Scale-Model Simulation

Wenjie Liu, Wim Heirman, Stijn Eyerman, Shoaib Akram, and Lieven Eeckhout

Abstract—Computer architects extensively use simulation to steer future processor development and research. Simulating large-scale multicore processors is extremely time-consuming and is sometimes impossible because of simulation infrastructure limitations. This paper proposes scale-model simulation, a novel methodology to predict large-scale multicore system performance. Scale-model simulation first constructs and simulates a scale model of the target system with reduced core count and shared resources. Target system performance is then predicted through machine-learning (ML) based extrapolation. Configuring the scale model (i.e., changing core count while proportionally scaling the shared resources) enables trading off accuracy versus simulation speed. For a 32-core target system running multiprogram workloads, configuring the scale model for the highest simulation speedup of 28× yields an average absolute prediction error of 6%. Configuring the scale model for highest accuracy brings down the prediction error to 2.7%, while still delivering a 2.6× simulation speedup.

1 Introduction

Predicting performance for a future computer system is a challenging and critical problem. The traditional approach is to employ detailed architectural simulation. Unfortunately, simulation is extremely time-consuming. In addition, simulation infrastructures have their limitations and may not be able to simulate a future large-scale system because of excessive memory consumption or insufficient compute capabilities in the simulation host system when simulating large numbers of cores. Researchers and practitioners employ a variety of techniques to tackle the simulation challenge. A widely used solution is sampled simulation [1], [2]. Unfortunately, this approach does not solve the simulation problem when it comes to simulating increasingly large target systems. In particular, we find that simulating an 8-core, 16-core and 32-core target system using Sniper [3], a fast and state-of-the-art parallel multicore simulator, takes 8, 17 and 43 hours, respectively, on a powerful 36-core simulation host when running multiprogram SPEC CPU workloads with (only) one billion instructions per benchmark. The super-linear increase in simulation time and complexity as a function of system size is a major challenge for computer architects in academia and industry.

In this paper, we propose *scale-model simulation*, a novel paradigm to predict future system performance. Scale-model simulation combines architectural simulation with machine learning to predict performance for large-scale systems based on detailed simulation of a scaled-down configuration of the target system, called the *scale model*. Scale model simulation first simulates a scale model of the target system. Performance for the target system is then predicted through extrapolation. Scale models solve the two problems aforementioned: (1) scale models speed up the simulation of large-scale systems: scale models are small enough to simulate in reasonable amount of time while performance extrapolation is instantaneous; and (2) scale models make simulation feasible for large-scale systems that cannot be simulated on existing infrastructure because of limitations in memory and compute capacity.

Scale models are widely used in a variety of engineering disciplines, including civil engineering (e.g., construction, fluid dynamics), mechanical engineering (e.g., aerodynamics, engine design), construction (e.g., architectural design, city development), etc. The most familiar scale models are miniatures, i.e., scaled-down versions of an original object. A key property of a scale model is that it accurately maintains relationships between various important aspects, but not necessarily all aspects, of the original object. Scale models enable demonstrating or studying some behavior of the original object. To the best of our knowledge, scale models have not been applied to the field of general-purpose computer architecture. While building an exact miniature of a target system may be hard in the context of processor architectures, if at all possible, we leverage the idea of scale models to predict future computer system performance.

The scale-model simulation paradigm can be decomposed into two sub-objectives: (1) scale-model construction and (2) scale-model extrapolation. The first objective relates to how to construct a scale model of a (much) larger target system. A scale model should take substantially less time to simulate than the target system, yet it should enable an accurate prediction of the performance of the large-scale target system. This is not a simple endeavor for general-purpose multi-core processors because of resource contention in the shared caches, the network-on-chip (NoC), main memory, etc. Because the scale model is not an exact miniature of the target system, the second objective relates to how to extrapolate performance from the scale model to the target system. Shared resources lead to a variety of complex interactions at the system level, which the scale models may or may not capture to a sufficient degree. Scale-model extrapolation is employed to predict the impact of contention effects in shared resources on target-system performance based on the simulated scale model.

