Contents lists available at ScienceDirect

Integration, the VLSI Journal

journal homepage: www.elsevier.com/locate/vlsi

CAPE: A cross-layer framework for accurate microprocessor power estimation

Monir Zaman [a,*], Mustafa M. Shihab [a], Ayse K. Coskun [b], Yiorgos Makris [a]

[a] The University of Texas at Dallas, Richardson, TX, USA
[b] Boston University, Boston, MA, USA

ARTICLE INFO

Keywords:
Cross-layer simulation
Workload profiling
Power modeling
gem5
AnyCore
SimPoint toolset

ABSTRACT

State-of-the-art system-level simulators can deliver fast power estimates for microprocessor designs, but often at the expense of reduced accuracy. The inaccuracies mainly stem from incorrect or over-simplified modeling of the target architecture. On the other hand, modern register-transfer level (RTL) simulators are cycle-accurate but overwhelmingly time consuming for most real-life workloads. Consequently, the design community often has to make a compromise between accuracy and speed. In this work, we propose a novel cross-layer power estimation (CAPE) technique that carefully integrates system-level and RTL profiling data for the target design in order to attain better accuracy. Our proposed methodology first leverages the SimPoint tool to transform a workload into weighted simulation points. We, then, present two different strategies to represent the critical segment of an application - either with a workload-specific simulation point (CAPE-WSSP) or, with the highest-weighted simulation point (CAPE-HWSP). Next, we profile the critical simulation point with an RTL simulator for maximum accuracy, while the other simulation points are simulated at system-level for fast evaluation. Finally, we input the integrated set of profiling data to the power simulator (McPAT). Our evaluation results show that CAPE can improve the power estimation accuracy by up to 15% for individual simulation points and by ~8% for the full application, compared to that of a system-level only simulation scheme while adding minimal runtime overhead.

1. Introduction

In recent years, continuous process scaling has rendered power dissipation a key consideration and figure of merit for microprocessor designs – often superseding the conventional performance parameters. At each stage of development, accurate simulation frameworks are instrumental for exploring the design space and identifying the Pareto-optimal points. Since exact technology libraries are initially not available for a new architecture, designers can simulate it with either a system-level model or a register-transfer level (RTL), for estimating performance and power. In fact, such selection between high and low level simulation frameworks results in a trade-off between accuracy and latency [1].

RTL description of designs are written in hardware description languages (HDL) such as VHDL or Verilog. An RTL model can imitate the actual hardware in a cycle-accurate manner, and thereby is exact. However, characterizing a microprocessor mandates simulation with real-life applications, which can be impractically time-consuming with RTL simulators. We illustrate this in Fig. 1 by comparing the RTL simulation time with that of a system-level (SL) simulator for three applications from the SPEC CPU2006 benchmark suite [2]. For example, simulating 100 million instructions of 401.bzip2 with a system-level simulator (i.e., gem5 [3]) takes only 54 min, whereas for an RTL simulator (i.e., AnyCore [4]) it takes 377 min – *a 600% increase in simulation time.* We can observe that this trend is common across all three benchmarks and degrades exponentially for higher number of instructions.[1] Furthermore, the latest intellectual properties (IP) are often copyrighted by the commercial vendors, and are unavailable in the public domain. Therefore, the research community often has to depend on dated and less accurate simulation models [5].

* Corresponding author.
E-mail addresses: monir.zaman@utdallas.edu (M. Zaman), mustafa.shihab@utdallas.edu (M.M. Shihab), acoskun@bu.edu (A.K. Coskun), yiorgos.makris@utdallas.edu (Y. Makris).

[1] The RTL simulation times for the full benchmarks have been extrapolated from that of their respective first 100 M instructions.

https://doi.org/10.1016/j.vlsi.2019.05.002
Received 9 October 2018; Received in revised form 11 April 2019; Accepted 3 May 2019
Available online 13 May 2019

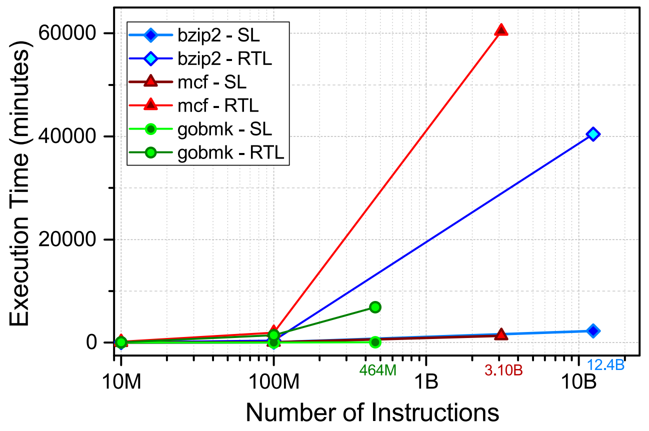


Fig. 1. RISC-V microprocessor simulation – Exponentially growing execution time often renders RTL simulation infeasible for designers.

On the other hand, system-level simulators, typically written in general-purpose programming languages such as C or, C++, model designs at a higher level of abstraction. Consequently, the system-level model of a microprocessor is significantly faster (and easier) to develop, modify, and parametrize for early design space exploration purposes, compared to its RTL counterpart. Most importantly, unlike RTL simulation, system-level simulators can profile large applications within reasonable time (Fig. 1). Unfortunately, such speedup in simulation comes with a compromise in accuracy [6,7]. While the system-level models attempt to capture the real hardware, they often fall short due to cycle inaccuracies and/or other internal design mismatches. These inaccuracies can be broadly categorized into *modeling*, *specification* and *abstraction* errors [1]. While the *modeling* errors tend to improve over time by validating against hardware, *specification* and *abstraction* errors are typically more persistent and difficult to correct [8].

The research community has ventured into multiple directions for addressing the challenges in design simulation. For example, Huang et al. proposed hardware/software co-simulation techniques for improved performance accuracy [9], while Sanchez et al. proposed to reduce simulation time by leveraging dynamic binary translation for instruction driven timing models [10]. Unfortunately, the prior works rely on the limited accuracy of the system-level simulator in collecting performance parameters and lack the accuracy of RT-level data. The inaccuracy of system-level simulation motivated Kim et al. to present a sampling technique for power accuracy improvement, but their work critically depends on the availability and applicability of FPGAs [11,12]. On the other hand, Walker et al. proposed a hardware-validated, improved power modeling technique; however, their technique depends on lengthy regression models and actual hardware for the validation, which is unemployable in early stage design explorations. Therefore, a feasible solution to make *fast and accurate* power estimation for microprocessors remains an open problem that is becoming increasingly more critical.

To this end, in this work, we present **CAPE** – a <u>C</u>ross-layer fr<u>A</u>mework that can facilitate accurate <u>P</u>ower <u>E</u>stimation by selectively integrating results from both system-level and RTL simulation of the target application.[2] We first leverage the concept of phase-based workload representation to represent the workload into different representative simulation points. We then isolate the most critical simulation point and simulate it at the RT-level for significantly improving the micro-architectural profiling accuracy. We propose two different strategies to select the critical simulation point – (i) CAPE-WSSP selects the critical workload specific simulation point (WSSP) based on micro-architectural characterization, whereas (ii) CAPE-HWSP chooses the highest weighted simulation point (HWSP) for RTL simulation. In both variations of CAPE, the non-critical simulation points are profiled with a system-level simulator for fast evaluation. Finally, we use the profiled data for each simulation point (i.e., RTL profiling for the critical simulation point and system-level data for the rest) to generate power estimation for individual simulation points using the power simulator. Next, the *weighted aggregate* is calculated in order to estimate overall power consumption by the respective workload. The **key** contributions of this work are as follows:

- We propose a cross-layer platform capable of integrating RTL simulation data with system-level profiling parameters in order to improve the accuracy of power simulator.
- We present two different strategies for finding the *critical* simulation point for the given workload: (i) CAPE-WSSP: based on user-defined workload-specific performance characteristics, and (ii) CAPE-HWSP: based on the highest-weight assigned by the SimPoint toolset.
- We present a comparative analysis of profiling data between system-level and RTL simulation of the *critical* simulation point to demonstrate the inaccuracies of the system-level abstraction.
- Lastly, we show power estimation accuracy improvement using the CAPE framework. We apply the proposed scheme on a state-of-the-art RISC-V microprocessor model and evaluate its performance for multiple SPEC CPU2006 workload applications.

Our evaluation results show that the proposed cross-layer framework can improve power estimation accuracy by up to 15% for individual simulation points and by approximately 9% for the full application, compared to that of existing schemes which leverage data from a system-level simulator only.

2. Background

2.1. Design simulation frameworks

In most cases, modern microprocessor designs are evaluated and tuned with either RTL or system-level simulators – particularly in the early stages of development. While the RTL simulators utilize behavioral HDLs for cycle-accurate modeling of the hardware, system-level simulators use high-level models that are faster, albeit less accurate. In the following sections, we briefly discuss the well-established RTL and system-level simulators leveraged in our proposed cross-layer scheme.

