UNIVERSITÄT AUGSBURG
Universitätsbibliothek

OPEN ACCESS
OPUS AUGSBURG

Fast and precise cache performance estimation for out-of-order execution

Roeland J. Douma, Sebastian Altmeyer, Andy D. Pimentel

Angaben zur Veröffentlichung / Publication details:

Douma, Roeland J., Sebastian Altmeyer, and Andy D. Pimentel. 2015. "Fast and precise cache performance estimation for out-of-order execution." In *DATE '15: Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition, 9-13 March 2015, Grenoble, France*, edited by Wolfgang Nebel and David Atienza, 1132–37. Piscataway, NJ: IEEE. https://doi.org/10.7873/DATE.2015.0066.

Nutzungsbedingungen / Terms of use:

licgercopyright

Dieses Dokument wird unter folgenden Bedingungen zur Verfügung gestellt: / This document is made available under these conditions:
Deutsches Urheberrecht
Weitere Informationen finden Sie unter: / For more information see:
https://www.uni-augsburg.de/de/organisation/bibliothek/publizieren-zitieren-archivieren/publiz/

Fast and Precise Cache Performance Estimation for Out-Of-Order Execution

Roeland J. Douma, Sebastian Altmeyer, Andy D. Pimentel
Computer Systems Architecture Group, University of Amsterdam, Netherlands
{r.j.douma, altmeyer, a.d.pimentel}@uva.nl

***Abstract*—Design space exploration (DSE) is a key ingredient of system-level design, enabling designers to quickly prune the set of possible designs and determine, e.g., the number of the processing cores, the mapping of application tasks to cores, and the core configuration such as the cache organization. High-level performance estimation is a principle component of any system-level DSE: it has to be fast and sufficiently precise. Modern out-of-order architectures with caches pose a significant problem to this performance estimation process, as no simple one-to-one mapping of the number of cache misses and resulting cycle time exists.**

We present a high-level cache performance-estimation framework for out-of-order processors. Evaluation shows that our prediction method is on average 15 times faster than cycle-accurate simulation, while our estimates only show an average error of below 3.5%, reduce the pessimism of a naive high-level performance estimation by around 66%, and still maintain a high fidelity. Our approach thus enables quick yet accurate performance estimation and extends the applicability of system-level DSE to out-of-order processors with caches.

I. Introduction

As the complexity of modern embedded systems increases, the development of these systems becomes more and more challenging. Design space exploration (DSE) methods are thus used to cope with the increasing complexity, to speed-up the development process and thus, to reduce the time-to-market. Automated exploration tools quickly prune the set of possible architectures and select the set of (pareto) optimal candidate architectures according to multiple objectives, such as performance, costs, size and power consumption. The system specification automatically selected by the exploration tools range from high-level information such as the number and the type of the processing cores or the mapping of application tasks to cores to more low-level specifications such as the cache hierarchy configuration. As the pruning of the set of candidate architectures is already required at an early development stage, a high-level performance estimation of the target application is necessary. Such an estimation has to fulfil two opposing requirements: it has to be fast (to evaluate a sufficient set of candidates) and precise (to select the correct candidates).

Caches are nowadays an integral component of most embedded systems where performance is critical. On the downside, caches require a significant amount of area on the chip and can increase the system's energy consumption [1]. Choosing the right cache size and right cache parameters is thus a paramount task of any DSE, and the aforementioned high-level performance estimation must account for caches.

In case of processors with in-order execution, it is sufficient to compute the number of hits and misses as a simple one-to-one mapping from misses to cycles exist. To this end, an abundance of cache miss estimation techniques, such as stack-distance histograms and cache-miss equations, exist and fast and accurate performance estimation is available.

In contrast, modern out-of-order architectures with caches pose a significant problem to the performance estimation, as no simple one-to-one mapping of the number of cache misses and cycles exist: the actual address trace can change depending on the selected cache configuration and pipelining of outstanding memory requests change the average memory latency. A naive estimation of the number of cycles to execute an application purely based on the number of hits and misses (as valid for in-order processors) thus provides unreliable results. Hybrid approaches using source-code instrumentation and relatively fast source-level simulation exist, but typically are limited to in-order execution. Consequently, the currently available performance estimation techniques are either precise (based on slow cycle-accurate simulation) or fast (based on hit/miss ratios), but not both at the same time.