To address the first objective, we explore how to construct scale models and we find that proportionally down-scaling the shared resources with system size is effective. In particular, using multiprogram SPEC CPU2017 workloads, we find that a single-core scale model in which the shared last-level cache and memory bandwidth are proportionally scaled, predicts 32-core system performance with an average error of 15% and at most 32%. To further improve accuracy, we explore a variety of machine learning (ML) based scale-model extrapolation techniques and we find that support vector machines (SVM) yield the most accurate approach. By adjusting the configuration of the scale model (i.e., by changing core count), scale model simulation spans a range of simulation speed versus accuracy trade-offs. The fastest configuration yields a 28× simulation speedup with an average absolute prediction error of 6% (and at most 21%) compared to a 32-core target simulation. The most accurate scale model extrapolation configuration brings down

- *W. Liu and L. Eeckhout are with Ghent University, Belgium. E-mail: wenjie.liu, lieven.eeckhout@ugent.be.*
- *W. Heirman and S. Eyerman are with Intel Corporation, Belgium. E-mail: wim.heirman, stijn.eyerman@intel.com.*
- *S. Akram is with the Australian National University. E-mail: shoaib.akram@anu.edu.au*

#cores	LLC	NoC	DRAM
32	32 MB: 32 slices	128 GB/s: 4 CSLs, 32 GB/s per CSL	128 GB/s: 8 MCs, 16 GB/s per MC
16	16 MB: 16 slices	64 GB/s: 4 CSLs, 16 GB/s per CSL	64 GB/s: 4 MCs, 16 GB/s per MC
8	8 MB: 8 slices	32 GB/s: 2 CSLs, 16 GB/s per CSL	32 GB/s: 2 MCs, 16 GB/s per MC
4	4 MB: 4 slices	16 GB/s: 2 CSLs, 8 GB/s per CSL	16 GB/s: 1 MC, 16 GB/s per MC
2	2 MB: 2 slices	8 GB/s: 1 CSL, 8 GB/s per CSL	8 GB/s: 1 MC, 8 GB/s per MC
1	1 MB: 1 slice	4 GB/s: 1 CSL, 4 GB/s per CSL	4 GB/s: 1 MC, 4 GB/s per MC

TABLE 1: Constructing scale models through *Proportional Resource Scaling*: LLC capacity in MB; on-chip interconnection network in GB/s: number of cross-section links (CSLs) and bandwidth per CSL; main memory bandwidth in GB/s: number of memory controllers (MCs) and bandwidth per MC.

the average prediction error to 2.7% (and at most 14%) while still delivering a 2.6× simulation speedup.

2 Scale-Model Architectural Simulation

Scale-model simulation is a novel methodology that combines architectural simulation of scale models with machine learning to predict large-scale system performance. There are two key objectives to be addressed for scale-model architectural simulation: (1) we need to construct the scale models, and (2) we need to build an accurate extrapolation model. We will now discuss both objectives.

2.1 Scale Model Construction

The first step is to create a scale model of the target large-scale multi-core system. A scale model is a scaled-down version of the large-scale target system such that its performance is still a (relatively) accurate representation of the target system. More precisely, the scale model needs to be configured such that its per-core performance is similar to per-core performance in the target system. The challenge when constructing scale models for general-purpose multicore processors is how to deal with shared resources. In a multi-core processor, there are various shared resources, including the last-level cache (LLC) which is typically shared among the cores while the L1 and L2 caches are typically private, the on-chip interconnection network in which the inter-core links are shared, and main memory in which the memory controllers as well as the memory banks and channels are shared. The question is how to scale these shared resources in the scale models.

One option is to simply scale the number of cores in the scale model while keeping the shared resources unchanged as in the target system — we refer to this approach as *No Resource Scaling (NRS)*. For example, a scale model consisting of a single core would simply have access to the fully sized LLC capacity as well as the same NoC bandwidth and main memory bandwidth as in the target system. We find though that such a scale model is largely inaccurate, as we will quantify in the results section of the paper. This can be understood intuitively: for example, the performance of a single-core scale model with a 32 MB LLC, 128 GB/s NoC bandwidth and 128 GB/s memory bandwidth will be (very) different from the per-core performance observed in a target system with 32 cores competing for the same shared LLC capacity and NoC/memory bandwidth.