2.1.1. AnyCore toolset

The AnyCore toolset is based on a synthesizable, parameterized RTL model of a superscalar, out-of-order microprocessor core. The parameterized description renders it easy to modify various microarchitectural details. Currently, the toolset is able to simulate two different instruction sets – PISA [14] and RISC-V [15]. While AnyCore provides the option to choose between a dynamic or a static configuration, we use the static option in this work.

The AnyCore RISC-V RTL model implements the RV64G user-level ISA along with monitor-mode (M-mode) and supervisor-mode (S-mode) for the privileged levels. System calls in the benchmarks are handled by the RISC-V proxy-kernel (PK) and the front-end server. In addition, the AnyCore design includes a set of L1-caches, where the memory management and address translation tasks are performed by the functional simulator. The functional simulator also emulates the main memory.

Fig. 2 presents a high-level view of the AnyCore RISC-V co-simulation framework. The framework guarantees functional correctness of RTL simulation by using “Spike” (RISC-V functional simulator) for each committed instruction. Spike is also used to initialize the registers prior to actual simulation at the RT-level. At the beginning of the simulation, the benchmark is loaded into the *PK* which boots the

[2] A preliminary version of this work has been published in *International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, 2018 [13].

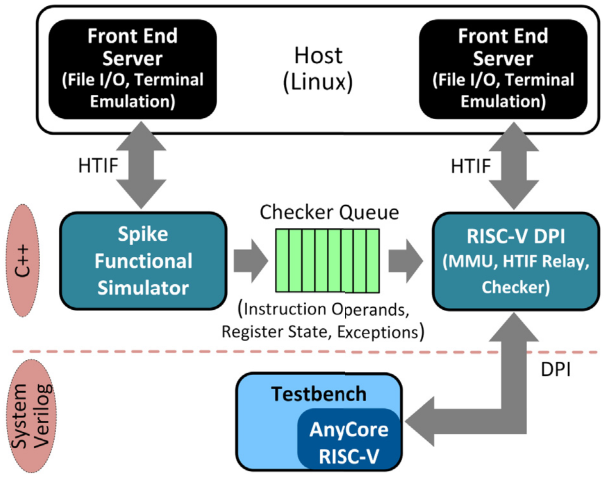

Front End Server (File I/O, Terminal Emulation)
Host (Linux)
Front End Server (File I/O, Terminal Emulation)
HTIF
HTIF
C++
Spike Functional Simulator
Checker Queue
RISC-V DPI (MMU, HTIF Relay, Checker)
(Instruction Operands, Register State, Exceptions)
DPI
System Verilog
Testbench
AnyCore RISC-V

Fig. 2. AnyCore framework: the high-level functional simulator verifies correctness of retired instructions, while DPI calls implement the functions at HDL-level.

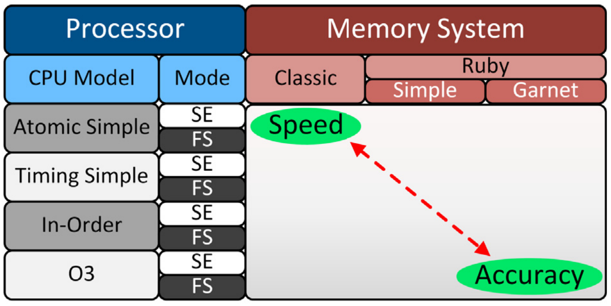


Fig. 3. The gem5 simulator provides multiple CPU models with varying focus on speed and accuracy.

CPU by setting up the registers, loading the benchmark to the main memory and setting the start program-counter (PC) for the benchmark. Once the desired instruction is reached, the framework starts running detailed simulation with the RTL simulator.

2.1.2. gem5 simulator

gem5 is one of the well-known system-level performance simulators in the open-source domain. As shown in Fig. 3, gem5 supports a range of CPU models, simulation modes and memory system hierarchy that corresponds to different levels of simulation speed and accuracy.

The gem5 CPU models are capable of capturing various processor designs and functionality. The *Atomic* CPU is the fastest but least accurate, while the detailed CPU corresponds to most time-consuming but accurate simulations. The detailed CPU model has two sub-categories – the *In-Order* and the *Out-of-Order* (O3/DerivO3) models. Both of the detailed CPU models are pipelined and highly configurable.

In addition, the gem5 CPU models can run in two simulation *modes* – the system-call emulation (SE) mode and the full-system (FS) mode. In the SE mode, no operating system is loaded by gem5 during the simulation and system-calls are emulated by the host system. In contrast, the FS mode executes both user-level and kernel-level instructions, and models a complete system by loading an OS in the simulator. The OS boots the machine, simulates all the system-calls, and handles the virtual-to-physical translations.

Also, gem5 is capable of modeling data and instruction caches, memory management unit (MMU) and a unified L2 cache, and supports two types of memory hierarchy. For simpler memory modeling, gem5 uses the *Classic* memory model, where the emphasis is put on the pipeline simulation. The memory uses simple timing model to calculate hits, misses and other memory performance data. On the other hand, the *Ruby* memory model contains various coherence protocols, and can support a more detailed memory hierarchy simulation.

Finally, while the gem5 simulator can simulate different instruction set architectures (ISA), we use the recently implemented RISC-V ISA in gem5 [16].

2.2. SimPoint toolset

While the most accurate method to profile a workload is to simulate all the instructions, for many real-life applications such an evaluation can be impractically long. For example, the SPEC CPU2006 benchmark on average contain 2249.75 billion instructions and executing even a system-level simulation can take days [17,18]. The SimPoint tool addresses this issue by generating representative phases of a workload and aggregating the results in order to represent the whole application [19]. The tool identifies and isolates unique phases/regions where the program execution is stable and has a relatively constant CPI. SimPoint starts by generating dynamic execution traces of the given workload and then slices them into user-defined sizes. Typically, slices of 1 M or 10 M instructions can deliver high accuracy with reasonable simulation times [20]. The tool then uses a K-means algorithm to form clusters of slices. Towards the end of this stage, a representative slice is chosen from each cluster and set as a *simulation point*. Each simulation point is assigned a weight based on the cluster size it represents, and the sum of the weights is always 1 (i.e., the full application). The weighted simulation points can be simulated in parallel and then aggregated based on weight, in order to generate a fast and accurate characterization profile for the full application. For example, when using the SimPoint tool, Sherwood et al. reported an average IPC error of 3% for SPEC CPU2000 benchmark running Alpha binaries [21].

2.3. McPAT simulator

The McPAT power simulator integrates power, area, and timing information for comprehensive design space exploration [22]. The framework is capable of simulating power based on the built-in models for different technology nodes, single or multi-core, in-order and out-of-order processors. Furthermore, McPAT model also takes into consideration the off-chip elements, such as network-on-chip, multi-domain clocks, caches, etc. The framework uses an XML based interface as an input to the program. The interface contains critical micro-architectural information, as well as various activity factors which contribute to the power consumption of the microprocessor. The performance simulators generate the activity factors which are then parsed to fill out the XML interface. The power, area and timing models in McPAT are validated against multiple microprocessors and the hierarchical output report contains information on low-level pipeline modules.

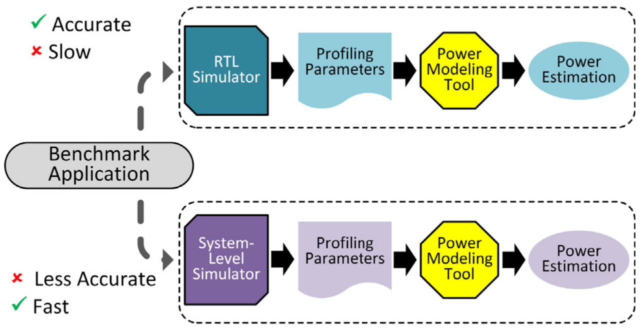


Fig. 4. Conventional power estimation frameworks: Benchmarks are run either using system-level simulator or RTL simulator.