We present a cache performance estimation framework for out-of-order processors. The framework is split in two phases: a setup phase and a DSE-phase. In the setup-phase, we perform cycle-accurate simulations for two cache configurations, extract stack-distance histograms and the *memory-overlap* of the target application. This information serves as input for the DSE-phase which is then used to quickly estimate the performance of the target application for the candidate cache configurations. This two-step approach thus enables quick yet accurate performance estimation to be used for system-level DSE.

Our framework improves upon related work in the following aspects: The framework is

- fast during the actual DSE using a two-phase approach,
- precise as it explicitly models the effect of out-of-order execution on the memory performance, and
- agnostic of the target architecture as it only relies on the existence of a cycle-accurate simulator, but does not require an in-depth analysis of the hardware.

The paper is structured as follows: Section II reviews the related work and Section III introduces the required terminology and notation. In Section IV, we present the performance-estimation framework and evaluate the proposed techniques in Section V. Section VI concludes the paper.

II. Related Work

The estimation of cache performance has been an intensive subject of research in the past decades. We distinguish two types of approaches: static analysis and simulation based approaches.

The two most prominent examples of the former type are the cache-miss equations by Ghosh et al. [2], and the stack-distance computation [3]. Cache-miss equations provide a high-level representation of the cache behaviour. They are used in compiler frameworks to estimate the effect of optimizations. Estimation techniques based on the stack distance [3] provide

more accuracy, but suffer from the memory overhead. Several methods have been proposed to enable an efficient computation [4]–[7] or approximation [8] of the stack distance. Stack distances and cache miss equations, however, only provide the number of misses and hits for different cache configurations. While this is sufficient for simple in-order architectures that stall in case of a cache miss, the performance of out-of-order architectures is not sufficiently described by this metric alone.

Simulation techniques provide an alternative to the static analyses, but are either restricted to in-order processors [9]–[11] or to the simulation of only one cache configuration at a time, thus resulting in an unacceptable execution time [12].

A different set of performance estimation techniques is source-level simulation based on instrumentation [13]–[15]. The source code is instrumented with memory access annotations and timing information, enabling the performance estimation via source-code simulations instead of using a cycle-accurate low-level simulator. These approaches thus still require explicit cache simulation and do typically not take the effect of out-of-order execution into account. Only recently, Plyaskin et al. [16] presented a source-level simulation approach that accounts for out-of-order execution behavior. However, this approach still requires explicit cache simulation.

To the best of our knowledge, no fast and accurate high-level performance estimation for out-of-order processors with caches exist. DSE techniques thus either do not support these systems or have to restrict precise performance estimation to a small subset of the design space [17].

III. Background and Notation

In this section, we introduce the necessary notation and concepts needed in the remainder of the paper. We assume a target architecture with disjoint instruction and data caches that are both connected to main memory via a shared bus.

A. Caches and Address Traces

We concentrate in the following on set-associative caches with LRU replacement: i.e, caches that are partitioned into s *cache sets*, where each memory block of size l (i.e., cache line size is l) maps to exactly one of the cache sets. Each cache set in turn may contain up to k different memory blocks at once, where k is referred to as the *associativity* of the cache. The *cache size* is thus given by $l \cdot s \cdot k$.

We will later on use the concept of a *perfect cache*, which denotes a fully associative cache that is large enough to store the complete data of a task/program. A perfect cache thus only exhibits cold misses, but no conflict or capacity misses [5]. Note that direct-mapped and fully associative caches are special cases of set-associative caches with $k = 1$ or $s = 1$, respectively.

A cache configuration or *cache setup* ζ is a tuple (k, s) consisting of the associativity and the number of cache sets. For the sake of simplicity, we assume a fixed line size l throughout the paper. The set of all cache setups is denoted by $\mathbb{C}$ with the perfect cache pc as special instance of a cache setup, i.e.: $\mathbb{C} = \{(k, s) | (k, s) \in \mathbb{N})\} \cup \{pc\}$. Architectural restrictions or a pre-selection of candidate cache setups allows us to reduce the complete set of cache setups $\mathbb{C}$ and to focus only on a subset $C \subset \mathbb{C}$.

