Follower Core: A Model to Simulate Large Multicore SoCs

Tanuj Agarwal
AMD India Private Limited
Bangalore, India
Tanuj.Agarwal@amd.com

Bill Jones
Advanced Micro Devices, Inc.
Fort Collins, USA
Bill.Jones@amd.com

Anasua Bhowmik
AMD India Private Limited
Bangalore, India
Anasua.Bhowmik@amd.com

ABSTRACT

Cycle accurate simulator is a critical tool for processor design and as the complexity and the core count of the processor increase, the simulation becomes extremely time and resource consuming and hence not very practical. Accurate multi-core performance estimation in realistic time is needed for making the right design choices and make high quality performance projections.

In this work we present a multi-core simulation model called **Follower Core**, that helps us to approximate the multi-core simulations by simulating some cores in detail and abstracting out the other cores without reducing the overall activities at the shared resources. This enables us to simulate all the critical shared resources in the multi-core system accurately and hence the detailed core can provide correct performance estimation. The approach is applied over existing simulation models and it reduces the simulation time significantly, especially for long running workloads. The 'Follower Core' model provides an average speed up of 3x compared to baseline and is an accurate approximation of detailed multi-core simulations with a maximum error of 2% with the baseline model and extends our capabilities by improving our coverage and providing flexibilities to run mixed workloads.

CCS CONCEPTS

• **Computing methodologies~Modeling and simulation**
• Computer systems organization~Multicore architectures

KEYWORDS

Follower Core; Heterogeneous Follower Core; Baseline Simulation Methodology

ACM Reference format:

Tanuj Agarwal, Bill Jones, Anasua Bhowmik. 2019. Follower Core: A Model to Simulate Large Multi-core SoCs. In *Proceedings of 10th ACM/SPEC International Conference on Performance Engineering (ICPE' 19), Mumbai, India, April 7-11, 2019.* ACM, New York, NY, USA 8 Pages, https://doi.org/10.1145/3297663.3309678

1 INTRODUCTION

Cycle accurate simulators play a critical role in processor design by estimating the performance of the proposed hardware. With every generation, the processor cores are becoming more complex and more cores are added in the processor die to use the available transistors more efficiently and run more threads or applications in parallel. With the increasing complexity of the core architecture, the simulator complexity and simulation time is also increasing proportionally. Simulating a modern-day processor with many such cores becomes challenging and prohibitively time consuming. The simulation time can increase from hours for a single core simulation to several days for a four-core or eight-core system. Figure 1 demonstrates the increase in run time for different simulation models for SPECrate®2017_int_base[1] [1] Integer suite. We consider a micro-architecture similar to AMD "Zen" [3] architecture where each core has private L1 and L2 caches and they all share the same L3 cache. The increasing run-time is a real challenge in the design of upcoming features as the turnaround time for analysis and debug increases.

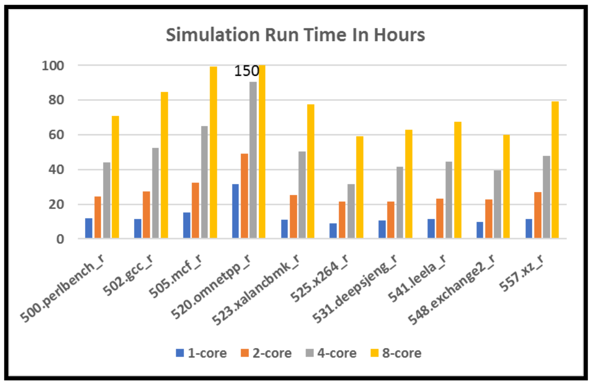

Figure 1: Simulation time of SPECrate®2017_int_base applications with varying number of cores.

[1] SPEC, SPEC CPU, SPECint, SPECfp, and SPECrate are registered trademarks of the Standard Performance Evaluation Corporation. See www.spec.org for more information

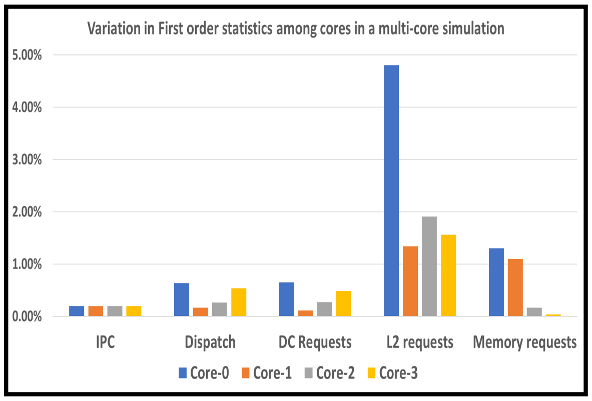

Figure 2: Core level first order statistics normalized to average simulation values.

