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.2022.3203483

LMT: Accurate and Resource-Scalable Slowdown Prediction

Peter Salvesen and Magnus Jahre

Abstract—Multi-core processors suffer from inter-application interference which makes the performance of an application depend on the behavior of the applications it happens to be co-scheduled with. This results in performance variability, which is undesirable, and researchers have hence proposed numerous schemes for predicting the performance slowdown caused by inter-application interference. While a slowdown predictor's primary objective is to achieve high accuracy, it must typically also respect resource constraints. It is hence beneficial to be able to scale the resource consumption of the predictor, but state-of-the-art slowdown predictors are not resource-scalable. We hence propose to construct predictors using Linear Model Trees (LMTs) which we show to be accurate *and* resource-scalable. More specifically, our 40-leaf-node LMT-40 predictor yields a 6.6% prediction error compared the 8.4% error of state-of-the-art GDP at similar storage overhead. In contrast, our LMT-10 predictor reduces storage overhead by 34.6% compared to GDP while only increasing prediction error to 9.4%.

Index Terms—Multi-core processors, slowdown prediction, memory system resource management, cloud computing.

1 Introduction

Multi-core processors typically share memory system components, such as the Last Level Cache (LLC) and memory controllers, to improve utilization. Unfortunately, resource sharing also creates the possibility for requests from co-executing applications to interfere with each other in the shared memory system. This is undesirable because interference causes performance variability which in turn can result in problems such as missed deadlines, priority inversion, and unpredictable interactive performance [1]. This problem is particularly important in cloud infrastructures where interference can lead to users being billed for resources they could not use effectively as well as Quality of Service (QoS) and Service-Level Agreement (SLA) violations [2].

Broadly speaking, the performance variability caused by memory system interference can be mitigated by system software — e.g., by allocating more time to co-runners that are slowed down by interference [1] — or by architectural schemes — e.g., by allocating LLC capacity or memory bandwidth such that slowdowns are reduced [3], [4]. In both approaches, the fundamental problem is to predict the slowdown incurred by each co-running application, or in other words, predicting each application's normalized progress. Normalized progress is the ratio of the performance of the application during multi-tasking (i.e., the *shared mode*) over its performance with exclusive access to all shared resources (i.e., the *private mode*) [2], [4]. While shared-mode performance can be straightforwardly measured during multi-tasking with Performance Monitoring Counters (PMCs), private-mode performance must be predicted as it is impractical, or even impossible, to run all possible applications with all possible input sets in isolation to measure their actual private mode performance.

Many slowdown predictors have been proposed (e.g., GDP [4], PTCA [1], and ASM [3]), and they either (i) detect interference in hardware and thereby predict slowdowns without affecting performance, or (ii) let each application run with high priority in a round-robin fashion in the shared mode, thereby predicting slowdown by minimizing interference. These predictors occupy single points in the accuracy versus overhead design space which is undesirable because they might require more resources than can be devoted to slowdown prediction in a particular architecture. Since predictor accuracy typically increases with resource consumption, it is more desirable to integrate a (slightly) less accurate predictor than none at all — creating a need for accurate *and* resource-scalable slowdown predictors.

We hence propose to design slowdown predictors using Linear Model Trees (LMTs) which are decision trees that use linear regression rather than constants in leaf nodes. LMT-based predictors can be straightforwardly implemented in hardware to predict slowdowns at runtime and are *accurate*, e.g., LMT-40, which has 40 leaf nodes, yields an average Root Mean Squared (RMS) prediction error of 6.6%, compared to 8.4% error for state-of-the-art GDP [4], at 4.4% higher storage overhead. LMT predictors are *resource-scalable* because their accuracy typically decreases (improves) when reducing (increasing) node count, e.g., our 10-leaf-node LMT-10 predictor yields an RMS error of 9.4% while reducing storage overhead by 34.6% compared to GDP.