A cache address trace T of size n is an ordered sequence $[m_1, \ldots, m_n]$ of memory blocks $m_i \in \mathbb{M}$, where $\mathbb{M}$ is the set of all memory blocks. The set of all traces is given by $\mathbb{T}$. The *stack distance* sd_s (s denotes the number of sets) [3] is the number of distinct memory elements mapping to cache set s' accessed in between an access to a memory element that also maps to $s'$and the previous access to the same memory element, with ∞ denoting that there is no prior access:

$$sd_s : \mathbb{M} \times \mathbb{T} \rightarrow \mathbb{N} \cup \{\infty\}$$

$$sd_s\left(m_l, [m_1, \ldots m_{l-1}]\right) = \begin{cases} |\{m_j | i < j < l \wedge \text{cs}(m_l) = \text{cs}(m_j)\}| & \text{if } m_i = m_l \\ & \wedge \forall_{i<j<l} : m_l \neq m_j \\ \infty & \text{else} \end{cases} \quad (1)$$

where $\text{cs}(m)$ denotes the cache set to which memory element m maps. The condition $\text{cs}(m_l) = \text{cs}(m_j)$ thus ensures that the stack distance of element m_l only considers memory elements that compete for the same cache set as m_l.

Table I summarizes the definitions used throughout the paper. To simplify the notation, we omit the subscripts whenever the parameters are sufficiently defined by the context.

TABLE I. Cache parameters and domains

Line size	$l \in \mathbb{N}$
Associativity	$k \in \mathbb{N}$
Number of sets	$s \in \mathbb{N}$
Cache size	$l \cdot s \cdot k = CS \in \mathbb{N}$
Cache setup	$\zeta = (k, s) \in \mathbb{C}$
Candidate cache setups	$C \in \mathbb{C}$
Perfect cache	pc
Memory Latency	$\gamma \in \mathbb{N}$
Number of cycles on trace T and cache setup ζ	${}^{cyc}\zeta,T$

B. Stack Distance Histogram Computation

To derive the number of cache misses, we will later use stack distance histograms [4]–[7] within our performance estimation framework. These stack distance histograms only depend on the number of cache sets and the cache line size (which we assume to be fixed). It is therefore sufficient to compute one stack distance histogram per distinct number of cache sets, irrespective of the associativity of the cache configuration. The computation of several stack distance histograms for different numbers of cache sets can be done in parallel.

Consider for example the following address trace

$$\underbrace{A, B, B', A', B'', A''}_{\text{cold misses}}, A, B, A', B', B, A''$$

where we assume that all addresses map to distinct cache lines. The first 6 accesses are thus cold misses in any cache setup and have a stack distance of ∞.

In a fully-associative cache, three accesses have a stack distance of 5 (A, B, B'), and two accesses have a stack distance of 4 (A', A''). Only the last access to B has a stack distance of 2. In case of the set-associative cache, we assume that all references to A, A', A'' map to the same cache set, but to distinct cache lines. The same holds for B, B', B''. Moreover, we assume that A, B all map to distinct cache sets. Thus, to determine e.g. the stack distance for A, only accesses to A, A', and A'' need to be taken into account. The number of cold misses is unchanged, five accesses have a stack distance of 2 (A, B, A', B', A''). The last access to B has a stack distance of 1. This example shows that the stack distance can significantly change if the cache setup changes.

IV. Cache Performance Estimation

System-level DSE requires fast, yet precise estimation of the execution time of the target application on the candidate architecture. Exhaustive evaluation of all cache configurations

using simulation is thus not an option; the simulation runs would consume more time than budgeted for the complete DSE.

We propose a method that relies on a minimal number of cache simulations needed to predict the performance of all candidate cache setups from the set C, and a limited number of trace analyses. We divide the set of candidate setups into q congruent sets C_s, where q is the number of distinct numbers of sets within C:

$$q = |\{s|(_, s) \in C\}| \tag{2}$$

C_s contains all candidate cache setups with s cache sets:

$$C_s = \{\zeta|\zeta = (_, s) \in C\} \tag{3}$$

Instead of performing $|C|$ simulations, our approach relies on the results of only 2 *simulations in total* (one to extract the address trace and one to estimate the $memory\ overlap$, a performance metric explained in detail in Section IV-B) and q trace analyses, which can be performed in parallel.