An accurate single core simulation cannot estimate the performance of a multi-core processor even when copies of same application are run on all cores, since the multi-core processor performance is heavily dependent on the usage of shared resources like last level cache, memory system and bandwidth. We concentrate on the 'RATE' behavior where all cores run the same benchmark with unique address spaces and no data sharing between the cores. Figure 2 shows the absolute percentage variation of some important first order statistics from the corresponding average value for a 4-core simulation of the SPECrate®2017_int_base [1] and SPECint®_rate_base2006 [2] binaries. We chose the statistics to be IPC (instructions committed per cycle), instruction dispatch, data cache, private L2, shared LLC and memory requests. It can be observed that the variation of these statistics among cores are very small implying that the cores behave similarly**.** Hence it is possible to infer the behavior of all cores by studying the behavior of one core when the same application is run on all cores. Therefore, if we can model the contention and interference at the shared resources by abstracting out a few cores, then we can reduce the multi-core simulation time and complexity significantly without sacrificing the accuracy of the performance projection of the full system.

We propose a multi-core simulation methodology, 'Follower Core' that can be implemented in an existing cycle accurate simulator model by simulating a subset of cores in detail and abstracting out the others. The contributions of this work are following,

1. The 'Follower Core' simulation methodology helps us to get a significant reduction in simulation time for multi-core simulation.
2. It has a good correlation with the detailed simulation model.
3. We propose a 'Heterogenous Follower Core' model which helps us to provide a good correlation as we scale the number of cores. It also provides us the flexibility to run different applications together on a multi-core system where the applications have different characteristics.

The rest of the paper is organized as follows. Section 2 deals with relevant work and highlights the previous efforts to speed up multi-core simulation. Section 3 discusses our proposed 'Follower Core' model in detail and the associated challenges. We provide the experimental setup and benchmarks used for evaluation in Section 4 and present the results in Section 5. Section 6 discusses the additional advantages of our methodology and Section 7 concludes our work.

2 RELEVANT WORK

Several approximation techniques have been proposed for multi-core performance projection whose accuracy varies with hardware architecture and workloads. Hassan *et al.* [4]propose a synthetic trace-based cache only simulation model using the adaptation of Least Recently Used Stack Model (LRUSM) [5] techniques. This catches the temporal and spatial locality patterns and generate phases to simulate a cache hierarchy and are further reduced using the two state Markov chain model. While this technique is efficient and provides a significant reduction in runtime, we observe several shortcomings to this approach as well. The cache traffic is susceptible to changes in the core and hence collecting cache traces can be a non-trivial task. For example, changing the front-end features like branch predictors can impact the timings of cache traces. Any changes to the core may require changing traces as well, which would require efficient synchronization between teams and may lead to ambiguity in results. This method is effective only when the core performance is deterministic and smaller traces can be used to verify the specific scenarios in the cache.

Kanaujia *et al.* [6] proposed a simulation model, Fast MP, where they try to exploit the RATE behavior or the homogeneity of the workloads. All cores show a similar behavior for SPEC applications, hence simulation results of one is like all the others. They propose a model where they simulate one real core in detail and use abstract cores for the rest of simulation. The abstract cores copy the memory traffic from the real core and send it to memory to generate realistic memory traffic for the complete multi-core system. They have studied their model on the SPEC CPU 2000 [9] Integer suite over a multi-core machine. They have showed that the model has a reasonable correlation for a two and four core machines for most benchmarks. However, the correlation becomes significantly worse (max miscorrelation 38%) as they increase the number of cores to 8. They have used a large private L2 cache (8MB per core) which reduces the overall traffic at shared memory resources.

Our 'Follower Core' model is based on the similar principle as the Fast MP base model. Instead of large private L2 we have used much smaller private L2 (512KB) and a larger shared L3. Although our model receives significantly more shared LLC accesses compared to Fast MP configuration, it shows highly accurate correlation for the contemporary SPECrate®2017_int_base and SPECint®_rate_base2006 [1,2] benchmarks. We have also studied combination of workloads where benchmarks behaviors vary across cores. The details of which are provided in the later section.

3 FOLLOWER CORE MODEL

In this section we discuss the details of the 'Follower Core' model where a multi-core system is simulated without simulating all the cores in details. As observed in Figure 2, when

the same application is run on different cores at the same time, the cores have a similar behavior and they generate similar cache and memory traffic. This is the base principle of 'Follower Core' methodology. In this model, one or few cores are simulated with full details and the L2 cache traffic generated by the detailed core(s) is replicated to the cache hierarchies of the other cores. To maintain an equivalence with the detailed multi-core simulation (referred as Baseline), we need to take care that,
- Address spaces for all cores are exclusive. This helps us to eliminate any false sharing and maintain the RATE behavior.
- The phase behavior between cores is similar to the Baseline model.

