Scale-Model Simulation

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

Abstract—Computer architects extensively use simulation to steer future processor research and development. 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. Scale-model simulation predicts 32-core target system performance based on a single-core scale model with an average error of 8.0% and 15.8% for homogeneous and heterogeneous workloads, respectively, while yielding a 28× simulation speedup.

Index Terms—Performance prediction, multi-core system simulation, machine learning

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 observe 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. 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 predicts the impact of contention effects in shared resources on target-system performance based on the simulated scale model.

We find that proportionally down-scaling the shared resources with system size is an effective approach for constructing scale models. 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. Our evaluation using multiprogram SPEC CPU2017 are encouraging: while yielding a simulation speedup of 28×, scale-model simulation predicts 32-core target system performance based on a single-core scale model with an average prediction error of 8.0% and 15.8% for homogeneous and heterogeneous workload mixes, respectively, while not requiring target-system simulation runs during ML training.

2 Scale-Model Architectural Simulation

We now discuss (1) how to construct the scale models, and (2) how to build an accurate extrapolation model.

2.1 Scale Model Construction

A scale model is a scaled-down version of the large-scale target system such that its performance is 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 multicore processors is how to deal with shared resources.

- *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.

One option is to simply scale the number of cores in the scale model while keeping the shared resources 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 have access to the fully sized LLC capacity as well as the same NoC and memory bandwidth as in the target system.

Another, more accurate, option is to proportionally scale the shared resources with core count — we refer to this approach as *Proportional Resource Scaling (PRS)*. The intuition behind PRS is to provide balanced scale models that exhibit similar degrees of resource contention as in the target system. In particular, when scaling the number of cores by a factor F, we scale LLC capacity, NoC bandwidth and memory bandwidth by the same factor F. In other words, we keep LLC capacity per core constant and we keep interconnection and memory 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. See Table 1 for how we scale shared resources in our setup.

2.2 Scale Model Extrapolation

We consider two extrapolation models in this work.

2.2.1 No Extrapolation

The simplest way to predict target system performance is to use 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 observed in the shared resources in the scale model is similar to (or the same as in) the target system. The scale model that we assume is a single-core system with the shared resources proportionally scaled following the PRS approach. The performance measured for this single-core scale model then is the prediction for per-core performance in the target system.

While we assume a single-core scale model in this paper, it might be worth considering a two-core scale model or a four-core scale model (again, with the shared resources proportionally scaled). This typically leads to higher accuracy, while incurring longer simulation times. In this paper we report results for a single-core scale model which yields the highest possible simulation speedup.

2.2.2 Machine Learning-based Prediction and Regression

Leveraging Machine Learning (ML) enables achieving higher accuracy compared to the No Extrapolation method. We consider two ML-based approaches: *ML-based Prediction* and *ML-based Regression*. Both methods involve a training phase during which a performance model is trained. The training phase incurs a one-time cost. The key difference between both approaches is that ML-based Regression does not need simulation runs of the target system during training, in contrast to ML-based Prediction. This has important implications in practice. In case it is impossible to simulate the target system for some reason (too long simulation time or other simulator limitations), one has to resort to ML-based Regression.

ML-based Prediction involves a training phase in which a set of training benchmarks are run on both the scale model and the target system. On the single-core scale model, we measure performance (i.e., IPC) and memory bandwidth utilization. The latter is a function of the number of LLC misses per unit of time and has a significant impact on resource contention in the memory subsystem during co-execution with other benchmarks. In other words, it provides a measure for how much contention the particular application is going to create on the shared resources when co-executed with other applications. Our results confirm that considering both performance and memory bandwidth utilization improves accuracy (results not shown here due to space constraints). The performance and bandwidth utilization numbers on the single-core scale model serve as independent variables to the ML technique. More precisely, the input variables to the ML model are per-core performance for each of the benchmarks in the mix, alongside the sum of the per-core bandwidth utilization numbers for the co-running applications in the workload mix. On the target system, we measure performance for each of the benchmarks in the multi-program workload mix. Target system performance for each of the benchmarks in the multiprogram workload mix serves as dependent variables to the ML technique. We consider three ML techniques in our work, namely decision tree (DT), random forest (RF) and support vector machines (SVM). RF includes a diverse set of decision trees to avoid overfitting. We use the radial basis function (RBF) as the SVM kernel to capture non-linear performance scaling trends. The inference (prediction) phase involves simulating a previously unseen application (i.e., the workload of interest) on the single-core scale model. The measured performance and bandwidth utilization numbers serve as input to the prediction model which then yields a prediction for performance of the application of interest on the target system. More specifically, the prediction model takes the IPC of the application of interest on the single-core scale model as input, alongside the total bandwidth consumption of the co-running applications in the workload mix. The latter is computed as the sum of the bandwidth consumption for each of the applications in the workload mix as observed in the single-core scale model.