The first simulation assumes a perfect cache. This simulation provides us with the number of cycles, cyc_{pc}, and the address trace T. The second simulation is performed for a reference setup $\zeta' \in C$, as explained later on.

The q trace analyses are needed to compute a histogram of the stack distances for a given s: $H_s : \mathbb{N}^\infty \to \mathbb{N}$

$$H_s(x) = |\{i|sd_s(m_i, T) = x\}| \tag{4}$$

Using the histograms, the number of cold misses for cache setup $\zeta = (k, s)$ is given by:

$$miss_\zeta^{cold} = H_s(\infty) \tag{5}$$

The number of conflict or capacity misses by:

$$miss_\zeta^{warm} = \sum_{k' \geq k \wedge k' \neq \infty} H_s(k') \tag{6}$$

And the total number of misses is given by:

$$miss_\zeta = miss_\zeta^{cold} + miss_\zeta^{warm} \tag{7}$$

A. In-Order Execution

In an in-order processor, a cache miss means that the processor stalls until this cache line is retrieved from main memory. This means that the number of cycles required for a given cache setup can be approximated as follows:

$$cyc_\zeta = cyc_{pc} + miss_\zeta^{warm} \cdot \gamma \tag{8}$$

Where γ is the nominal memory latency of the system.

For the sake of simplicity, we assume unrestricted and immediate access to all potentially shared resources such as a bus and only concentrate on the execution cycles of the target application in isolation. Even though this assumption may be considered a strong restriction, it allows us to focus on the problem at hand and is common in the related work.

B. Out-of-Order Execution

Out-of-order processors do not stall in case of a cache miss, but execute other instructions while waiting for data. To fully exploit the advantages of out-of-order execution many processors also allow multiple outstanding memory requests to be handled simultaneously. Consequently, the effective memory latency $\hat{\gamma}$ is often significantly smaller than the actual time to transport data from main memory to the cache; and the number of cache misses alone is not sufficient to estimate the performance anymore.

1) Illustrated Example: We illustrate the effect of out-of-order execution on the memory latency with an example depicted in Figure 1. We assume two cache configurations: Configuration 1 is a direct mapped cache with 2 sets; Configuration 2 has an associativity of 2 and also 2 sets. The second cache is thus twice as large as the first one.

	Address trace												
	A	B	B'	A'	B''	A''	A	B	A'	B'	B	A''	Sum
DM	M	M	M	M	M	M	M	M	M	M	M	M	
	150		150		150		150		150		150		900
2 Way	M	M	M	M	M	M	M	M	M	M	H	M	
	150		150		150		150		150		0	100	850
Perfect	M	M	M	M	M	M	H	H	H	H	H	H	
	150		150		150		0	0	0	0	0	0	450

Fig. 1. Visualisation of the address trace and the number of hits (H), misses (M) and the number of cycles for three different caches.

We consider the address given in Section III-B and we assume again that all letters map to pairwise distinct cache sets, but all the accented letters map to different cache lines (but the same cache set). The cache is initially cold, i.e., empty, which means that the first 6 accesses in this trace are all misses for all cache configurations. The trace with corresponding hits, misses and cycle counts is shown in Figure 1.

The memory latency is 100 cycles and two outstanding memory requests can be overlayed with a combined memory latency of 150 cycles (instead of 200 for an in-order processor). Note that this is an oversimplification purely for the purpose of the example. For the sake of simplicity, we also assume that if two memory requests overlap they have to finish before another request can start.

Although Figure 1 shows the entire address trace, for this example we will only focus on the final part of the trace, since the first 6 acceses are all cold misses. The processor with the direct mapped cache has 6 cache misses and spends a total of 450 cycles waiting on memory. The average memory latency is thus 75 cycles. The processor with the 2-way set associative cache has only 5 cache misses but waits for a total of 400 cycles for data. The average memory latency is thus 80 cycles.