Figure 3 shows the block diagram of the implementation of 'Follower Core' in a 4 core CPU model where all the cores share last level L3 cache (LLC). In this model one core is simulated in complete detail which we call as 'main core'. The cache hierarchies of other cores follow the main core and they see the traffic being generated from a dummy core, hence termed as 'Follower Core(s)'. Detailed model for private L2 caches, shared LLC, interconnect(s) and memory system is included in the model without any changes. In the setup shown in Figure 3, we have used a single main core and 3 follower cores to model a 4-core system.

The main core simulates the complete front-end pipeline with all stages. We chose the L1-L2 boundary as the interface for replicating and generating the traffic at follower cores. As seen in Figure 2, the behavior of primary L1 cache (DC/IC cache) is uniform across the cores and thus incorporating it in our model does not provide with any added simulation benefits. Therefore, the first level primary caches are considered part of detailed main core in this work. Selecting the shared LLC boundary as the boundary for follower core traffic generation speeds up our simulation but abstracts out the phase level variation encountered at the shared cache in a detailed simulation. We observe that for memory intensive benchmarks, performance of private L2 cache can be sensitive to shared LLC and memory latency and thus the L1-L2 cache boundary becomes the interface for follower cores.

The L1 misses generated by the main core are stored in the follower core(s) buffers which are then processed to generate the traffic at their L1-L2 interface. These buffers are implemented

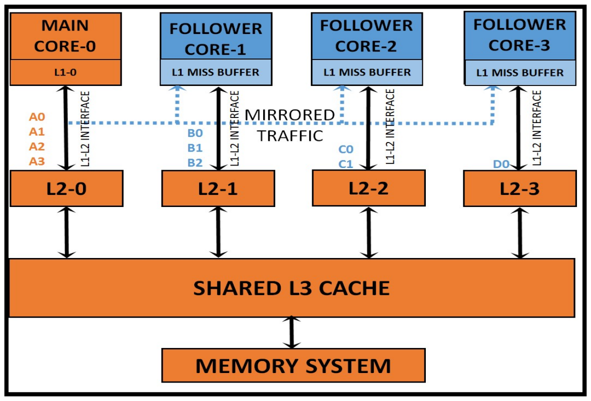

Figure 3: Follower Core Implementation.

through linked lists and follow the strict FIFO order. Apart from the address, the follower core buffers store the critical information for each transaction like the type of request (Data/Instruction/Prefetch), time it was issued, logical core number (in case of simultaneous multi-threading) etc. While these addresses are processed and sent to their respective L2 caches the follower cores considers all this information for converting into appropriate requests and tracking them. The L1-L2 interface boundary at the follower core is modeled similarly to the real core maintaining all the resource and design constraints. This makes sure that the follower cores generate the L2 traffic similar to the Baseline and do not flood their L2 caches and shared resources with premature requests.

This setup helps us to generate a full system traffic with a high degree of accuracy. The model preserves the performance dependence between different CPU modules. The shared resources see the traffic as if it was generated by all real cores and are simulated accordingly. Simultaneously, the main core's performance is also dependent on the shared resource characteristics and memory bandwidth and latency. Any change in one is reflected over other as in a detailed Baseline model. For examples, prefetchers can be extremely sensitive to both core/engine and memory latency. The follower core takes care that the prefetcher behavior is captured accurately taking both into account. Similarly, any dependency of the fetch or branch prediction on the cache/memory latency, is reflected accurately as well.

We need to ensure proper modeling of buffers and address conversion schemes. Maintaining each core has an exclusive address space and address conversion in that space eliminates any false sharing behavior. Similarly, improper address conversion can lead to artificial conflicts. For example, if the cache requests are generated to same LLC index then we can see thrashing of some indexes while others are being unused leading to higher LLC miss rates. Therefore, choosing a proper address conversion algorithm is critical to minimize such scenarios. In our model, address conversion is done using specific masks which are generated after a detailed analysis of workload behavior. It maintains that the traffic from each core is within its address space and avoids any false conflicts.

Correlating the phase behavior between the detailed Baseline model and 'Follower Core' is a challenging task. Phase behavior is mostly observed in memory intensive workloads. In a detailed baseline simulation of memory intensive workloads the core performance is heavily dependent on the performance of memory system. The contention at shared cache, interconnects and memory is high, which can cause the execution speed of cores to differ as they can observe different memory latencies at different stages of program. The 'Follower Core' is a highly synchronous model where all the follower cores have a fixed delay with respect to main core. As a result, the traffic from the follower core(s) will be injected after a specific duration, therefore maintaining a constant phase at the L1-L2 boundary. To counter the problem, we make the follower core delays to be configurable, that is, depending on the benchmark behavior we can tune the phase between detailed main core and follower cores. This helps us to approach the detailed Baseline behavior with a reasonable degree of accuracy.