Another, more accurate, option is to proportionally scale the shared resources with core count — we refer to this approach as *Proportional Resource Scaling (PRS)*. In particular, scaling LLC capacity, NoC bandwidth and memory bandwidth proportionally with core count leads to relatively accurate scale models. In particular, when scaling the number of cores by a factor F, we scale LLC capacity, NoC bandwidth and main memory bandwidth with the same factor F. In other words, we keep LLC capacity per core constant as we scale the number of cores. Similarly for interconnection and memory bandwidth, we keep bandwidth per core constant. In our setup, we assume 1 MB of LLC per core, 4 GB/s bisection bandwidth per core, and 4 GB/s memory bandwidth per core. How to implement PRS in practice depends on the specific nature of the shared resource, see also Table 1. Scaling cache capacity in our setup is trivial. Since we assume a NUCA LLC with a 1 MB slice attached to each core, we proportionally scale down LLC capacity as we consider fewer cores in the scale model. Scaling bandwidth is more complicated. We scale DRAM bandwidth by changing both the number of memory controllers and bandwidth per memory controller. Starting from the target system, we first scale down the number of memory controllers from 8 (at 32 cores) to 1 (at 4 cores), and then scale down the amount of bandwidth per memory controller. For the interconnection network, we scale link bandwidth as the number of cross-section links reduces with core count. In particular, moving from 32 to 16 cores, the number of cross-section links remains unchanged, hence we have to halve bandwidth per link from 32 GB/s to 16 GB/s. In contrast, when moving from 16 to 8 cores, the number of cross-section links halves from 4 to 2, hence we maintain the per-link bandwidth at 16 GB/s.

The intuition behind PRS is to provide balanced scale models that exhibit similar degrees of resource contention as in the target system. Intuitively speaking, a scale model with two cores and a total of 2 MB LLC capacity, 8 GB/s bisection bandwidth and 8 GB/s memory bandwidth, is likely to experience a level of interference in the shared resources that is (somewhat) comparable to the target system with 32 cores, 32 MB LLC, and 128 GB/s bisection NoC and memory bandwidth. In other words, the amount of resource contention is likely to be similar as the sizing of the shared resources is proportional to the number of cores in the system. Our experimental results confirm that PRS is indeed more accurate than NRS.

2.2 Scale Model Extrapolation

Scale model construction is only a first step. We need scale model extrapolation to yield even more accurate target system performance predictions. Scale-model extrapolation considers scale-model simulation results to predict target-system performance. We consider two extrapolation models in this work. Each of these approaches can be configured to provide different trade-offs in simulation speed versus accuracy.

No Extrapolation uses the per-core performance observed in the scale model as a prediction for per-core performance in the target system. This approach implicitly assumes that the interference in the shared resources is the same in the scale model as in the target system. *No Extrapolation* can be configured in different ways leading to different simulation speed versus accuracy trade-offs. The fastest configuration considers a single-core scale model: we simulate a single-core system (with the shared resources proportionally scaled following the PRS approach), and its measured performance is a prediction for per-core performance in the target system. Accuracy can be improved though by simulating a scale model with more cores, e.g., two cores with twice the shared resources as the single-core scale model, following PRS — this scale model's per-core performance typically yields a more accurate prediction for per-core performance in the target system. Of course, simulating a

dual-core scale model takes longer than simulating a single-core scale model, thereby providing a simulation speed versus accuracy trade-off.

Machine Learning-based Prediction leverages ML to yield higher accuracy. We consider three ML techniques: decision trees, random forest and support vector machines (SVM). We use the radial basis function (RBF) as the SVM kernel to capture non-linear performance scaling trends. *ML-based Prediction* follows a two-step process involving a training phase followed by an inference phase. The training phase is a one-time cost and involves simulating a number of training benchmarks for the scale model as well as the target system. When simulating the scale model, in addition to measuring performance (i.e., execution time), we also measure other possibly relevant performance metrics including memory bandwidth utilization, LLC misses per thousand instructions (MPKI) and NoC delay. The ML model is trained using the simulation data as input: the performance metrics obtained for the scale model are the independent variables, while performance of the target system is the dependent variable. At inference time, a previously unseen application of interest is simulated on the scale model. These simulation results (per-core performance of the scale model plus optional performance metrics) serve as input to the prediction model to predict performance for the unseen application on the target system. Training is a one-time cost whereas inference is a recurring cost for each application of interest. The more cores we consider in the scale model, the higher the accuracy — at the cost of a higher recurring cost, i.e., longer simulation times to collect the scale-model performance results. We perform meta-training of the ML parameters through cross-validation during training using the scikit-learn framework.[1]