2 Slowdown Prediction

Slowdowns are caused by memory requests of co-runners interfering with each other in the shared memory system, thereby increasing memory latencies. Requests may (i) need to wait for requests from other applications in the interconnect or memory bus, (ii) the applications may evict each others data in the LLC, and (iii) applications can destroy each others DRAM page locality. Slowdowns can hence vary significantly since applications have different memory intensity and varying ability to hide memory latencies.

Interference-aware performance metrics, such as System Throughput (STP) and Average Normalized Turn-around Time (ANTT), first compute normalized progress for each co-runner [5]. Normalized progress is a number between 0 (no

- *Peter Salvesen is with Arm, Trondheim, Norway. The work was performed while Salvesen was at NTNU. E-mail: peter.salvesen@arm.com.*
- *Magnus Jahre is with the Norwegian University of Science and Technology (NTNU), Trondheim, Norway. E-mail: magnus.jahre@ntnu.no.*

M. Jahre is supported by the Research Council of Norway under Grant 286596.

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.2022.3203483

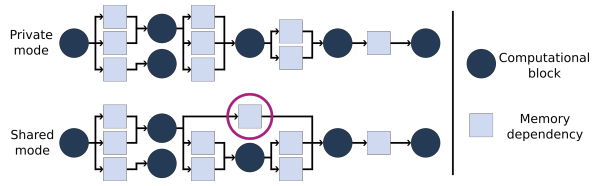

Fig. 1: Partial GDP data-flow graph for Lucas.

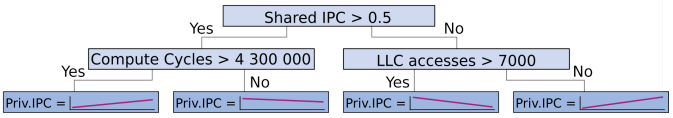

Fig. 2: Linear Model Tree (LMT) example.

progress) and 1 (no interference) because private-mode performance is always greater than or equal to shared-mode performance; recall that each application runs alone with exclusive access to all shared resources in the private mode. While shared-mode performance can be straightforwardly measured at runtime with PMCs, measuring private-mode performance requires running the each application in isolation. This is impractical or even impossible in practice and researchers have hence proposed schemes that *predict* private-mode performance from information that is available in the shared mode [1], [3], [4], [6].

Existing slowdown predictors rely on heuristics to identify when interference results in performance loss. GDP [4] for instance assumes that the application's data-flow graph is very similar in the shared and private modes, but Figure 1 shows that the shared and private-mode data-flow graphs of the Lucas benchmark are clearly different. More specifically, interference delayed a memory request in the shared mode which resulted in its latency being hidden (see the red ring in Figure 1) whereas the latency of this request was exposed in the private mode. PTCA [1] on the other hand assumes that interference affects performance whenever a load stalls at the head of the reorder buffer, yet some requests that are delayed by interference in the shared mode will also stall the core in the private mode. Predicting private mode performance is hence challenging because slowdown predictors must appropriately account for all important shared-mode interference-related behaviors and their impact on private-mode performance. Our LMT-based predictors learn this relation whereas prior work rely on heuristics.

3 LMT-based Slowdown Prediction

Capturing data sets. Training LMT predictors require capturing a data set which maps the shared-mode PMC values of an application to its private-mode performance, i.e., Instructions Per Cycle (IPC). Memory system resource management schemes are typically epoch-based which means that the predictor must capture PMC values across a fixed clock-cycle epoch. We hence retrieve shared-mode PMCs for all applications in a workload once every 5 million clock cycles (which is in line with prior work [4], [7]). Relating shared-mode PMCs to private-mode performance is challenging because interference affects the performance of a single application differently across workloads, but PMCs and private-mode IPC must be taken from the exact same instruction window for the LMTs to yield good predictions. We address this challenge by recording the dynamic instruction window alongside PMC values in the shared mode and then capture private-mode performance traces for these instruction windows for each application in the workload. Since we focus on a 4-core architecture, this results in one shared-mode experiment and four private-mode experiments for each workload. Note however that each workload runs for between 90 million and 7 billion clock cycles and hence yield from 18 to 1,400 data points.