The example illustrates the imprecision of Eq. (8) when applied to out-of-order processors. The nominal latency γ is 100 cycles while we observe values of 75 and 80 cycles instead. The naive estimation of the execution cycles is thus not valid for DSE. Instead of the simple one-to-one mapping of the number of misses and the execution time, the average memory latency depends on the actual cache configuration.

2) Effective Memory Latency: As Eq. (8) is restricted to in-order processors, we propose an adapted version for out-of-order execution. We multiply the number of warm misses $miss_\zeta^{warm}$ of cache configuration ζ with the effective memory latency $\hat{\gamma}$ instead of the nominal latency γ:

$$cyc_\zeta = cyc_{pc} + miss_\zeta^{warm} \cdot \hat{\gamma}_\zeta \tag{9}$$

Unfortunately, the effective memory latency is not constant, but depends on the cache configuration as we have seen in the example.

In the following, we show how we can compute an approximation of execution cycles cyc_ζ for cache configuration ζ based on the *memory overlap*, a metric which we compute using a reference cache configuration ζ'.

We first compute the miss-rate of the warm misses for the reference configuration ζ'

$$miss_rate_{\zeta'}^{warm} = \left(\frac{miss_{\zeta'}^{warm}}{n} \right) \tag{10}$$

where n is the size of the address trace, i.e., the number of memory accesses, and the average number of cycles per

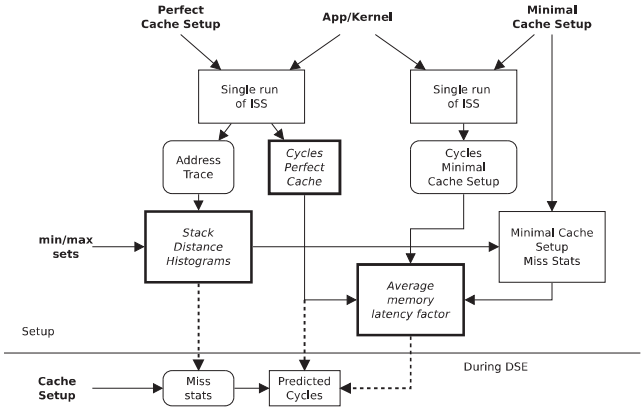

Fig. 2. Overview of the framework. The text in bold represents input. The boxes with thicker borders are generated data used in the DSE phase.

warm miss:

$$cycles_miss_{\zeta'}^{warm} = cyc_{\zeta'} \cdot miss_rate_{\zeta'}^{warm} \tag{11}$$

where $cyc_{\zeta'}$ is obtaind via simulation with cache setup ζ'.

Furthermore, we need the deviation (or error) in the number of cycles when assuming the nominal memory latency γ:

$$error_{\zeta'} = \left(cyc_{pc} + miss_{\zeta}^{warm} \cdot \gamma\right) - cyc_{\zeta'} \tag{12}$$

The value $error_{\zeta}$ thus provides the pessimism when using Eq. (8). We note that the effective memory latency is always upper bounded by the nominal memory latency, which means that $error_{\zeta}$ is always non-negative.

We now combine Eq. (12) and (11), i.e., the *error* and the *average number of cycles per warm miss* for the reference configuration ζ', to compute the *memory overlap*:

$$memory_overlap_{\zeta'} = \frac{error_{\zeta'}}{cycles_miss_{\zeta'}^{warm}} \tag{13}$$

The memory overlap is an indicator for the number of memory accesses that can happen simultaneously.

Using the memory overlap for ζ', we can approximate cyc_{ζ}, for any $\zeta \in C$, in an alternative fashion using Eq. (13):

$$cyc_{\zeta} \approx \frac{cyc_{pc} + miss_{\zeta}^{warm} \cdot \gamma}{1 + miss_rate_{\zeta}^{warm} \cdot memory_overlap_{\zeta'}} \tag{14}$$

We can now also approximate the effective memory latency $\hat{\gamma}_{\zeta}$ for cache configuration ζ by rewriting Eq. (9) as follows:

$$\hat{\gamma}_{\zeta} \approx \frac{\left(\frac{cyc_{pc}+miss_{\zeta}^{warm}\cdot\gamma}{1+miss_rate_{\zeta}^{warm}\cdot memory_overlap}\right) - cyc_{pc}}{miss_{\zeta}^{warm}} \tag{15}$$

Since Eq. (14) only requires the number of cycles for the perfect cache, the memory overlap for config ζ' and the number of misses for config ζ, the only information specific to the cache configuration ζ can thus be computed using the stack distances. We note that $\gamma = \hat{\gamma}$ holds if the error is zero.