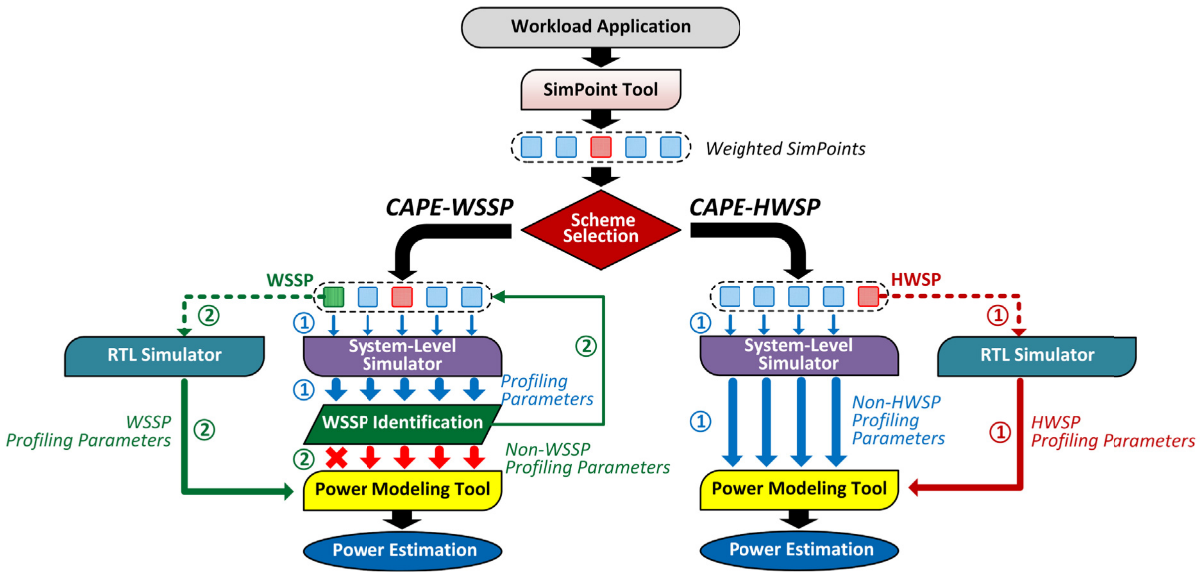


Fig. 5. Proposed cross-layer power estimation framework.

3. Cross-layer framework for power estimation

3.1. Overview

The conventional setting for performance and power modeling uses either a system-level or an RT-level simulation framework. The high-level depiction for this setting is shown in Fig. 4. The micro-architectural profiling data, as well as performance parameters (e.g., IPC) generated from workload execution, are captured by one of these frameworks and then forwarded to a power simulator. The power simulator utilizes such profiling parameters and the activity factors for various micro-architecture modules in order to calculate power consumption.

There are two critical takeaways regarding the existing methodology for power estimation. First, while RTL simulators typically possess an accurate description of a microprocessor using cycle-accurate HDL, simulating real-life workload applications with them can often be impractically time-consuming, which in turn forces the designers to opt for the less accurate system-level simulators [23]. Second, the accuracy of the power simulator critically depends on the accuracy of the profiling data it receives as an input. The workload activity profile is given as the input to the power simulator, which is then used to estimate power consumption based on the pre-built power models. Therefore, it is critical to provide the most accurate profiling data as the input to the power simulator in order to accurately estimate power consumption of the design.

In this work, we propose a cross-layer approach to overcome the challenges of stand-alone system or RT-level full workload simulation. Specifically, the cross-layer framework simulates representative phases of the workload at both system and RT-level. The phases are generated by the SimPoint toolset, each marked with an individual weight. The 'critical' phase is determined either by the weight assigned by the SimPoint toolset or by the 'critical score' calculated by the user based on microarchitectural performance parameters. Adopting the simpoint-based representation allows our cross-layer approach to eliminate lengthy RT-level simulation of the complete workload. Instead, we run the RT-level simulation only on the 'critical' representative phase.

After simulation completes in the CAPE framework, the microarchitectural parameters for each simulation point are collected and provided as an input to the power simulator for power estimation. The power simulator gives individual simulation point power as its output. Finally, power estimation for the entire workload is calculated by the weighted sum of all simulation points.

Our proposed framework makes the following assumptions:

- Because our framework provides improved accuracy at the early design stage, we assume that we only have access to the RT-level design of the microprocessor. The RT-level simulation provides highly accurate microarchitecture performance parameters to its faster system-level counter-part. When measuring any improvements achieved by the CAPE framework, this RT-level simulation results are considered as the baseline performance.
- The microarchitectural performance parameters for the full workload is available to us from offline simulation. This can be generated using the system-level simulator, as it is impractical to simulate the full workload using the RT-level simulator.

We present the CAPE framework in Fig. 5. The framework has four distinct steps for accurate power estimation:

Step 1: Represent the workload in phases (simulation points) using a workload phase detection toolset.
Step 2: Determine the CAPE scheme to be used and choose the 'critical' simulation point.
Step 3: Simulate the 'critical' simulation point at the RT-level; the other non-critical simulation points are simulated at the system-level.
Step 4: Finally, compute power consumption for each simulation point using the power simulator and aggregate the weighted results to represent overall power consumption for the workload.

We propose two separate schemes for the cross-layer simulation framework. Both schemes require phase based representation of the whole workload. This is shown in Fig. 5. We use the SimPoint toolset for generating these phases, also known as the simulation points or simpoints. Simpoints allow us to only simulate a fraction of the overall instructions to represent the whole benchmark. The accuracy of the simpoint representation for each workload depends on the granularity of the interval size and the number of maximum simulation points allowed. It is worth noting that simpoints can be simulated in parallel. Therefore, the total simulation time to characterize the whole workload is represented by the following equation:

$$\textit{Overall workload characterization time} = \max\left(\textit{Profiling time for a simulation point}\right) \tag{1}$$

Given the same number of instruction simulation, the RT-level takes larger amount of time to complete. Thus, based on Eq. (1), the time needed to characterize the performance of a workload using simulation points is bound by the time of the RT-level simulation. After completing the benchmark representation, the next step is to determine the CAPE scheme and identify the 'critical' simulation point.

3.1.1. CAPE-WSSP: workload specific SimPoint selection

Our first scheme for the cross-layer simulation framework utilizes workload specific simulation points, which we call CAPE-WSSP. Using the system-level simulator, we initialize this scheme by running detailed simulation of all the simpoints generated by the SimPoint toolset. At the end of simulation, each phase generates its representative performance parameters, which we carefully profile. From the initial performance profile, we select key performance parameters and generate a 'critical score'.

We formulate the 'critical score' with the following equation:

$$\textit{critical score} = \sum_{i=0}^{N} (w_i \times v_i) \tag{2}$$

where w is the weight of the parameter, v is the value of the parameter, and N is the total number of profiled parameters.

We note that the key performance parameters are chosen and corresponding weights are assigned based on the designer's requirement and the workload-specific characterization of the design. Therefore, the 'critical score' for a given phase will vary.

Next, we generate the 'critical score' for the full workload simulation. Performance profiling for the full workload is available to us from offline simulation. For this step, the key parameters and their weights remain the same, and the generated 'critical score' represents the overall workload specific characteristics.

In the CAPE-WSSP scheme, we complete the selection process for the 'critical' phase by comparing the calculated scores between the full workload simulation and the simpoint-based simulation. The simpoint with 'critical score' closest to that of the full workload is considered as the 'critical' phase for this scheme. Consequently, this 'critical' simpoint is then simulated at the RT-level for accuracy improvement. If a given benchmark has multiple 'critical' simpoints, the user has the option to simulate all or one 'critical' simpoint at the RT level.

Each abstraction level generates corresponding performance parameters for every simpoint after the simulation completes. These parameters, along with the microarchitectural details are then used as input for the McPAT power simulator. The power simulator calculates power for each simpoint. Finally, for the overall workload power estimation, the McPAT simulator aggregates result based on all simpoint weights.

3.1.2. CAPE-HWSP: weight-based SimPoint selection

The second proposed scheme for our cross-layer framework uses the individual weights generated by the SimPoint toolset for each simpoint to determine the benchmark specific 'critical' phase. We call this scheme CAPE-HWSP, as it uses the highest weighted simulation point (HWSP) for the 'critical' phase detection. In this methodology, we simulate the HWSP at the RT-level for accuracy enhancement over the system-level simulator. The SimPoint toolset determines the weight of an individual simpoint by calculating the amount of time and the number of basic blocks present for each phase during simulation. The simpoint requiring longest simulation time for a phase is designated as the highest weighted simulation point by the tool. Therefore, improving the performance profiling accuracy of the HWSP increases the overall workload characterization in the CAPE-HWSP scheme. If multiple HWSPs are present for the workload, in the current CAPE-HWSP scheme, we select the one occurring first based on the simpoint id.

The final step in this scheme uses the performance parameters generated for individual simpoints as the input for the McPAT power simulator. The individual simpoint power estimation is then combined using the weighted-sum to generate the overall workload power consumption.

3.2. Implementation

In this section, we provide implementation details for each of the steps described in Section 3.1.

3.2.1. Simpoint generation

In the first step of our framework, we generate simulation points for each benchmark using the SimPoint toolset (version 3.2) [19]. We compile five benchmark application from the SPEC CPU2006 benchmark suite [2] for the RISC-V instruction set architecture [15]. The applications are chosen from the SPEC CPU2006 integer suite and we use Speckle [24] wrapper to compile them. We use the *riscv-gnu-toolchain* with *-O3* and *-static* flags for compiling the benchmarks, and use the *test* input set to run the simulation. We generated simulation points with three different cluster sizes: 1 M, 10 M, and 100 M. Finally, we evaluated each cluster size for a maximum simulation point number of 6, 10 and 20.