3.1 Heterogenous Follower Core Model

As we scale the multi-core simulation model to higher number of cores, we find that the phase difference between cores increases and becomes unpredictable with some cores leading in some phases and vice-versa. Introducing delays in the follower cores do not capture this behavior properly as the phase differences between the cores are not uniform and predicatable. We propose a 'Heterogenous Follower Core' model which simulates more than one main cores in detail to capture the randomness originated in such scenarios. Figure 4 shows a 'Heterogenous Follower Core' setup which consists of two main cores and two follower cores. Each main core drives a follower core. If there are differences originating due to the performance of memory system, it would be reflected at the detailed core and they will start to exhibit a phase difference with each other. The follower core(s) that replicate the traffic will also incorporate these phases and will be much more accurate compared to a single main core model. In fact, this modelling scheme can capture the phase variations quite efficiently and reduces the simulation error seen among its predecessor scheme. While the model is slower compared to its predecessor, the speedup is still significant, saving us days for extensively long simulations. The model simulates larger multi-core systems more accurately and also enables us to simulate hetergenous workloads environment where applications with different characteristics are run together in the multicore system.

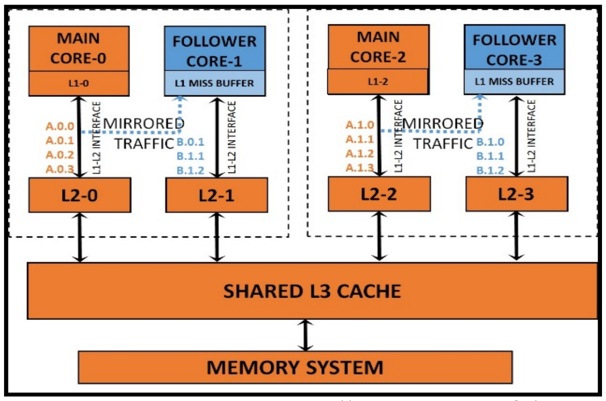

Figure 4: Heterogenous Follower Core Model.

Table 1. Micro-architectural Configuration

Core Organization	4 core, 2 way SMT
Primary Cache (Per Core)	Split, Data cache 32KB, Instruction cache 64KB
Secondary Cache (Per Core)	8 way, 512K Private Cache
Last level cache	16 way, 8MB, shared among 4-cores
Memory bandwidth	8 DDR channels, 2400 MHz

4 EXPERIMENTAL SETUP

We present our results using our proprietary in-house cycle accurate trace-based simulator that implements all details of the core, cache hierarchy and memory system. We simulate a four-core CPU architecture inspired from AMD "Zen" architecture [3] and micro-architecture details are summarized in Table 1.

We simulate the state of art SPEC binaries, SPECrate®2017_int_base [1] and SPECint®_rate_base2006 [2]. These binaries have an efficient mix of compute and memory intensive benchmarks that simulate and analyze all the critical components of any hardware design. The SPECrate®2017_int_base [1] binaries are 'aocc' [7] compiled and the SPECint®_rate_base2006 [2] binaries are 'gcc' compiled using the '-O2' flag.

5 RESULTS

In this section we compare the 'Follower Core' model against the detailed Baseline model which simulates all the real cores. We evaluate the 'Follower Core' simulation model for simulation 'Accuracy' and 'Speedup'.

5.1 Accuracy

The most important aspect of any approximation techniques is simulation accuracy which we described as the error between the baseline full system simulation and the 'Follower Core' models. The error measures the correlation between the two models and ideally, we want an error close to 0%.

5.1.1 IPC (Instructions Committed Per Cycle)

For any performance projection, IPC is the most important metric and we begin by matching the IPC for two simulation models. Figure 5 shows the error in IPC for 'Follower Core'