3 Experimental Setup

We use Sniper v6.0, a parallel and high-speed cycle-level x86 simulator for multicore systems, using its most detailed cycle-level hardware-validated core model [3]. Our target system is a 32-core processor. We simulate 4-wide out-of-order cores with a 3-level on-chip cache hierarchy. The LLC is a 32 MB NUCA cache, and we assume a 128 GB/s bisection bandwidth mesh NoC and 128 GB/s main memory system with 8 memory controllers. We run rate-based multiprogram workloads with 32 benchmark instances from SPEC CPU 2017; we consider 1B-instruction simulation points per benchmark [1]. We run Sniper on a 36-core Intel PowerEdge R440 server.

4 Evaluation

We now evaluate scale model simulation. We first evaluate scale-model construction, and then evaluate scale-model extrapolation. Finally, we explore the simulation speed versus accuracy trade-off. We quantify accuracy using the following absolute prediction error metric:

$$error = \left| \frac{T_{predicted} - T_{actual}}{T_{actual}} \right|.$$

T_{actual} is the execution time of the application of interest on the target system — in our setup, this is the execution time of a single benchmark instance in a 32-copy multi-program workload. $T_{predicted}$ is the predicted execution time of the application of interest on the target system based on a measurement obtained through simulation of the scale model. In case of No Extrapolation, the predicted execution time on the target system *is* the execution time obtained on the scale model. In case of ML-based Prediction, the predicted execution time is provided by the ML model when given the performance metrics for the scale model as input. We use a cross-validation setup in which we use $N-1$ benchmarks for training the ML models when evaluating prediction accuracy for the Nth benchmark.

Scale Model Construction. We consider the following four scale-model construction techniques: (1) No Resource Scaling (NRS), i.e., the shared resources in the scale model are sized identically to the target system, (2) Proportional Resource Scaling (PRS) in which we only scale the LLC in the scale model (i.e., DRAM bandwidth in the scale model is the same as in the target system), (3) PRS with scaled DRAM bandwidth only (i.e., LLC capacity is the same in the scale model and target system), and (4) PRS with scaled LLC size *and* DRAM bandwidth. (Note that we evaluated NoC scaling as well but found it to have (virtually) no effect for the workloads considered in this work, hence we exclude it from the discussion.)

Figure 1 reports prediction error for the single-core scale model, i.e., we consider a scale model with a single core to predict per-core performance in the 32-core target system. NRS is highly inaccurate with an average absolute error of 60%. PRS is more accurate: scaling the LLC brings down the average error to 51%, while scaling DRAM bandwidth reduces the average error to 41%. Scaling both LLC capacity and DRAM bandwidth has synergistic effects, bringing down the prediction error to 15% and at most 32% (milc). Proportionally scaling all shared resources leads to a scale model that is a relatively accurate representation for per-core performance in the target system.

Scale Model Extrapolation. While PRS leads to relatively accurate scale models, we can do even better through scale-model extrapolation. No Extrapolation uses performance obtained for the scale model as a prediction for per-core performance in the target system — this is effectively PRS with scaled resources from the previous section. We further consider three ML-based Prediction techniques, namely *Decision Tree (DT)*, *Random Forest (RF)* and *Support Vector Machines (SVM)*. We use three independent variables obtained for the scale model as input to these ML models, namely execution time, memory bandwidth utilization and NoC delay. (LLC MPKI does not contribute to improved accuracy, hence we exclude it.)

Figure 2 reports prediction error for these techniques. ML-based Prediction brings down the average absolute prediction error by a significant margin compared to No Extrapolation (average error of 15% and up to 32%). SVM is to be the most accurate ML-based Prediction technique with an average error of 6% (maximum error of 21%). DT yields an average absolute prediction error of 9% (and up to 29%), whereas RF leads to an average error of 8% (and up to 21%).