Training and prediction. As mentioned before, an LMT combines a decision tree with linear regression (see Figure 2). We start out with a single node representing the total training data. The root node is split recursively to form a tree, where every data point belongs to one leaf node, following a line of binary splits from the root node. Using the average private-mode IPC of the training data points within a leaf node as their prediction, each split minimizes the RMS error of the prediction. The split compares a shared-mode PMC (feature) with a constant and determines whether to go left or right in the tree. We split recursively until we reach the desired number of nodes. At that point, an ordinary least squared linear regression model is trained in every leaf node based on the observed training data belonging to that node. Training is performed at design-time and is hence a one-time cost for each architecture. To predict private-mode performance at runtime, PMCs are retrieved at the end of an epoch and used to follow a path through the tree to a leaf node in which the the predicted private-mode performance (target) is the output of that node's linear regression.

For the LMT predictor to generalize beyond the applications and workloads of the training set, i.e., provide accurate predictions for unseen applications and workloads, it is necessary to create a training set that exposes all important behaviors. We hence use 51 SPEC CPU benchmarks from various suites and randomly generate 90 workloads for our training set and 25 workloads for our validation set yielding in total 15,284 (3,347) training (validation) data points. While this is sufficient for the architecture we consider in this work, architects must be careful to select a comprehensive and unbiased data set for each LMT deployment — an ubiquitous challenge within supervised machine learning. The main overhead of training an LMT-predictor is capturing the data set, but, once the data set has been created, training is practically instantaneous.

Feature selection. The features are the PMCs available in the architecture which record interference-relevant information. We identified 26 such counters which include the cycles the core was committing instructions and stalled for various reasons (e.g., on a shared memory system load) and cache misses and cache writebacks at different levels. (The details are available in [8].) We found that accounting for interference in the LLC requires (sampled) Auxiliary Tag Directories (ATDs) [7], and we hence include the predicted private mode LLC counters (e.g., hits and accesses) as possible features. Many of the selected PMCs are available in contemporary multi-cores; the key exception is the ATD-related counters. Once the PMCs are in place, there is however no significant overhead associated with making them available to the LMT predictor, i.e., their value must simply be routed to the predictor. Since training is performed at design-time, it is only necessary to route the PMCs selected during training to the hardware predictor.

The trade-off is different for the linear regression component in the leaf nodes of the LMT. In this case, all selected features will be assigned a coefficient during training which we have to allocate storage for in all leaf nodes to predict private-mode IPC at runtime. It is hence beneficial to select a limited number of features with high predictive power. To this end, we used

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.2022.3203483

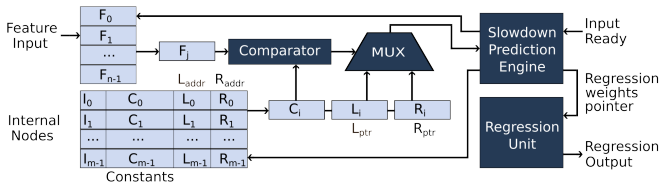

Fig. 3: A generic LMT-predictor implementation.

the R^2 metric to compute the linear variance of combinations of features while iteratively adding features used for linear regression, and we found that choosing 12 of the total 26 features gave 98.3% of the maximum potential R^2 score. We hence use 12 features for linear regression in all LMT configurations. For the 10-leaf-node tree, the features used within the decision tree are a subset of the features used for linear regression whereas the larger trees use some additional features (see [8] for details).