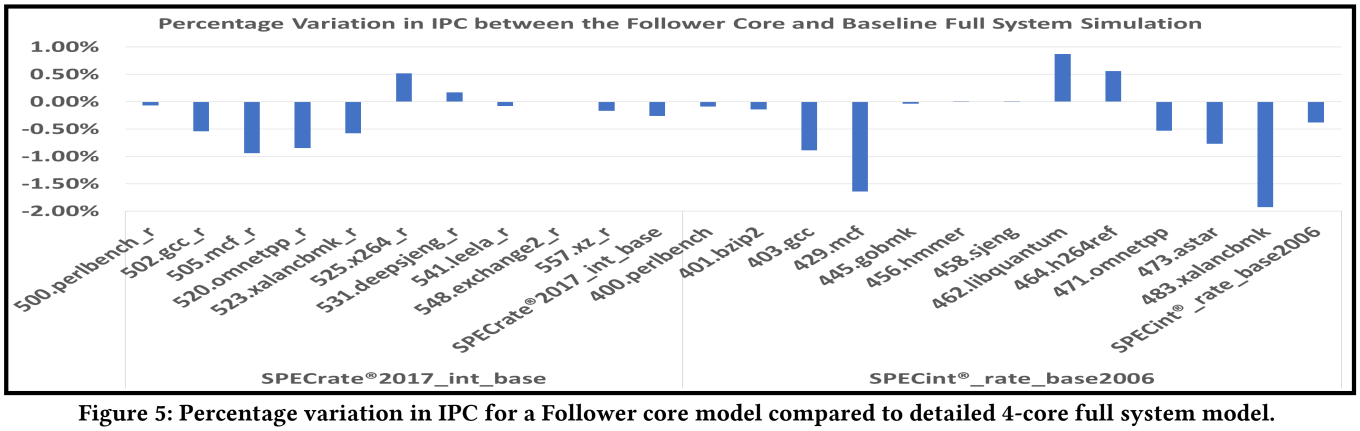

Figure 5: Percentage variation in IPC for a Follower core model compared to detailed 4-core full system model.

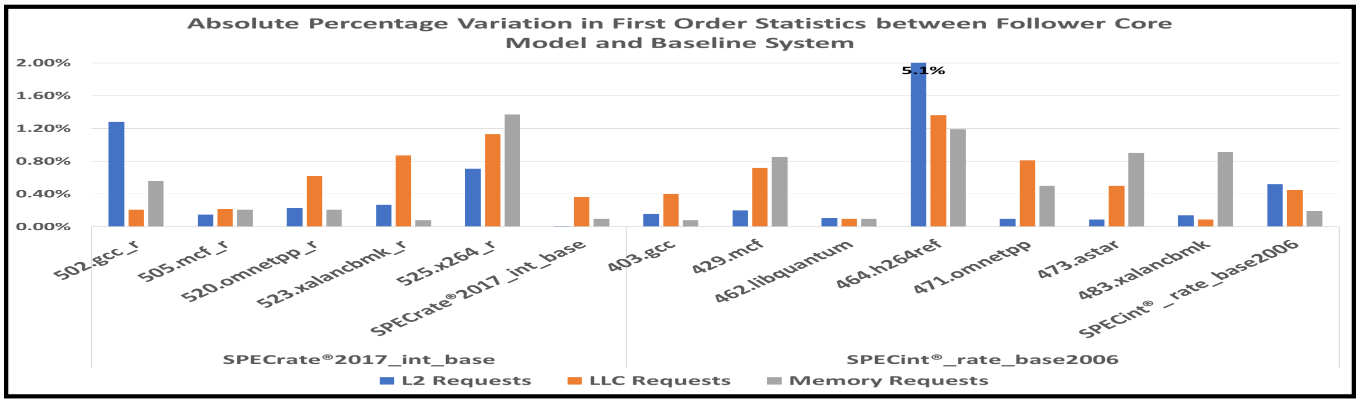

Figure 6: Percentage variation in First order statistics between Follower Core Model and Baseline System.

model, compared to the baseline model.

The IPC reported by the 'Follower Core' model matches closely with the all real core simulation model. Overall, the SPECrate®2017_int_base and SPECint®_rate_base2006 benchmarks have an error margin of 0.26% and 0.38% respectively. The compute intensive benchmarks have a better correlation than the memory intensive benchmarks. Most benchmarks have less than 1.5% miscorrelation in IPC. The memory intensive benchmarks like '429.mcf' and '483.xalancbmk' show a higher IPC miscorrelation due to the difference in phase behavior between the all real cores and the 'Follower Core' models. The IPC miscorrelation of '429.mcf' and '483.xalancbmk' from SPECint®_rate_base2006 suite can be reduced by increasing delays between the main core and follower cores. Apart from delays, the simulation accuracy can also be increased by using a 'Heterogenous Follower Core' model having two main cores as shown later.

5.1.2 Other first order statistics

We now investigate the difference with respect to some other important first order statistics to show the correlation between the 'Follower Core' and Baseline models. We present the difference in requests per thousand instructions (pti) for L2 cache, Last level cache and memory system.

Figure 6 shows the percentage difference in each first order statistic between the 'Follower Core' model and Baseline. We focus on benchmarks that have an absolute IPC error greater than 0.5%. From Figure 6 we find that most of the benchmarks show a negligible variation except '464.h264ref' from the SPECint®_rate_base2006 suite which has a very high number of L2 requests. The LLC requests and request to memory system (LLC misses) also match quite closely between the two models, with an absolute maximum difference less than 1.5%. The difference in memory latency cycles for LLC misses is a good representative of the difference of the phase between the two models as the overall statistic for cache and memory requests also matches closely with an average difference of around 2% and a maximum difference of 7%. We have also observed that the memory intensive benchmarks have a higher memory latency difference compared to compute intensive benchmarks.