3.2.2. Configuration for RTL and system-level simulation

AnyCore RTL: We use the AnyCore RISCV toolset for RTL simulation. Specifically, we configure the *static core-1* setting for our experiments with minor modification. This setting allows the superscalar, out-of-order microprocessor to fetch, decode, and rename one instruction every clock cycle. The core issues three instructions each cycle and has three functional units in the pipeline. At every clock, one instruction is committed. The pipeline also implements a simple 2-bit branch predictor unit to predict branch directions in the fetch stage. The L1 cache is direct-mapped and L2 cache is always assumed to be a hit. Table 1 shows key microarchitectural details for the *core-1* settings used in the RTL simulation. The functional simulator in the AnyCore framework is used for fast-forwarding the simulation to the desired starting point. Once the desired instruction is reached, the functional simulator transfers the architectural register states to the RTL framework. The detailed RTL simulation starts at that point and continues until the end of the simulation stop point. The performance parameters are probed from different pipeline stages and reported at the end of the simulation.

gem5 Simulator: We start by modifying the gem5 simulator to match the microarchitectural details in Table 1. This modification includes changing the branch predictor unit, pipeline width, and depths, different stage parameter sizes, etc. We also modify the functional units' latency and the number of functional units used by the gem5 out-of-order CPU. The number of pipeline stages in the RTL

Table 1
Microarchitecture details for AnyCore core-1.

Feature	Value	Feature	Value
Fetch-to-Dispatch width	1	L1 Ins. Cache	2 KB
Issue-to-Execute width	3	L1 Data Cache	8 KB
Retire width	1	Active List size	96
Issue Queue	16	Functional units	3
Load/Store Queue	32/32	Physical Register	160
BTB size	1024	RAS	16
BPU entries	1024	Floating-point Pipeline	0

design is matched by modifying the individual pipeline stage delay in gem5. Finally, we turn off the indirect branch predictor in gem5 because the RTL design does not use this feature.

We simulate each simulation points using the out-of-order detailed CPU in parallel using the modified gem5. To reduce the effect of cold-cache start, we run 100 million instructions for warm-up prior to running detailed simulation from the simulation start point. If the detailed simulation starting point is less than 100 million instructions, then we either warm-up for 1 million instructions or do not warm-up at all (in the case of detailed simulation starting from the first instruction count). At completion, gem5 generates detailed performance parameter statistics for each simulation points.

3.2.3. *Power estimation with McPAT power simulator*

The final step in our framework uses the McPAT power simulator to estimate runtime dynamic power consumed by the core [22]. McPAT uses a detailed XML file as its input. The input file contains various architectural details and activity data for various performance parameters, which are generated by either gem5 or RT-level simulator. We mimic the micro-architectural details of the microprocessor and modify the generic McPAT model to represent the AnyCore RTL design. For parameters, such as 'duty cycle', we use the default values in the XML file. The effect of using default values for these parameters for both gem5 and RTL data remains the same. McPAT can generate peak, leakage and total runtime power consumption by the core and each of its sub-modules for a chosen process technology. We use 65 nm technology node for the power estimation results.

4. Evaluation

In this section, we discuss our evaluation results and demonstrate the improvements achieved in power estimation accuracy with the proposed CAPE framework.

4.1. *Sensitivity analysis of different SimPoints*

The accuracy of the SimPoint toolset depends on the simpoint cluster size and the total number of simpoints available [25]. In this section, we study the impact of cluster size and the maximum number of simulation points for the five SPEC2006 benchmarks: `401.bzip2`, `429.mcf`, `445.gobmk`, `458.sjeng` and `471.omnetpp`.

4.1.1. *Varying simulation point interval sizes*

To create the 'full' profile for each benchmark, we start by simulating them in gem5 simulator using the detailed out-of-order CPU without any SimPoint representation. Next, each benchmark is represented with 1 M, 10 M and 100 M simulation point interval sizes with maximum 6 simulation points. Each simulation points are then simulated with the detailed CPU in gem5. After simulation completes, the weighted sum of the instruction-per-cycle (IPC) parameter is used as the simpoint-based representation of each benchmark. In Fig. 6, we show the accuracy of each simpoint-based representation of IPC compared with the 'full' benchmark IPC generated from offline simulation. The result shows, IPC for `401.bzip2`, `429.mcf`, `445.gobmk`, `458.sjeng` and `471.omnetpp` are 0.24, 0.08, 0.204, 0.13 and 0.06, respectively for 'full' benchmark simulation. This is shown as 'Full BM' in the figure.

For 1 M simpoint cluster size, the IPC for `401.bzip2`, `429.mcf`, `445.gobmk`, `458.sjeng` and `471.omnetpp` are 0.24, 0.08, 0.35, 0.16 and 0.06, respectively. This result is represented as 'SPS-1M' in Fig. 6.

In the same figure, 'SPS-10 M' represents the cluster size of 10 M instructions. The IPC for `401.bzip2`, `429.mcf`, `445.gobmk`, `458.sjeng` and `471.omnetpp` are 0.43, 0.087, 0.22, 0.13 and 0.06, respectively for the 10 M simulation point size.

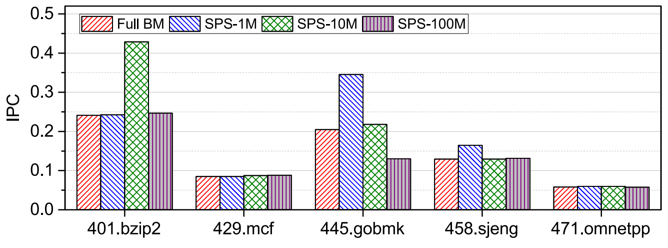


Fig. 6. IPC differences for 'full' vs. simpoint representation of benc hmarks simulated in gem5. Each benchmark is represented by 1 M, 10 M and 100 M simpoint size with the maximum of 6 simulation points.

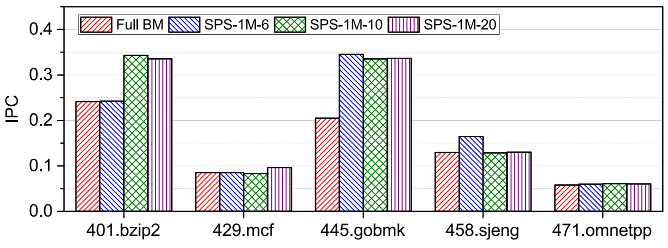


Fig. 7. IPC difference between the full benchmark and different simpoint variations. The maximum number of simpoints (max K) are varied between 6, 10 and 20 for the 1 M simpoint cluster size.

Finally, the IPC for 100 M simpoint interval size (represented as 'SPS-100 M') is 0.25, 0.09, 0.13, 0.13 and 0.06 for `401.bzip2`, `429.mcf`, `445.gobmk`, `458.sjeng` and `471.omnetpp` benchmark, respectively.

We can observe that the average (geometric mean) IPC difference for the five benchmarks is 2%, 3%, and 3% respectively for 1 M, 10 M, and 100 M simulation point interval sizes when compared against the 'full' benchmark IPC. This result implies that the 1 M simulation point interval size gives the closest IPC representation for the five benchmarks we have simulated. The 10 M and 100 M simulation point representation is also reasonably close and, therefore, can be chosen for the CAPE framework. However, simulation time at the RT-level increases exponentially with increasing number of simulated instructions, therefore, it is often impractical to simulate larger number of instructions (for example 100 M) at the RT-level for accuracy improvement. In this work, we chose the 1 M simpoint interval size for the CAPE framework.

4.1.2. *Varying maximum number of simulation points*

From Section 4.1.1, we extend our simulation for 1 M simulation point interval size with varying simulation points number (max K). We vary the max K size between 6, 10 and 20 and observe the IPC accuracy for each variation. Our objective is to find the max K value for which the difference in IPC is minimum compared with the 'full' benchmark result.

Fig. 7 shows the comparison of max K values. The IPC for the 'full' benchmark simulation is shown as 'Full BM'. Our results show that for max K value of 6, the IPC for `401.bzip2`, `429.mcf`, `445.gobmk`, `458.sjeng`, and `471.omnetpp` is 0.24, 0.08, 0.35, 0.16 and 0.06, respectively.

Similarly, for max K value of 10, the IPC is 0.34, 0.08, 0.34, 0.13, and 0.06, respectively for `401.bzip2`, `429.mcf`, `445.gobmk`, `458.sjeng`, and `471.omnetpp` benchmarks.

Finally, for max K value of 100, the IPC observed for `401.bzip2`, `429.mcf`, `445.gobmk`, `458.sjeng`, and `471.omnetpp` is 0.34, 0.10, 0.34, 0.13, and 0.06, respectively.

When compared against the IPC obtained from 'FULL BM', the overall average (geometric mean) IPC varies by 2.32%, 7.15%, and 9.24%, for max K values of 6, 10 and 20, respectively. Therefore, for the five

Table 2
SimPoint details for evaluated SPEC CPU2006 benchmarks.