As mentioned above, ML-based Prediction requires simulation runs of the target system during training, which may be a significant impediment in practice. *ML-based Regression* overcomes this drawback by relying on simulation runs of a variety of scale models instead, which is typically easier to achieve in practice. ML-based Regression consists of two steps. In the first step, ML-based Regression leverages the ML-based Prediction method discussed above to predict performance for a number of (multi-core) scale models. In particular, this involves measuring performance and bandwidth utilization on the single-core scale model, and measuring performance for a number of multi-core scale models. ML-based Prediction is used to train the prediction models for the multi-core scale models. For example, we use ML-based Prediction to train prediction models for 2-core, 4-core, 8-core and 16-core scale-model performance, respectively, based on the single-core scale model. As a second step, once these prediction models have been trained, we use extrapolation techniques to predict per-

formance for the larger scale target system. More specifically, we use regression to predict performance for the 32-core target system based on the predicted performance of the 2-core, 4-core, 8-core and 16-core scale-models. We consider a number of regression techniques, including linear regression, power-law regression and logarithmic regression, to predict target-system performance. We find that logarithmic regression yields the highest accuracy (detailed analysis excluded due to space constraints), which is what we report in this paper.

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 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 Sniper on a 36-core Intel PowerEdge R440 server.

We consider both homogeneous and heterogeneous multi-program workload mixes in the evaluation. The benchmarks are taken from SPEC CPU2017 and we consider 1B-instruction simulation points per benchmark [1]. The homogeneous workloads assume co-running instances of the same benchmark, all starting at (slightly) different offsets. The heterogeneous workload mixes are randomly composed. We finish the simulation and measure performance when the first benchmark in the workload mix has reached the end of its simulation point.

We make sure that the training set is completely disjoint from the evaluation set in all of our experiments. For the homogeneous workload mixes, we use a cross-validation setup in which we use $N-1$ benchmarks for training the models when evaluating prediction accuracy for the Nth benchmark, with $N=29$ for SPEC CPU2017. For the heterogeneous workload mixes, we consider 8 randomly chosen benchmarks in the evaluation set while using the 21 remaining benchmarks in the training set. The workload mixes in the training set are random mixes: when simulating a K-core system, we randomly select K benchmarks from the training set. This yields K training results, i.e., K per-core performance results for all K benchmarks in the mix. We consider a total of 320 training results to train the ML models. Prediction is done for a previously unseen application of interest which we simulate on the single-core scale model. We then use the prediction and extrapolation models to predict performance for the application of interest on the target system when co-run with additional copies of the application of interest for the homogeneous workloads. For the heterogeneous workloads, we predict performance for the application of interest on the target system when co-run with 10 random heterogeneous mixes of previously unseen applications from the evaluation set; we report the average prediction error across these 10 mixes.

4 Evaluation

We now evaluate scale model simulation. We first evaluate scale-model construction, and then evaluate scale-model extrapolation. We quantify accuracy using the following absolute prediction error metric: $error = \left|\frac{IPC_{predicted}-IPC_{actual}}{IPC_{actual}}\right|$. IPC_{actual} is the IPC of the application of interest on the target system — in our setup, this is the IPC of a single benchmark instance in a 32-instance multi-program workload. $IPC_{predicted}$ is the predicted IPC of the application of interest on the target system based on measurements obtained through simulation of the scale model. In case of No Extrapolation, the predicted IPC on the target system *is* the IPC obtained on the scale model. In case of ML-based Prediction and Extrapolation, the predicted IPC is provided by the ML model when given the performance metrics for the scale model as input.

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. (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. The benchmarks are sorted by their LLC MPKI from left to right. NRS works well for compute-intensive workloads but is generally inaccurate with an average absolute error of 60%. PRS is more accurate, especially for memory-intensive workloads: scaling the LLC brings down the average error to 51.3%, while scaling DRAM bandwidth reduces the average error to 40.5%. Scaling both LLC capacity and DRAM bandwidth has synergistic effects, bringing down the prediction error to 14.7% and at most 32.2% (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 ML-based Prediction and ML-based Regression; we consider three ML techniques — Decision Tree (DT), Random Forest (RF) and Support Vector Machines (SVM) — and we use logarithmic regression for the Regression approach.

Figure 2 reports the prediction error for these techniques assuming homogeneous workload mixes. ML-based Prediction brings down the average absolute prediction error by a significant margin compared to No Extrapolation (average error of 14.7% and up to 32.2%). SVM is the most accurate ML-based Prediction technique with an average error of 6.4% (maximum error of 20.8%). DT yields an average absolute prediction error of 9.3% (and up to 29.1%), whereas RF leads to an average error of 8.3% (and up to 21.3%). ML-based Regression is slightly less accurate than ML-based Prediction as it does not require simulating the target system during training. Yet, accuracy is still high and SVM with logarithmic regression (SVM-log) yields the highest accuracy among the ML-based Regression techniques with an average absolute prediction error of 8.0% (and at most 26.4%).