Implementation. Figure 3 shows a generic hardware implementation of an LMT-based slowdown predictor. During each epoch, the architecture counts performance events and stores them in the n feature registers (i.e., $F_0, \ldots, F_{n-1}$). At the end of the epoch, the Slowdown Prediction Engine (SPE) traverses the tree. For each node i, SPE retrieves F_j, which is the value of feature j, and compares it to the constant C_i. The table that contains C_i also contains flags that identify the feature to be used, j in this case, and if the node is a leaf node. In the root node of Figure 2, the control unit would for example retrieve shared-mode IPC (the feature) and use the comparator to check if it is greater than 0.5 (the constant). The outcome of the comparison determines which node SPE will fetch next by accessing the L_i or R_i pointers. When the evaluation reaches a leaf node, SPE instructs the Regression Unit (RU) to compute the linear regression using the weights of this particular leaf node. These are stored in a table which RU iterates through. Since linear regression is a sum of products, RU consists of a multiplier, an adder, and simple control logic.

All the LMT-predictor hardware is off the critical path and has no direct performance impact. We estimate prediction latency by accounting for the cycles spent on feature retrieval, tree traversal, and computing the linear regression, assuming a one cycle latency for addition, three for multiplication, and 25 for division. This results in prediction latencies between 91 cycles (LMT-10) and 98 cycles (LMT-80). The key drivers of storage overhead are the regression tables and the ATD. The regression tables require one entry for each leaf node and feature which yields storage overheads of 1.8 kB, 7.3 kB, and 14.3 kB for LMT-10, LMT-40, and LMT-80, respectively, when assuming 24-bit values. The storage overhead of the ATD is 7.3 kB, yielding overall storage overheads of 9.1 kB, 14.6 KB, and 21.6 KB, respectively. The main energy overhead of the LMT predictors is the accesses to the ATD, but this is negligible compared to the energy cost of an LLC access. In our setup, the energy cost of an ATD access is 8.1 pJ compared to 430.6 pJ for an LLC access according to CACTI [9] assuming 22 nm technology.

4 Experimental Setup

We implement our LMT-based predictors within the simulator infrastructure that was used to evaluate GDP [4], and model a 4-core system configured as shown in Table 1. We simulate four-benchmark multi-programmed workloads in which each benchmark is pinned to a physical core. We select a representative 100 million instruction sample for each benchmark using SimPoint [10], and simulate the workload until all benchmarks have committed 100 million instructions. We restart benchmarks when they reach the end of their simulation sample. For each workload, we run the private-mode simulations after the shared-mode simulation and retrieve statistics that exactly match the instructions that were executed by each benchmark in each epoch in the shared mode.

TABLE 1: Simulator configuration.

Parameter	Value
Clock frequency	4 GHz
Processor Cores	128 entry reorder buffer, 32 entry load/store queue, 64 entry instruction queue, 4 instructions/cycle, 4 integer ALUs, 2 integer multiply/divide, 4 FP ALUs, 2 FP multiply/divide, 2048 entry hybrid branch predictor, 2048 entry 4-way BTB
L1 Data Cache	2-way, 64KB, 3 cycles latency, 16 MSHRs
L1 Instr. Cache	2-way, 64KB, 3 cycles latency, 16 MSHRs
L2 Private Cache	4-way, 1MB, 9 cycles latency, 16 MSHRs
L3 Shared Cache	16-way, 8MB, 16 cycles latency, 256 MSHRs, 4 banks
Ring Interconnect	4 cycles per hop transfer latency, 32 entry request queue, 1 request ring, 1 response ring
Main memory	DDR4-2666, 18-18-18-43 timing, 64 entry read queue, 64 entry write queue, 16 banks, 2 channels, FR-FCFS scheduling, open page policy

We consider 51 benchmarks from different SPEC CPU suites, and classify them as as streaming (S), highly memory sensitive (H), medium memory sensitive (M), and low memory sensitive (L) by comparing the slowdown incurred in the private mode when we reduce LLC capacity and memory bandwidth. We then randomly generated 90 workloads for our training set out of which 25, 25, 10, and 10 workloads consist of only S, H, M, and L benchmarks, respectively, and we refer to these as S, H, M, and L-workloads. In addition, we generate 20 A-workloads where all benchmarks are eligible. For our validation set, we randomly generate 25 workloads (5 in each category). The workloads and more details regarding our setup is available in [8].