Benchmark	Total Instruction (in millions)	Number of Simpoints	Instruction Per Simpoint	Simpoint ID	Weight	Starting Instruction (in millions)	Ending Instruction (in millions)
401.bzip2	12403	6	1 million	0	0.632	10093	10094
				1	0.066	817	818
				2	0.087	1497	1498
				3	0.016	8839	8840
				4	0.037	8224	8225
				5	0.162	4470	4471
429.mcf	3102	5	1 million	0	0.532	1776	1777
				1	0.063	11	12
				2	0.206	2177	2178
				3	0.112	1183	1184
				4	0.088	2918	2919
445.gobmk	464	6	1 million	0	0.173	434	435
				1	0.153	232	233
				2	0.108	62	63
				3	0.279	277	278
				4	0.212	80	81
				5	0.076	9	10
458.sjeng	19126	5	1 million	0	0.009	229	230
				1	0.422	8339	8340
				2	0.183	864	865
				3	0.007	101	102
				4	0.379	12951	12952
471.omnetpp	1772	6	1 million	0	0.195	320	321
				1	0.476	423	424
				2	0.005	1766	1767
				3	0.001	1771	1772
				4	0.001	0	1
				5	0.322	211	212

benchmarks we simulated, the SimPoint tool setting with 1 M simpoint interval size with max K value of 6 is the closest average representation of the full benchmark. Table 2 shows the detailed SimPoint breakdown along with overall SimPoint generation runtime for each simulated benchmark.

4.2. Case study 1: CAPE-WSSP

Based on the result from Section 4.1.2, we break each benchmark into representative phases using the SimPoint tool. Next, the ‘critical score’ for ‘full’ benchmark and each simpoint are calculated using Eq. 2. We select three performance parameters: branch, load and store instruction count and assume 40% weight for the branch instructions and 30% for load, and 30% for store instructions. These non-timing related performance parameters are used only as an example. The user has the option to select any other performance parameters as necessary. For example, if the user needs to find more accurate performance profiling of the microprocessor register accesses, then s/he can use the `register read` count as one of the key performance parameter and assign a higher weight to it. For floating point unit utilization, the user should include `floating point instruction` count as a key parameter with appropriate weight distribution, ensuring workload specific simpoint selection. The ‘critical’ simpoint in the CAPE-WSSP scheme varies based on the selected performance parameters and their weights.

Based on the weights selected, the ‘full’ benchmark’s ‘critical scores’ are 153.7, 236.7, 194.2, 149.8, and 212.8 for `401.bzip2`, `429.mcf`, `445.gobmk`, `458.sjeng`, and `471.omnetpp` benchmarks, respectively. Next, we calculate the ‘critical score’ for each simpoint for individual benchmark and summarize in Table 3. For each of the benchmarks, the simpoint highlighted in yellow represents the critical simpoint as it corresponds to the critical score *closest* to that of the full benchmark simulation. Specifically, the critical simpoint ids are: 5, 3, 3, 1, and 5 for `401.bzip2`, `429.mcf`, `445.gobmk`, `458.sjeng`, and `471.omnetpp` benchmarks, respectively.

We present the differences observed for various executed performance parameters in Fig. 8. The ‘critical’ simpoint for each benchmark is simulated both in AnyCore RTL and in gem5 simulator for 1 M instructions.

Load count. Fig. 8a shows the difference in number of load instructions for five simulated benchmarks. The WSSP of `458.sjeng` exhibits the lowest difference of 6% between the AnyCore and gem5 simulation, while the difference is highest for `445.gobmk` at 66%.

Store count. The comparative result for number of store instructions is shown in Fig. 8b. We can see that `458.sjeng` again manifests the minimum difference of 3%, whereas for `429.mcf` the difference is the maximum at 82%.

Branch count. Fig. 8c shows that the difference in branch instruction count is the highest for `401.bzip2` at 94%. In contrast, `458.sjeng` shows the least difference of 8%.

Branch misprediction. In Fig. 8d, we see an average difference of 48% across five benchmarks for branch misprediction count. `401.bzip2` exhibits the lowest 30% difference, while for `445.gobmmk` it stands highest at 87%.

Cache miss. Fig. 8e portrays the fluctuation in instruction cache misses between the two simulation platforms. We observe that,

Table 3
Critical simulation point selection for CAPE-WSSP scheme.

Benchmark	Full Benchmark (PKI)				Simpoint ID	Performance Parameters (PKI)			
	Branch Count	Load Count	Store Count	Critical Score		Branch Count	Load Count	Store Count	Critical Score
401.bzip2	136	222	109	153.7	0	128	197	57	127.29
					1	126	391	158	215.10
					2	100	498	169	239.98
					3	123	209	39	123.87
					4	138	73	39	88.75
					5	12	331	169	154.82
429.mcf	270	381	48	236.7	0	267	355	54	229.24
					1	220	181	101	172.58
					2	337	441	0	267.01
					3	240	467	30	244.95
					4	252	504	0	252.22
445.gobmk	175	205	209	194.2	0	143	286	143	185.71
					1	233	89	345	223.41
					2	143	286	143	185.71
					3	143	286	143	185.71
					4	158	215	141	169.98
					5	143	286	143	185.71
458.sjeng	181	184	74	149.8	0	181	182	65	146.61
					1	184	181	71	149.00
					2	180	185	71	148.65
					3	250	0	500	250.00
					4	178	180	83	150.34
471.omnetpp	226	243	165	212.8	0	225	246	168	214.20
					1	225	247	168	214.55
					2	265	98	210	198.08
					3	1000	0	0	400.00
					4	245	151	80	167.44
					5	225	247	168	214.54

`401.bzip2`, `429.mcf`, `445.gobmk`, `458.sjeng`, and `471.omnetpp` report differences of 41%, 64%, 31%, 61% and 44%, respectively. Additionally, Fig. 8f shows the differences in data cache misses to be 47%, 55%, 60%, 71% and 61%, for `401.bzip2`, `429.mcf`, `445.gobmk`, `458.sjeng`, and , `471.omnetpp`, respectively.

IPC. As our final point of comparison, we present the comparison of IPC values attained from the gem5 and the AnyCore simulator in Fig. 8g. One can note that, `458.sjeng` exhibits the minimum difference of 2%, while `429.mcf` shows the maximum difference of 83%. The average IPC difference observed is 21% for the simpoints in the CAPE-WSSP scheme.

4.3. Case study 2: CAPE-HWSP

In CAPE-HWSP scheme, we use the HWSP for each benchmark as the critical simulation points. Each simpoint details along with corresponding weight is listed in Table 2 with the HWSP highlighted. In order to explore the amount of discrepancy in the various performance parameters, we simulate the HWSP of each benchmark in both the RTL (AnyCore) and the system-level (gem5) simulator. The variations are presented in Fig. 9. Each simulation is run for 1 million instructions from the starting point listed in Table 2.

Load count. Fig. 9a shows the difference in number of load instructions for all five benchmarks. We can see from the figure that, the HWSP of `429.mcf` exhibits the lowest difference of 5% between the RTL and system-level simulation followed by `401.bzip2` and `458.sjeng` with a difference of 6%. The HWSP for `445.gobmk` benchmark shows the maximum difference of 66% for its system-level counterpart.

Store count. As shown in Fig. 9b, for store instructions, `445.gobmk` again manifests the maximum difference of 61%, whereas for `458.sjeng` the difference is the minimum at 3%. The overall average difference across five benchmarks for the 'store count' is 16%.

Branch count. Fig. 9c shows that the difference in branch instruction count is the highest for `445.gobmk` at 39%. In contrast, `458.sjeng` shows the least difference of 8%.

Branch misprediction. As shown in Fig. 9d, for branch misprediction count, `429.mcf` exhibits the minimum difference of 17%, while for `401.bzip2` it stands maximum at 95%.

Cache miss. Fig. 9e compares instruction cache misses for the two simulation platforms. The average instruction cache miss across five benchmarks is 52%. The `445.gobmk` shows the maximum and `401.bzip2` benchmark shows the minimum amount of difference for this parameter. Similarly, in Fig. 9f, we find an average difference of 55% across five benchmarks for the data-cache miss count. For this performance feature, the `429.mcf` benchmark shows the minimum variation of 29%.

IPC. As our final point of comparison, we present difference in IPC values attained from the gem5 and the AnyCore simulator in Fig. 9g. One can note that, `471.omnetpp` exhibits the highest amount of variation in IPC at 75%, while `458.sjeng` shows the lowest variation of 2%.

We further discuss the reasons for the observed variations in performance parameters (and, thereby, in IPC) between gem5 and RT-level simulation in Section 4.6.