Heterogeneous Workload Mixes. So far, we considered homogeneous workload mixes. Figure 3 reports prediction error for the various prediction techniques under heterogeneous workload mixes. The results are consistent with the homogeneous workload mixes, i.e., ML-based Prediction is slightly more accurate than ML-based Regression, and SVM is the most accurate ML approach. We do note higher prediction errors for the heterogeneous workload mixes compared to the homogeneous workload mixes: average prediction error of 15.8% (max 28.7%) for SVM-log versus 13.2% (max 27.5%) for SVM, versus 27.8%

This article has been accepted for publication in IEEE Computer Architecture Letters. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/LCA.2021.3133112

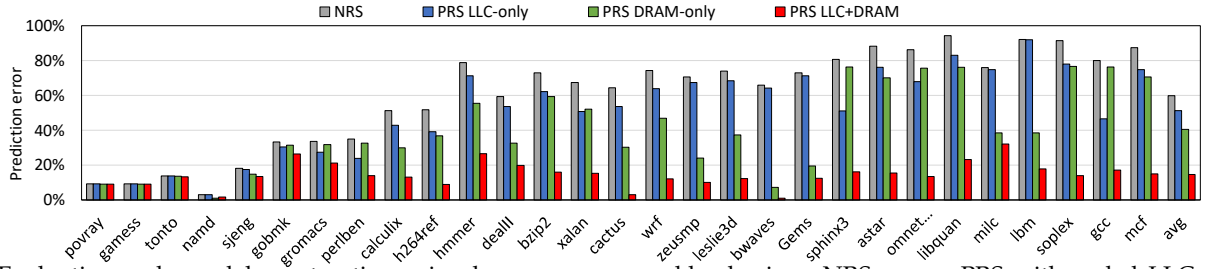

Fig. 1: Evaluating scale model construction using homogeneous workload mixes: NRS versus PRS with scaled LLC capacity, scaled DRAM bandwidth, and both. *Proportional Resource Scaling (PRS) in which all shared resources are scaled proportionally leads to the most accurate scale models.*

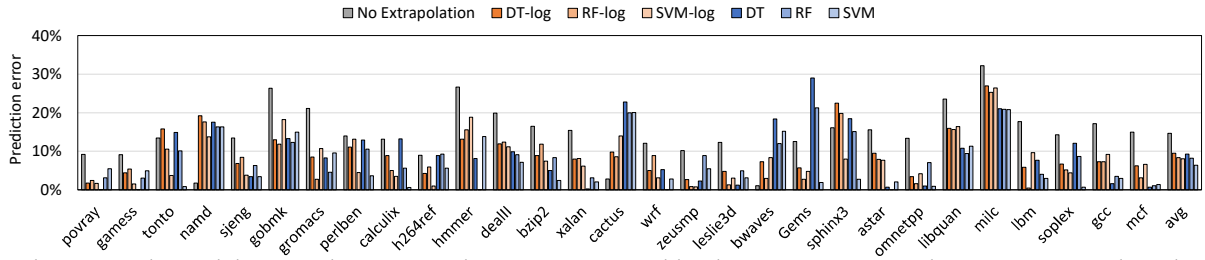

Fig. 2: Evaluating scale model extrapolation using homogeneous workload mixes: No Extrapolation versus ML-based Prediction and Regression. *The SVM-based Prediction method yields the highest accuracy (6.4% average absolute prediction error), while SVM-based Regression (SVM-log) is only slightly less accurate (8.0% average absolute prediction error).*

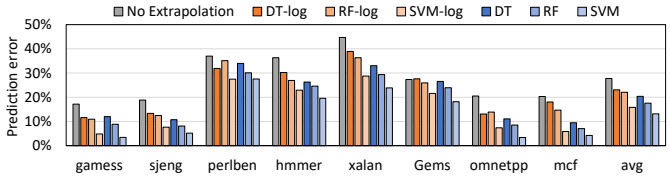

Fig. 3: Evaluating scale model extrapolation using heterogeneous workload mixes: No Extrapolation versus ML-based Prediction and Regression. *The SVM-based Prediction method yields the highest accuracy (13.2% average absolute prediction error), while SVM-based Regression (SVM-log) is only slightly less accurate (15.8% average absolute prediction error).*

(max 44.7%) for No Extrapolation.

Simulation Speedup. Scale-model simulation yields a substantial simulation speedup because simulating a single-core scale model takes considerably less time than simulating the target system. And in some cases, it might not even be possible to simulate the target system, due to simulator infrastructure constraints. Once scale-model simulation results are available, predicting target-system performance is almost instantaneous provided that the ML model has been trained offline. For our setup, we find that scale-model simulation yields a 28× simulation time reduction.

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 a novel methodology to predict large-scale system performance based on single-core scale-model simulation results. 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).

7 Acknowledgments

This work is supported through the European Research Council (ERC) Advanced Grant agreement No. 741097, Research Foundation Flanders (FWO) grants No. G.0434.16N and G.0144.17N. Wenjie Liu is supported through a CSC scholarship.

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.