Accuracy versus Simulation Speed Trade-Off. So far, we assumed single-core scale models. These scale models yield the highest simulation speedup. Indeed, simulating a single-core scale model is substantially faster than simulating a 32-core target system. In our setup, the simulation of a single-core scale model yields a 28× simulation speedup compared to simulating a 32-core system, while yielding an average 6% prediction error as reported in the previous section. It is possible to further increase accuracy by considering scale models with higher core count. In particular, we could simulate a scale model with 2, 4, 8 or even 16 cores to predict 32-core performance. It is to be expected that as we increase core count, accuracy is going to improve as the scale models gradually incur more contention in the shared resources with increasing core count.

Figure 3 reports prediction error versus simulation speed for No Extrapolation versus ML-based Prediction (DT, RF and SVM). The different data points on each of the curves correspond to scale models with 16, 8, 4, 2 and 1 cores, from left to right. Larger scale models (generally) improve accuracy while

1. https://scikit-learn.org/

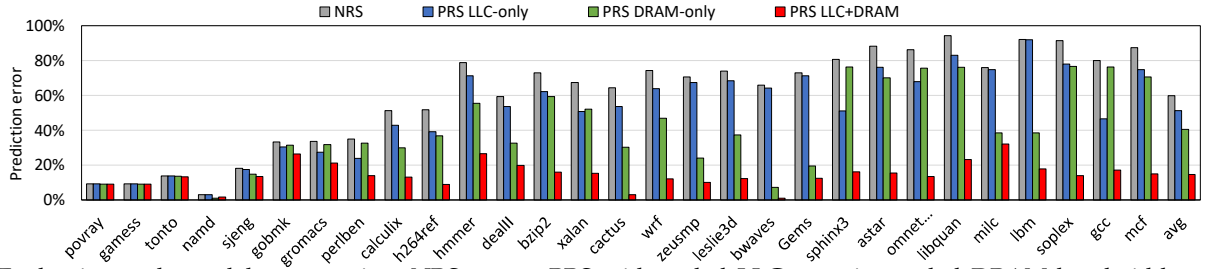

Fig. 1: Evaluating scale model construction: NRS versus PRS with scaled LLC capacity, scaled DRAM bandwidth, and both. *Proportional Resource Scaling (PARS) in which all shared resources are scaled proportionally leads to the most accurate scale models.*

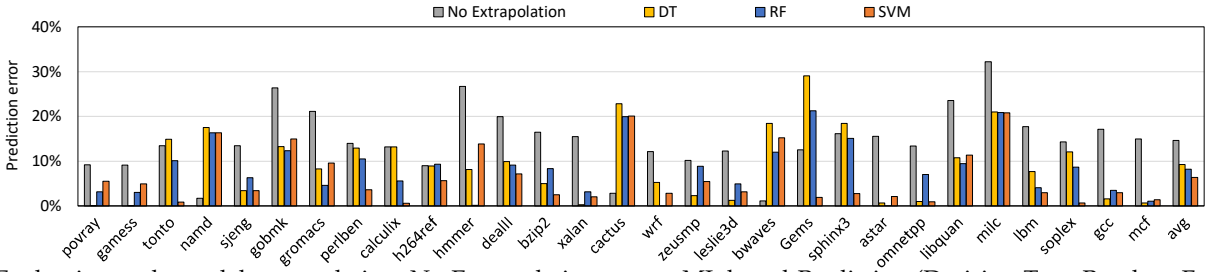

Fig. 2: Evaluating scale model extrapolation: No Extrapolation versus ML-based Prediction (Decision Tree, Random Forest and Support Vector Machines). *The SVM-based prediction method yields the highest accuracy (6% average absolute prediction error).*