4.4. Power estimation results

For power estimation, we present (i) individual power estimation variations observed for the 'critical' simpoints, and (ii) overall workload power estimation variation using combined weighted-sum of simpoints. The performance parameters obtained from RTL simulation is the *baseline performance*, and the power estimated by the McPAT simulator using these values, therefore, is the baseline power for 'critical' simpoints. Figs. 10 and 11 present power estimation variations observed

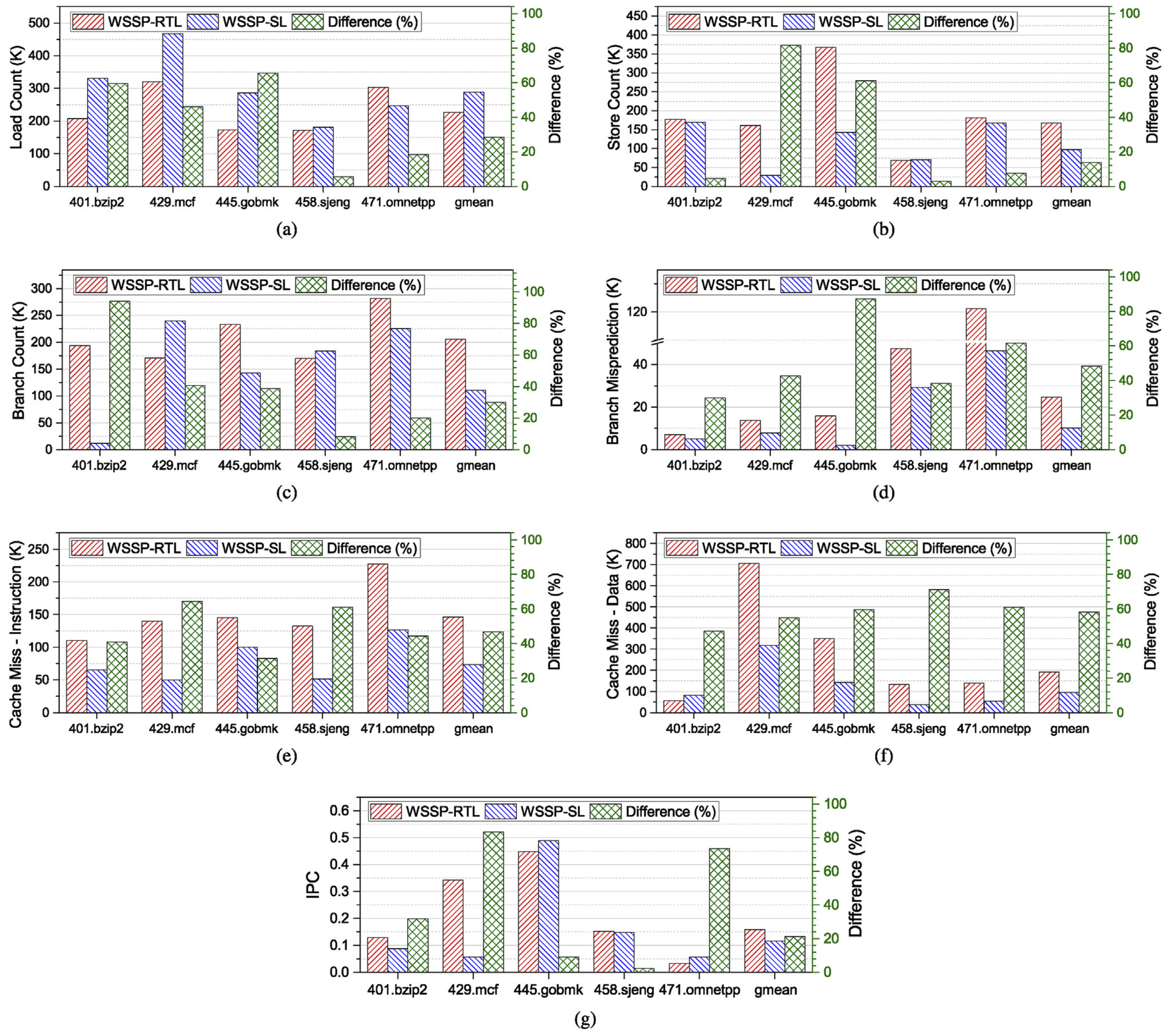


Fig. 8. Comparison of parameter profiling between gem5 and AnyCore RTL framework for simulation points selected in CAPE-WSSP scheme.

using the CAPE-WSSP and CAPE-HWSP schemes, respectively. For comparison, both schemes present 'gem5' as the power estimated *only* from the system-level simulation using simpoint representation of the benchmarks.

The improvement in power estimation is measured as:

$$\text{Power improvement} = \frac{\text{baseline power} - \text{gem5 power}}{\text{baseline power}} \qquad (3)$$

Fig. 10a shows an average power estimation improvement of ~6% across all five benchmarks when only the 'critical' simpoints are considered in the CAPE-WSSP scheme. The maximum power improvement of 14% is observed for the `445.gobmk` benchmark and the minimum power improvement is noticed for the `458.sjeng` benchmark for the respective 'critical' simpoints. This matches with the trend observed in Section 4.2, where `458.sjeng` has minimum difference for five out of seven key performance parameters we profiled.

The overall workload power estimation using the CAPE-WSSP scheme is presented in Fig. 10b. Results show an average power variation of 1.5% when compared between gem5-only performance parameter used for each simpoint and CAPE-WSSP scheme used for the critical simpoint.

Fig. 11a shows power differences observed for the HWSP between performance parameters generated from gem5-only and RTL-only simulation. We see an average power improvement of 8% for individual HWSPs across five benchmarks. For individual HWSPs, `445.gobmk` shows maximum and `458.sjeng` shows minimum amount of power estimation difference. The overall benchmark power estimation comparison between 'gem5-only' simpoint simulation and CAPE-HWSP scheme is presented in Fig. 11b. This shows an average of ~4% accuracy improvement over five benchmarks using the CAPE-HWSP scheme. For this scheme, the `429.mcf` benchmark has the maximum variation of ~6%, followed by ~5% deviation observed for `the 445.gobmk` benchmark. The lowest difference is noticed for the `458.sjeng` benchmark with ~2% power estimation accuracy improvement.

4.5. CAPE framework runtime

Fig. 12 presents the CAPE runtime and compares it against two 'corner' cases. We measure simulation runtime using `time` command in the Unix operating system. The 'gem5-Full BM' represents a typical benchmark characterization using the gem5 simulator. Using the

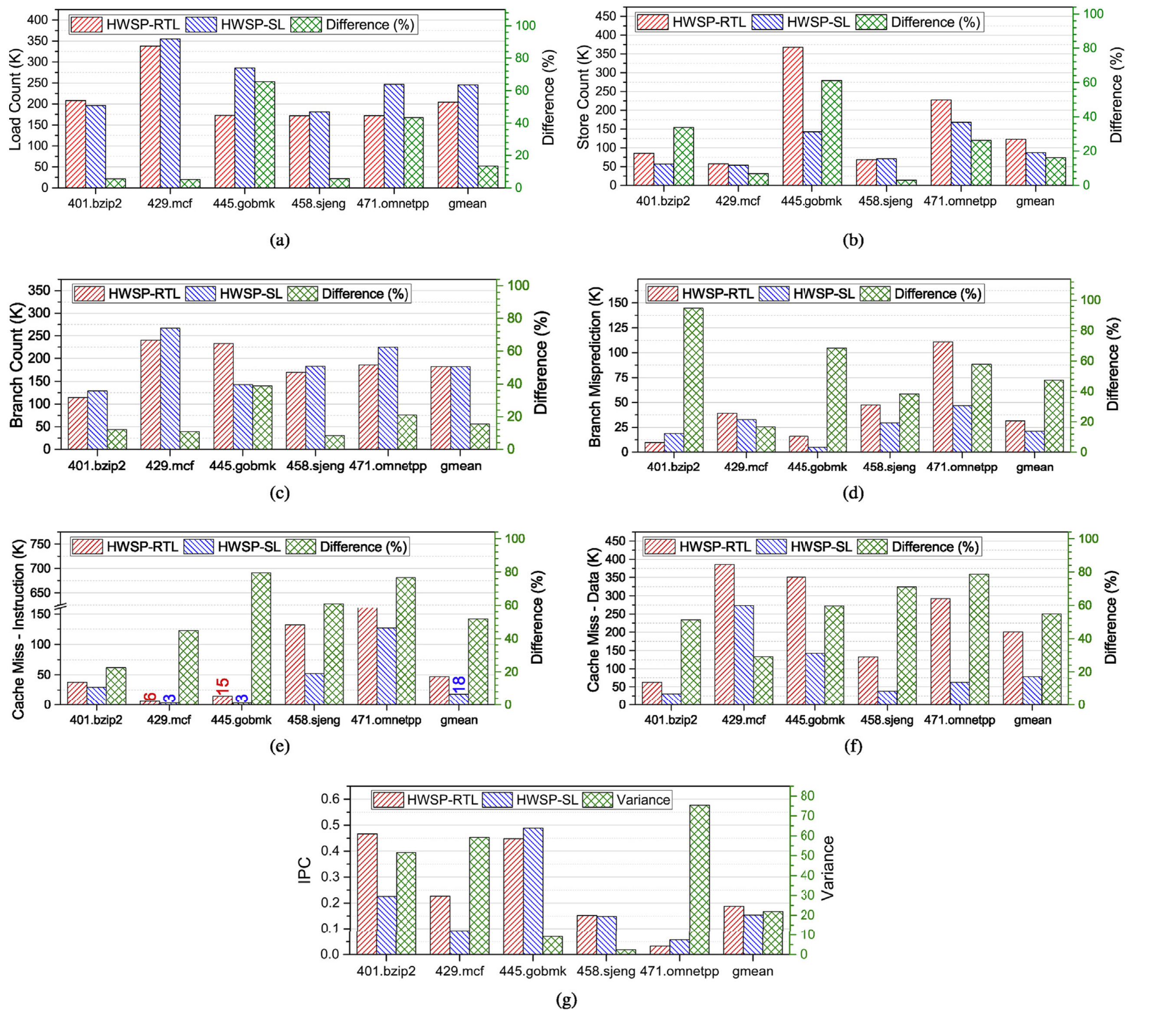


Fig. 9. Comparison of parameter profiling between gem5 and AnyCore RTL framework for the highest weighted simulation point. Our CAPE-HWSP scheme improves accuracy by enabling RTL (AnyCore) simulation for the critical segment of the workload application.