5.2 Speedup

The main objective of the 'Follower Core' model is to achieve significant reduction in simulation time compared to all core model. Figure 7 shows the speedup of 'Follower Core' model over the baseline all core simulation. Overall, we see a 3x speedup for a 4-core model, with the maximum speedup of 4.5x and a minimum speedup of 2x. As we increase the number of cores to eight, we observe that an average speed-up of 4x, with a maximum speed-up of 7x. Similar trends are observed for SPECint®_rate_base2006 suite. Since the 'Follower Core' model saves the simulation time by reducing the core simulation time, we find that the compute intensive benchmarks have a higher speedup compared to memory intensive benchmarks. The memory intensive benchmarks spend a considerable time at the

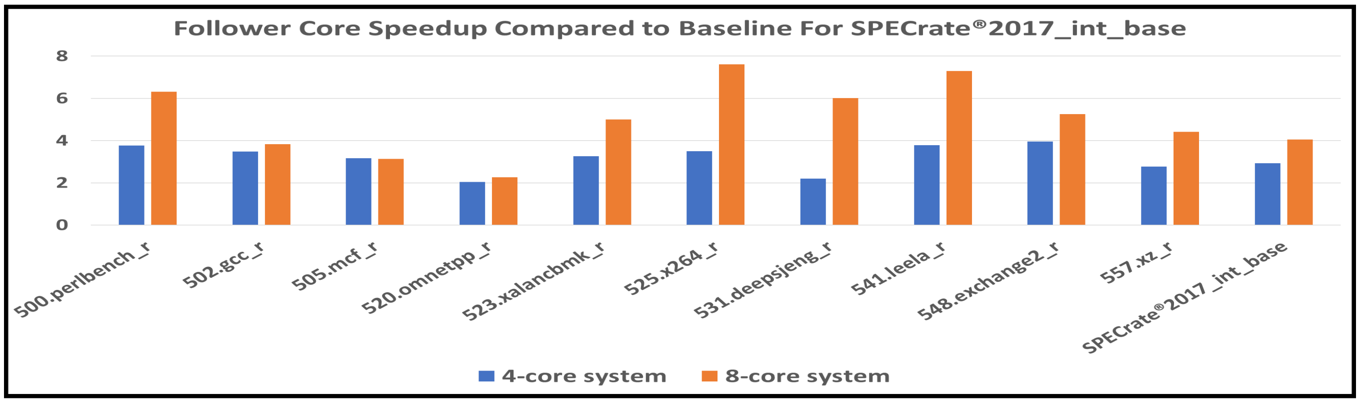

Figure 7: Simulation Speedup for Follower Core Model.

LLC and memory system apart from cores, that is why we see that the savings is low. For some of the memory intensive benchmarks, we see that the simulation time is order of days, thus a speed-up of 2-3x also saves a minimum simulation time of 1 day per simulation, which improves the turnaround time for analyzing and fixing simulation bugs and issues considerably.

5.3 Scalability

We now analyze the performance of 'Follower Core' model as we increase the number of cores. For the eight-core system we have assumed that four cores share the last level cache and therefore there are two instances of LLCs both connected to the same memory system. In the previous section, we observed that the speed-up increases as we increase the number of cores.

However, the simulation accuracy of the 'Follower Core' model for an 8-core system drops compared to the four-core system as shown in Figure 8. We observe that there is a significant IPC miscorrelation between Follower Core and a detailed eight-core simulation model especially for memory intensive benchmarks like "505.mcf_r" and "520.omnetpp_r". This is due to the phase difference exhibited by the cores in a full system simulation which also increases as we simulate more cores. As a result, there is a mismatch at the shared resources causing miscorrelation between the two models.

To reduce the miscorrelation, we use a heterogenous 'Follower Core' model as described in Section 3. The model is able to capture the phase variations with the help of two main cores and reduces the miscorrelation between the 'Follower Core' and Baseline model. Since the 'Heterogenous Follower Core' model simulates two detailed cores, the simulation time increases reducing the gains achieved in simulation speedup. However, we are still able to achieve a minimum speedup of 2x and reduce the error margins to a maximum of 2.5% for "505.mcf_r" and "520.omnetpp_r" as shown in Figure 8 and Figure 9.

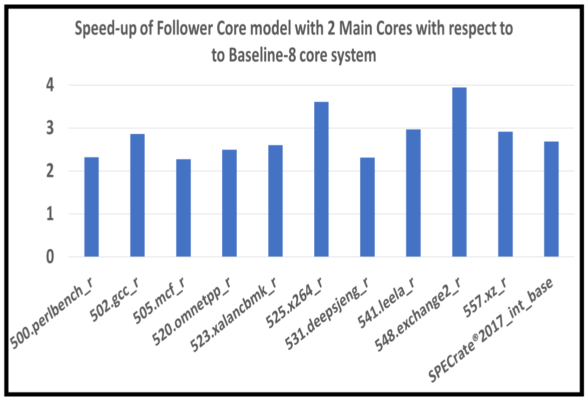

Figure 9: Speed-up of Heterogenous Follower Core Model compared to Baseline 8-core system.