5 Results

Prediction error. Figure 4 compares RMS private-mode IPC prediction error for LMT-10, LMT-40, and LMT-80 to GDP [4]; the storage overhead of LMT-40 and GDP are similar. A key take-away is that accuracy improves with node count for the LMT predictors, i.e., LMT-80 typically has lower error than LMT-40 which in turn typically has lower error than LMT-10. The exceptions are A3, in which LMT-40 has lower error than LMT-80, and M0 where the error trend is reversed. In these cases, the LMT predictors misclassify behavior because the required PMC to private-mode IPC mapping is not well represented in the training data. Our LMT predictors perform favorably compared to GDP, in particular for the H- and S-workloads where GDP has higher errors because more memory congestion makes GDP's data-flow graph prone to being different in the shared and private modes (see Section 2). This leads to LMT-40 providing 21.0% lower error than GDP on average with similar storage overhead. LMT-10 reduces overhead by 34.6% compared to GDP, which results in 12.6% higher error than GDP, whereas LMT-80 reduces error by 35.9% compared to GDP at the cost of 55.3% larger overhead.

GDP performs well compared to our LMT-predictors in the M-workloads. This is primarily caused by two factors. First, the training data is skewed towards H- and S-workloads. The relatively limited training data for M-workloads makes them prone to larger prediction errors. Specially workload M0 points to this, as prediction error grows with LMT size. Second,

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.2022.3203483

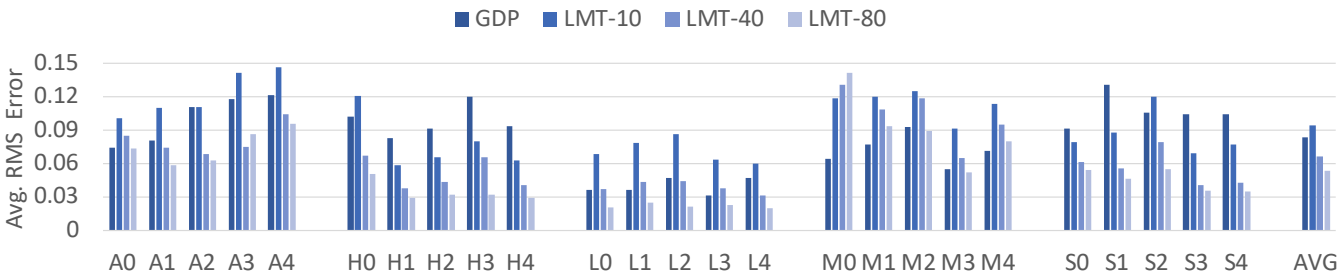

Fig. 4: Average private mode IPC RMS error for our LMT-predictors compared to GDP [4] for our test set workloads.

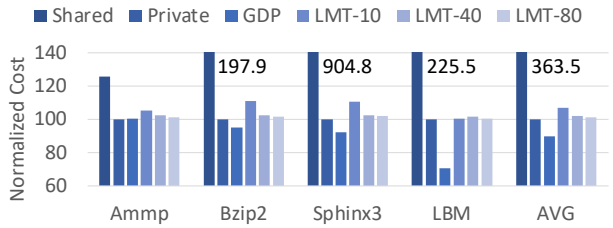

Fig. 5: Interference-aware cloud-pricing case study for H0.

the interference characteristics of the benchmarks in the M-workloads are less distinct than for other workload categories, making accurate sub-population classification more difficult. As M-workloads are somewhat in the midpoint of all the other workload categories, it is more likely to have traits in the traced features which overlap with those of other categories.