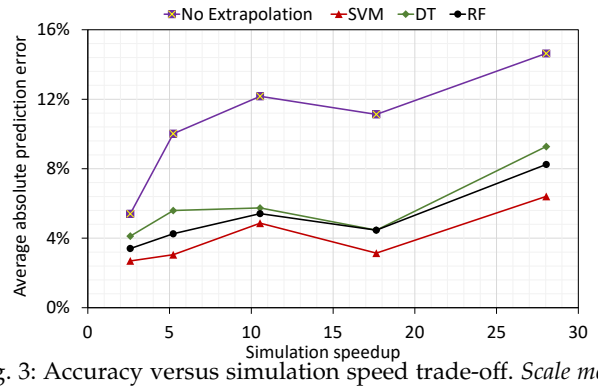

Fig. 3: Accuracy versus simulation speed trade-off. *Scale models with increased core count (generally) lead to higher accuracy at the cost of reduced simulation speedup.*

sacrificing simulation speed. ML-based Prediction uniformly outperforms No Extrapolation by a significant margin. In particular, the average absolute prediction error for SVM-based extrapolation decreases from 6% to 2.7% (at most 14%) as we go from a single-core to a 16-core scale model. This comes at the 'cost' of a simulation speedup reduction from 28× to 2.6×.

5 Related Work

The most closely related work by Eyerman et al. [4] proposes scale models for an experimental Intel processor, called PIUMA (Programmable Integrated Unified Memory Architecture), that is specifically designed for the efficient execution of graph analytics workloads. The lack of resource sharing among processor cores makes the development of scale models for this type of architecture relatively easy. More specifically, the PIUMA architecture does not have shared caches; each core has a dedicated memory controller; and a highly scalable interconnection network provides high bandwidth and low latency to each individual core. In contrast, general-purpose multi-core processors feature a vastly different architecture with various interference opportunities in the shared caches, NoC and memory.

Machine learning (e.g., neural networks [5] and spline-based regression [6]) was previously proposed to explore microprocessor design spaces. Alameldeen et al. [7] propose a methodology for scaling down commercial workloads in both size and runtime, allowing commodity machines to simulate much more powerful server systems. Hoste et al. [8] and Piccart et al. [9] determine the optimum platform among a set of previously benchmarked platforms for an application of interest. Other prior work predicts performance across architecture paradigms. Baldini et al. [10] and Ardalani et al. [11] propose machine-learning based methodologies to predict GPU performance using (single-threaded) CPU implementations.

6 Conclusion

This paper proposed scale-model simulation, a novel methodology to predict large-scale system performance. Configuring the scale model enables trading off accuracy versus simulation speed. Our experimental results demonstrate high accuracy and simulation speed. We plan to extend and evaluate scale-model simulation for multi-threaded workloads as well as other architecture paradigms (e.g., GPUs).

References

[1] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically characterizing large scale program behavior," in *ASPLOS*, 2002.

[2] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe, "SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling," in *ISCA*, 2003.

[3] T. E. Carlson, W. Heirman, S. Eyerman, I. Hur, and L. Eeckhout, "An evaluation of high-level mechanistic core models," *ACM Transactions on Architecture and Code Optimization*, pp. 1–25, 2014.

[4] S. Eyerman, W. Heirman, Y. Demir, K. Du Bois, and I. Hur, "Projecting performance for PIUMA using down-scaled simulation," in *HPEC*, 2020.

[5] E. Ipek, S. McKee, R. Caruana, d. B. R., and M. Schulz, "Efficiently exploring architectural design spaces via predictive modeling," in *ASPLOS*, 2006.

[6] B. C. Lee and D. M. Brooks, "Accurate and efficient regression modeling for microarchitectural performance and power prediction," in *ASPLOS*, 2006.

[7] A. R. Alameldeen, M. M. Martin, C. J. Mauer, K. E. Moore, M. Xu, M. D. Hill, D. A. Wood, and D. J. Sorin, "Simulating a $2 M Commercial Server on a $2 K PC," *Computer*, 2003.

[8] K. Hoste, A. Phansalkar, L. Eeckhout, A. Georges, L. K. John, and K. De Bosschere, "Performance prediction based on inherent program similarity," in *PACT*, 2006.

[9] B. Piccart, A. Georges, H. Blockeel, and L. Eeckhout, "Ranking commercial machines through data transposition," in *IISWC*, 2011.

[10] I. Baldini, S. J. Fink, and E. Altman, "Predicting GPU performance from CPU runs using machine learning," in *SBAC-PAD*, 2014.

[11] N. Ardalani, C. Lestourgeon, K. Sankaralingam, and X. Zhu, "Cross-architecture performance prediction (XAPP) using CPU code to predict GPU performance," in *MICRO*, 2015.