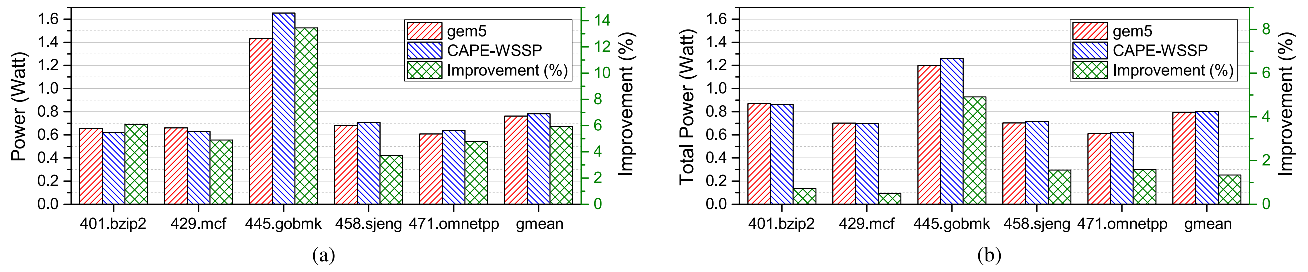
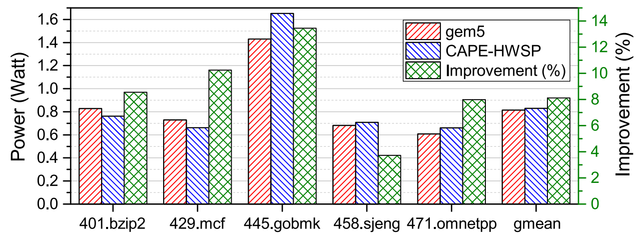


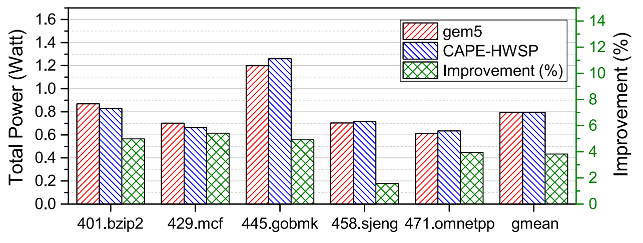
Fig. 10. Power estimation improvement using CAPE-WSSP scheme. (a) represents power improvement over individual simulation point. (b) represents overall power improvement for each benchmark using the CAPE-WSSP scheme.

detailed cpu model, the benchmark is simulated to completion without any simpoint representation. The 'gem5-SP' represents the average of 1 M instruction simulation time when all simpoints are simulated using the gem5 simulator. This simpoint-based benchmark simulation does not use any RT-level simulation. Finally, the 'HWSP' and 'WSSP' represent the RTL simulation time for 1 M instruction for the 'critical' simpoints in CAPE-HWSP and CAPE-WSSP scheme, respectively.

On average, the 'gem5-Full BM' approach takes 651 min to simulate five benchmarks. In contrast, the 'gem5-SP' method takes only 10 min on average to simulate all simpoints across five benchmarks. The 'HWSP' method spends an average of 31 min simulation runtime for 1 M instructions at the RT-level. For the 'WSSP' method, the average



(a)



(b)

Fig. 11. Power estimation improvement using CAPE-HWSP scheme. (a) represents highest weighted simulation point power improvement for each benchmark. (b) represents overall benchmark power estimation improvement using the CAPE-HWSP scheme.

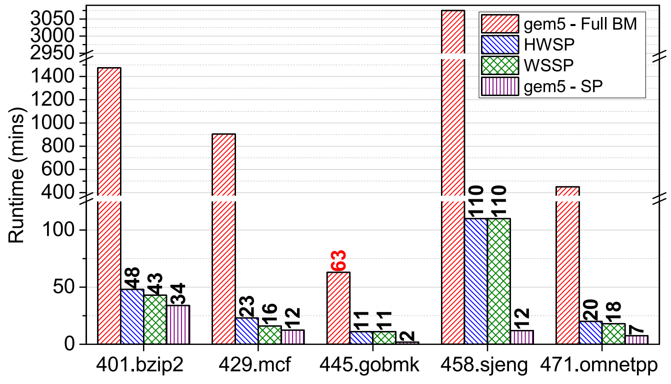


Fig. 12. Runtime comparison for different simulation methods. The CAPE framework is ~24× faster than gem5 full benchmark simulation.

runtime is 28 min for the same setting. Compared to the ‘gem5-Full BM’ approach, both ‘HWSP’ and ‘WSSP’ schemes are 21 times and 24 times (on average) faster, respectively. However, compared with the ‘gem5-SP’ approach, both the ‘HWSP’ and ‘WSSP’ approaches are ~3 times slower (on average). This increased runtime is the trade-off required to achieve the improved accuracy offered by the CAPE framework.

4.6. *Discussion*

4.6.1. *Variations in performance parameters*

Throughout this work, we incorporate the accurate RT-level information with the system-level simulation results to improve the overall workload profiling, and thereby improve power estimation accuracy. Figs. 8 and 9 show the detailed comparison of key performance parameters. The expected differences between the number of executed key performance parameters in gem5 and AnyCore framework are confirmed from the experimental results. While micro-architectural parameters have been matched both in AnyCore and gem5 models, the existing differences are caused by the simplified, generic design of key microarchitectural modules [5,26]. For example, the `fetch` stage in gem5 always aligns itself with the cache-line boundary for fetching new instructions. This results in two fetches when critical instructions cross over the cache-line boundary. Similarly, the default write-back depth in the gem5 `O3CPU` is 1, which generates write-back stalls for long-latency instructions and reduces the overall performance of the simulated design [6]. In order to improve the microarchitectural modeling in gem5, each design needs to be verified against real hardware architecture. However, at early design-space-exploration, this option is impractical and often impossible [1]. Nevertheless, system-level simulators (gem5) generally provide the flexibility of quick simulation with an acceptable level of accuracy, which in turn depends on various parameters such as: the underlying microarchitecture, validation against real hardware, runtime and performance parameter validation. Typically, system-level simulators can generate reasonably accurate microarchitecture performance information after multiple rounds of correcting the modeling, abstraction, and specification errors. As mentioned earlier, we address the specification errors in gem5 by matching the microarchitecture parameters and delays. It should be noted that we do not attempt to address the other errors in gem5 as that would skew the simulator towards a hardware specific design, which in turn nullifies the idea of leveraging a generalized system-level simulator during early design-space exploration. Instead, we leverage the state-of-the-art, expecting (and accepting) that there would be variations in data reported by gem5 and the AnyCore RTL.

4.6.2. *Accuracy improvement with CAPE framework*

The CAPE-WSSP scheme achieves an average of ~2% overall power estimation improvement across five simulated workloads. This nominal improvement is attributed to the fact that the CAPE-WSSP scheme does not consider simpoint weight when determining the ‘critical’ simpoint. Therefore, although the ‘critical’ simpoint results up to ~13% power estimation improvement (Fig. 10a: `445.gobmk`), when combined with respective simpoint weights, this high improvement is weighed down to ~5% (Fig. 10b: `445.gobmk`). The same is observed for rest of the workloads, thereby confirming that in the CAPE-WSSP scheme, the amount of overall power estimation accuracy is critically dependent on the ‘critical’ simpoint weight.

5. Related work

5.1. *gem5 simulator*

gem5 is a widely used system-level simulator for performance characterization, design modeling and design space exploration [3]. Fernando et al. used gem5 simulator to model both in-order and out-of-order arm microprocessors [5]. Their design modeled the microarchitectural details based on published and estimated data. Yang et al. extend gem5 to build a VLIW simulation platform [27]. They also modeled their design based on a cycle-accurate simulator and finally validated against the RTL simulator. We note that our scheme is different from the prior works because of the incorporation of RT-level information for accuracy improvement. Moreover, instead of running the full workloads, we leverage smart usage of SimPoint generated phases of the workload to reduce overall simulation time.

5.2. *SimPoint-based benchmark simulation*

Simpoint-based simulations create representative points/phases for a workload and simulate those points only. Maximilien et al. use Simpoint technique to profile benchmarks for different performance parameters and predict the performance of the benchmark [28]. Their model is solely dependent on the SimPoint accuracy and the hardware model used by the system-level simulator. Coskun et al. used the SimPoint tool to create a database for benchmarks and use that for dynamic thermal management [29]. However, their work did not include any RT-level data for improving accuracy.