Slowdown-aware pricing in cloud systems. We now consider a case study where we use private mode performance predictions to provide interference-aware billing in cloud infrastructures, i.e., billing customers according to private-mode performance rather than shared-mode performance. We assume that the cloud computing provider charges a fixed cost per compute hour, and the cost of running each benchmark is therefore proportional to its execution time. An interference-aware billing scheme hence aims to predict private-mode execution time as this is the execution time the benchmark would experience without interference. We use our LMT-predictors and GDP to predict private-mode execution time for each benchmark and normalize to the actual private-mode execution time to compute normalized cost (compute hour cost cancels during normalization).

Figure 5 shows normalized cost for the H0 workload. Interference is severe for H0 which means that shared-mode cost is 9.0× higher than private-mode cost for Sphinx3. In contrast, the shared-mode cost for Ammp is only 25.7% higher than the private-mode cost. Accounting for interference when billing in the cloud can hence be critical because slowdowns vary widely across benchmarks. Our LMT-predictors are accurate, and the trend with LMT-80 being more accurate than LMT-40 which in turn is more accurate than LMT-10 still holds. GDP underestimates cost by 29.3% for LBM due its data-flow graph being different in the shared and private modes, resulting in all our LMT-predictors being more accurate than GDP.

6 Related Work

Existing slowdown predictors are either fully implemented in hardware and hence non-invasive (e.g., GDP [4], PTCA [1], and ITCA [6]) or rely on periodically running each benchmark with high priority to approximate private mode performance (e.g., ASM [3]). Unlike these predictors, which all incur fixed overhead, our LMT-based predictors are resource-scalable. Moreover, we showed in Section 5 that our LMT-40 predictor provides better accuracy than GDP at a similar storage overhead; GDP has previously been shown to outperform PTCA, ITCA, and ASM [4].

7 Conclusion

We have demonstrated that slowdown predictors based on Linear Model Trees (LMTs) are accurate *and* resource-scalable. Our LMT-40 predictor reduces prediction error by 21.0% compared to GDP at a similar area overhead whereas our LMT-10 predictor incurs 12.6% higher error than GDP while reducing storage overhead by 34.6%.

References

[1] K. Du Bois, S. Eyerman, and L. Eeckhout, "Per-thread cycle accounting in multicore processors," *ACM Transactions on Architecture and Code Optimization*, vol. 9, no. 4, pp. 1–22, 2013.

[2] X. Zhao, M. Jahre, and L. Eeckhout, "HSM: A hybrid slowdown model for multitasking GPUs," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2020, pp. 1371–1385.

[3] L. Subramanian, V. Seshadri, A. Ghosh, S. Khan, and O. Mutlu, "The application slowdown model: Quantifying and controlling the impact of inter-application interference at shared caches and main memory," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2015, pp. 62–75.

[4] M. Jahre and L. Eeckhout, "GDP: Using dataflow properties to accurately estimate interference-free performance at runtime," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 2018, pp. 296–309.

[5] S. Eyerman and L. Eeckhout, "System-level performance metrics for multiprogram workloads," *IEEE micro*, vol. 28, no. 3, 2008.

[6] C. Luque, M. Moreto, F. J. Cazorla, R. Gioiosa, A. Buyuktosunoglu, and M. Valero, "ITCA: Inter-task conflict-aware CPU accounting for CMPs," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2009, pp. 203–213.

[7] M. K. Qureshi and Y. N. Patt, "Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2006, pp. 423–432.

[8] P. Salvesen, "Predicting interference-free performance with linear model trees," Master's thesis, Norwegian University of Science and Technology, 2020.

[9] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Optimizing NUCA organizations and wiring alternatives for large caches with CACTI 6.0," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2007, pp. 3–14.

[10] E. Perelman, G. Hamerly, and B. Calder, "Picking statistically valid and early simulation points," in *Proc. of the Int. Conf. on Parallel Architectures and Compilation Techniques (PACT)*, 2003, pp. 244–255.