6 ADDITIONAL ADVANTAGES OF FOLLOWER CORE MODEL

The section discusses the additional benefits and extended capabilities of the 'Follower Core' model besides providing fast and accurate simulation.

6.1 Simulation Coverage

As the hardware designs become more and more complex, tracing a workload with higher thread counts becomes challenging. The process is complex, cumbersome, time consuming and requires a critical validation to obtain an accurate representation of a workload. Lahiri *et al.* [8] analyze the tracing process of any new workload that must be added to existing repertoire of workload traces. They describe the steps and hurdles associated to trace any new workload which can be critical to any new hardware architecture.

The 'Follower Core' model provides us with the flexibility to run higher thread counts even when traces for high thread count

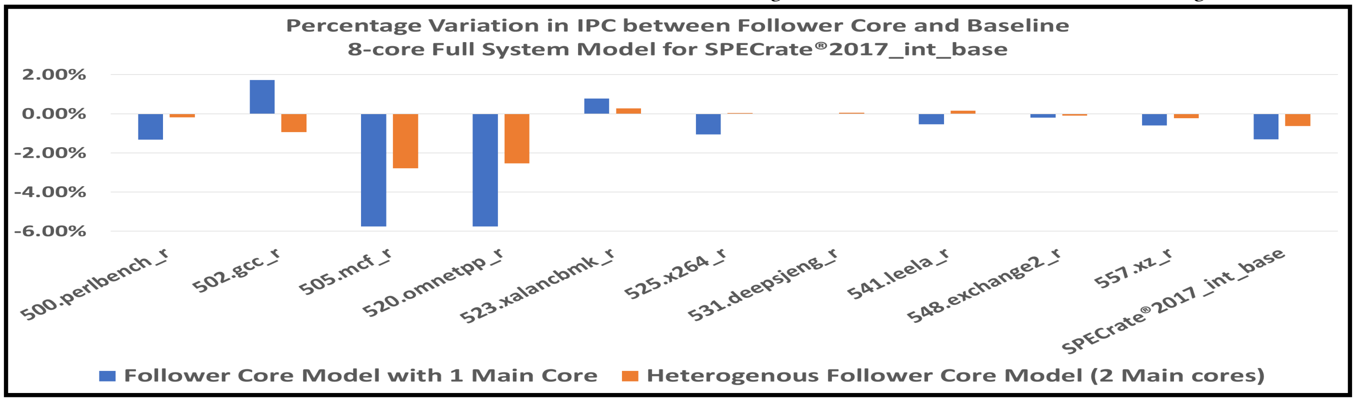

Figure 8: Percentage Variation in IPC for a Follower core model compared to Baseline 8-core system.

Table 2. Categorization of Mixed Workload and the Variation In Performance Between Benchmarks

Category	Type		B1:B2			Benchmark Characteristics	
	B1	B2	IPC	LLC Request	LLC Misses	Type	Bandwidth Requirement
Category-1	High	High	1.3	1.5	1.4	High	>2GBps
Category-2	High	Medium	2	2	3	Medium	0.5-1.5
Category-3	High	Low	2.7	5	8	Low	<0.5GBps

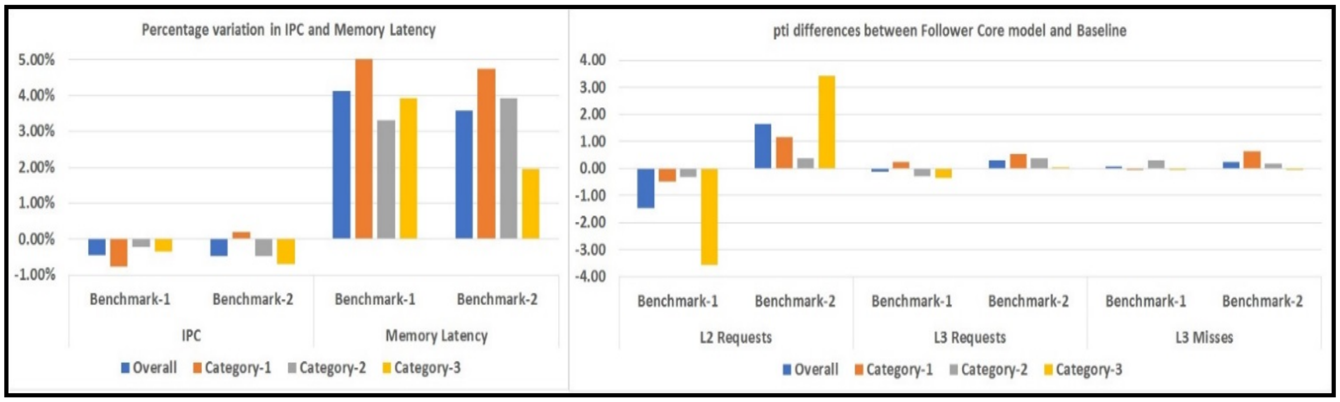

Figure 10: Correlation between the Heterogenous Follower Core and Baseline model for a mixed workload suite.

are not available for some workloads. This helps us to provide the accurate performance projections for these workloads.