5.3. Cross-layer simulation frameworks

The idea of multi-level simulation framework has been addressed by several research communities to tackle the issue of inaccurate simulation results. Huang et al. suggested the importance of hardware/software co-simulation techniques for improved performance accuracy but did not provide the necessary implementation details [9]. While Sanchez et al. proposed a microarchitectural simulator that can reduce the time for detailed simulation by leveraging dynamic binary translation for instruction driven timing models, their simulator utilizes system-level description of the design and lacks the accuracy of RT-level information [10]. On the other hand, Oboril et al. proposed a framework for simulating and modeling power/area at the microarchitectural level for exploring the impact of aging [30]. Their framework also relies on the limited accuracy of the system-level simulator in collecting performance parameters. Instead of using actual RT-level information, the authors introduce different aging models and utilize technology parameters and performance data generated by gem5. As a result, their work does not capture the hardware-level accuracy for performance/power characterization. The inaccuracy at the system-level simulation motivated Kim et al. to use FPGA based sampling technique for power accuracy improvement [11,12]. However, their work solely depends on the FPGA and RT-level design, which often time can be prohibitive for architecture designers for quick design space exploration. Walker et al. proposed a hardware-validated improved power modeling technique for the gem5 simulator [7]. Their technique depends on lengthy regression models for accuracy improvement. Moreover, their model requires actual hardware for the validation, which is prohibitive for early-stage design exploration. Furthermore, the model needs to be updated for every hardware, meaning lengthy simulation both at the hardware-level to create a database for the algorithm, and multiple simulations at system-level (gem5) for addressing all the modeling errors and finalizing an acceptable model for power estimation.

To the best of our knowledge, our proposed scheme is the first to introduce the ‘cross-layer’ approach by integrating system-level and RTL simulation results to increase the accuracy of power estimation, while maintaining reasonable fast simulation times overall.

6. Conclusion

The proposed CAPE framework enables accuracy improvement for microprocessor power estimation during early design stage exploration. The framework also allows user to selectively focus on specific microarchitectural module for power consumption using the CAPE-WSSP scheme. Utilizing the simpoint representation of the workload, the CAPE framework profile the ‘critical’ simpoint at the RT-level and rest at the system-level simulator. The parameters collected from the cross-layer profiling is used as input to the power simulator. Our evaluation results show that the proposed schemes can improve power estimation accuracy by more than 15% for individual simpoints, and by ~9% for full benchmark applications – compared to the existing system-level simulation based frameworks. The CAPE framework is ~3 times slower than the stand-alone system-level simulation of simpoints, while approximately 24 times faster than the conventional full benchmark simulation at system-level.

References

[1] A. Butko, R. Garibotti, L. Ost, G. Sassatelli, Accuracy evaluation of GEM5 simulator system, in: International Workshop on Reconfigurable and Communication-Centric Systems-On-Chip, ReCoSoC), 2012.

[2] J.L. Henning, SPEC CPU2006 benchmark descriptions, Comput. Architect. News 34 (4) (2006) 1–17.

[3] N. Binkert, B. Beckmann, G. Black, S.K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D.R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M.D. Hill, D.A. Wood, The gem5 Simulator, Comput. Architect. News 39 (2) (2011) 1–7.

[4] R.B.R. Chowdhury, A.K. Kannepalli, S. Ku, E. Rotenberg, AnyCore: a synthesizable RTL model for exploring and fabricating adaptive superscalar cores, in: International Symposium on Performance Analysis of Systems and Software (ISPASS), 2016.

[5] F.A. Endo, D. Courouss, H.P. Charles, Micro-architectural simulation of in-order and out-of-order ARM microprocessors with gem5, in: International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV), 2014.

[6] T. Nowatzki, J. Menon, C.-H. Ho, K. Sankaralingam, Architectural simulators considered harmful, IEEE Micro 35 (6) (2015) 4–12.

[7] M. Walker, S. Bischoff, S. Diestelhorst, G. Merrett, B. Al-Hashimi, Hardware-validated CPU performance and energy modelling, in: International Symposium on Performance Analysis of Systems and Software (ISPASS), 2018.

[8] B. Black, J.P. Shen, Calibration of microprocessor performance models, Computer 31 (5) (1998) 59–65.

[9] C.-Y.R. Huang, Y.-F. Yin, C.-J. Hsu, T.B. Huang, T.-M. Chang, SoC HW/SW verification and validation, in: Asia and South Pacific Design Automation Conference (ASP-DAC), 2011.

[10] D. Sanchez, C. Kozyrakis, ZSim: fast and accurate microarchitectural simulation of thousand-core systems, in: ACM SIGARCH Computer Architecture News, vol. 41, 2013, pp. 475–486.

[11] D. Kim, A. Izraelevitz, C. Celio, H. Kim, B. Zimmer, Y. Lee, J. Bachrach, K. Asanovicc, Strober: fast and accurate sample-based energy simulation for arbitrary RTL, in: International Symposium on Computer Architecture (ISCA), 2016.

[12] D. Kim, C. Celio, D. Biancolin, J. Bachrach, K. Asanovic, Evaluation of RISC-V RTL with FPGA-accelerated simulation, in: First Workshop on Computer Architecture Research with RISC-V, 2017.

[13] M. Zaman, M.M. Shihab, A.K. Coskun, Y. Makris, Towards a cross-layer framework for accurate power modeling of microprocessor designs, in: International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS), 2018.

[14] D. Burger, T.M. Austin, S. Bennett, Evaluating Future Microprocessors: the SimpleScalar Tool Set, University of Wisconsin-Madison, Computer Sciences Department, 1996.

[15] A. Waterman, Y. Lee, D.A. Patterson, K. Asanovi, The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Version 2.1, Tech. Rep. UCB/EECS-2016-118. EECS Department, University of California, Berkeley, May 2016http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-118.html.

[16] A. Roelke, M.R. Stan, Risc5: implementing the RISC-V ISA in gem5, in: First Workshop on Computer Architecture Research with RISC-V (CARRV), 2017.

[17] A.A. Nair, L.K. John, Simulation points for SPEC CPU 2006, in: International Conference on Computer Design (ICCD), 2008.

[18] K. Ganesan, D. Panwar, L.K. John, Generation, validation and analysis of SPEC CPU2006 simulation points based on branch, memory and TLB characteristics, in: SPEC Benchmark Workshop, 2009.

[19] G. Hamerly, E. Perelman, J. Lau, B. Calder, Simpoint 3.0: faster and more flexible program phase analysis, J. Instruct. Level Parallel. 7 (4) (2005) 1–28.

[20] E. Perelman, G. Hamerly, B. Calder, Picking statistically valid and early simulation points, in: International Conference on Parallel Architectures and Compilation Techniques (PACT), 2003.

[21] T. Sherwood, E. Perelman, G. Hamerly, B. Calder, Automatically characterizing large scale program behavior, Comput. Architect. News 30 (5) (2002) 45–57.

[22] S. Li, J.H. Ahn, R.D. Strong, J.B. Brockman, D.M. Tullsen, N.P. Jouppi, McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures, in: IEEE/ACM International Symposium on Microarchitecture (MICRO), 2009.

[23] Y. Nakamura, K. Hosokawa, I. Kuroda, K. Yoshikawa, T. Yoshimura, A fast hardware/software Co-verification method for system-on-a-chip by using a C/C simulator and FPGA emulator with shared register communication, in: Design Automation Conference (DAC), 2004.

[24] A wrapper for the SPEC CPU2006 benchmark suite. URL https://github.com/ccelio/Speckle.

[25] J.Y. Joshua, R. Sendag, D.J. Lilja, D.M. Hawkins, Speed versus accuracy trade-offs in microarchitectural simulations, IEEE Trans. Comput. 56 (11) (2007).

[26] J.H. Ahn, S. Li, O. Seongil, N.P. Jouppi, McSimA: a manycore simulator with application-level simulation and detailed microarchitecture modeling, in: International Symposium on Performance Analysis of Systems and Software (ISPASS), 2013.

[27] L. Yang, L. Wang, X. Zhang, D. Wang, An approach to build cycle accurate full system VLIW simulation platform, Simulat. Model. Pract. Theor. 67 (2016) 14–28.

[28] M.B. Breughe, S. Eyerman, L. Eeckhout, Mechanistic analytical modeling of superscalar in-order processor performance, Trans. Architect. Code Optim. (TACO) 11 (4) (2015) 50.

[29] A.K. Coskun, R. Strong, D.M. Tullsen, T. Simunic Rosing, Evaluating the impact of job scheduling and power management on processor lifetime for chip multiprocessors, in: ACM SIGMETRICS Performance Evaluation Review, 2009.

[30] F. Oboril, M.B. Tahoori, ExtraTime: modeling and analysis of wearout due to transistor aging at microarchitecture-level, in: International Conference on Dependable Systems and Networks (DSN), 2012.