynamic Voltage Frequency Scaling. In *Proc. of IGCC 2011*.
[25] D. Sanchez and C. Kozyrakis. Vantage: Scalable and Efficient Fine-grain Cache Partitioning. In *Proc, of ISCA 2011*
[26] S.Eranian. Issues with libpfm4.7.0 and perf on arm juno r0 https://sourceforge.net/p/perfmon2/mailman/message/34954001/
[27] K. Singh, M. Bhadauria, et al. Real Time Power Estimation and Thread Scheduling via Performance Counters. *SIGARCH Comput. Archit. News 37(2);46-55, May 2009*.
[28] S. Srinivasan, L. Zhao, et al. Efficient Interaction Between OS and Architecture in Heterogeneous Platforms. *SIGOPS Oper. Syst. Rev.*, 45(1):62–72, Feb. 2011.
[29] B. Su, J. L. Greathouse, et al. Implementing a Leading Loads Performance Predictor on Commodity Processors. In *Proc. of USENIX ATC 2014*.
[30] B. Su, J. Gu, et al. PPEP: Online performance, power, and energy prediction framework and dvfs space exploration. In *Proc. of MICRO 2014*
[31] S. Woo, M. Ohara, et al. The SPLASH-2 programs: characterization and methodological considerations. In *Proc. of 22nd ISCA, 1995*.
[32] R. Nishtala, D. Mossé, et al. Energy-aware thread co-location in heterogeneous multicore processors In *Proc. of EMSOFT 2013*
[33] Kanungo. T, Mount. D, et al An Efficient k-Means Clustering Algorithm: Analysis and Implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*
[34] S. Zhuravlev, J. C. Saez, et al Survey of Scheduling Techniques for Addressing Shared Resources in Multicore Processors. *ACM Comput. Surv.*, 45(1):4:1–4:28, Dec. 2012.