6.2 Mixed Workloads Simulation Capabilities

Contemporary datacenters and cloud-based environment run different benchmarks on the same system simultaneously. While analyzing the performance of any upcoming design feature, it would be good to see its capabilities on such mixed -workload scenarios. The 'Heterogeneous Follower Core' model helps us simulate these scenarios accurately. The 'Heterogenous Follower Core' can execute multiple detailed cores and provides us with the capabilities to run two or more different benchmarks at the same time. The follower cores can replicate the corresponding main cores and hence the shared resources can see realistic scenarios, where they are constrained by different requirements. Such arrangements help us to analyze the critical resources like shared last level cache or memory system in greater detail and help us to find solutions that can be useful to optimize performance in such situations. These capabilities help to analyze the bottlenecks that can be found at later stages of design and facilitates a superior design process. We profile the SPECrate®2017_int_base binaries according to their micro-architectural characteristics. We emphasize over the shared resource parameters, like LLC accesses, LLC misses and memory bandwidth utilization to categorize them as high, medium and low to indicate the pressure over shared resources. We mix different binaries for all possible combinations and try to replicate scenarios where individual benchmarks have a different shared resource requirement. The requirements can be conflicting, where we see one benchmark hurting the performance of other or supporting where the performance of both benchmarks improves. Table 2 shows the categorization of individual benchmarks (B1 and B2) and the broad categories obtained by mixing benchmarks. The average variation in the characteristics of the two benchmarks in each category is provided in Table 2. As we can see as the benchmarks have similar characteristics the performance and shared resource requirements are similar. We mix two benchmarks and replicate their copies over the four-core system in equal proportions. This helps us to replicate a contemporary behavior where the benchmarks can affect the performance of one another.

Figure 10 represents the correlation between the a 'Heterogenous Follower Core' simulation compared to a four-core baseline system running a mixed workload suite. The different benchmarks in each combination is run on a separate core. The 'Follower Core' model matches the baseline system quite closely for the mixed workloads with an error margin of less than 1% for IPC. The first order statistics are also similar to Baseline model which means that the 'Follower Core' model is able to capture and preserve the variations of mixed workloads effectively. The speedup observed for a 'Heterogenous Follower Core' model is 2x compared to Baseline.

7 CONCLUSION

'Follower Core' is an excellent simulation methodology that can approximate the multi-core simulations accurately and provide a significant speed-up compared to detailed model. It is a generic methodology that sits on top of any existing simulation framework and can be easily incorporated to generate accurate multi-core simulation results. Our results show that 'Follower Core' is able to capture the phase level variations seen in the detailed simulation effectively and helps to improve pre-silicon projection results through a quick turn-around time. It also helps us in increasing our simulation coverage. The 'Heterogenous Follower Core' improves the model further by reducing error margins as we scale the number of cores. It provides us with a framework that can run contemporary mixed workloads with a high degree of accuracy.

We evaluated our framework with RATE kind of behavior however the model can be extended to simulate data sharing applications as well. Our experiments show that the 'Follower Core' model is fast and accurate till eight cores. This model is highly scalable, and it is capable of simulating beyond eight cores as well.

REFERENCES

[1] SPECrate®2017_int_base benchmarks, https://www.spec.org/cpu2017/
[2] SPECint®_rate_base2006 benchmarks, https://www.spec.org/cpu2006/

[3] Michael Clarke, 2016. A new x86 core architecture for next generation of computing, Hot Chips: A Symposium on High Performance Chips
[4] Hassan Rahman, 2007. Synthetic Trace-Driven Simulation of Cache Memory, Advanced Information Networking and Applications Workshops, AINAW'07
[5] R. L. Mattson, 1970. Evaluation Techniques for Storage Hierarchies, IBM System Journal
[6] Shobhit Kanujia, 2006. Fast MP: A Multi-core simulation methodology, Workshop on Modeling, Benchmarking and Simulation, Boston, Massachusetts
[7] The AOCC Compiler, https://developer.amd.com/amd-aocc/
[8] Kanishka Lahiri, 2017. Fast IPC estimation for performance projections using proxy suites and decision trees, IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)
[9] SPEC CPU 2000 Integer benchmarks, https://www.spec.org/cpu2000/