It remains to be discussed how to select the reference configuration ζ'. We have seen that the maximal error typically occurs for small setups (see Figure 1), along with the highest number of misses. We therefore select the smallest cache setup as the reference to compute the memory overlap, i.e. $\zeta' = \zeta_{min}$ where we define the smallest cache setup as:

$$\zeta_{min} = (k, s) \in C \tag{16}$$

with $\forall (k', s') \in C \colon k \cdot s \leq k' \cdot s'$

3) Example Revisited: We now revisit the example depicted in Figure 1 to illustrate the presented concepts. The figure shows that a run of the simulator with a perfect cache takes 450 cycles and the run of the simulator with a direct mapped cache takes 900 cycles (i.e., $cyc_{\zeta_{min}}$). We see using Eq. (5) and (6) that the execution with configuration 1, direct mapped, results in 6 cold misses and 6 warm misses. Thus, the warm miss-rate as given by Eq. (10) is $\frac{6}{12}$. Using the miss-rate, the number of cycles obtained from simulation (i.e., $cyc_{\zeta_{min}}$) and Eq. (11), we calculate the fraction of cycles associated with the warm misses: $900 \cdot \frac{6}{12} = 450$. Furthermore, we calculate the error using Eq. (12): $(450 + 6 \cdot 100) - 900 = 150$ and the $memory\ overlap$ as described in Eq. (13): $\frac{150}{450} = \frac{1}{3}$. By combining the $memory\ overlap$ and the number of cycles of the perfect cache, we calculate cyc_{ζ} for cache configuration 1 using Eq. (14) as follows $cyc_1 \approx \frac{450+6\cdot100}{1+\frac{6}{12}\cdot\frac{1}{3}} = 900$ which provides an exact estimate (which was to be expected as we have used configuration 1 as reference configuration).

Using the same stack distance histogram, we also calculate that setup 2 results in 6 cold misses, 5 warm misses and 1 hit. Using Eq. (10), we get the following miss-rate $miss_rate_2^{warm} = \frac{5}{12}$ and using Eq. (14), we estimate the number of cycles as follows: $cyc_2 \approx \frac{450+5\cdot100}{1+\frac{5}{12}\cdot\frac{1}{3}} \approx 834$, which is only off by 16 cycles (error of less than 1.9%). Using the naive approach from Eq. (8), we get the following estimate $cyc_2 = 450 + 5 \cdot 100 = 950$ which is 100 cycles off (error rate of 11.7%).

C. Framework

Our framework to estimate the cache performance consists of two phases, the setup and the DSE phase, as shown in Figure 2. Within the setup phase, we perform the two simulations of the target application. One simulation assuming a perfect cache, and one assuming the minimal (i.e. reference) cache configuration ζ_{min}. These runs provide us with the number of execution cycles for these two cache setups (cyc_{pc}, $cyc_{\zeta_{min}}$) and the address trace. The address trace is then input to our stack histogram computation, where we derive a stack histogram for each set C_s (see Eq. (3)).

The last step within the setup phase consists of calculating the miss statistics for the reference cache setup ζ_{min}, i.e., the miss-rate (Eq. (10)), the cycles per miss (Eq. (11)), the error (Eq. (12)) and the memory overlap (Eq. (13)).

Input to the DSE phase is thus only: (i) the stack distance histograms H_s, (ii) the number of cycles for the perfect cache cyc_{pc}, and (iii) the memory overlap $memory\ overlap_{\zeta_{min}}$ for the minimal configuration. This information is sufficient to estimate the number of cycles cyc_{ζ} for any cache configuration we are interested in within negligible time overhead using Eq. (14).

V. Evaluation

In this section, we evaluate the effectiveness of our performance estimation framework. We focus in particular on (i) the accuracy of our predictions and the improvement with respect to the naive estimation from Eq. (8) (both in terms of predicted misses and predicted cycles), (ii) the fidelity in the predicted order of candidate configurations, and (iii) the speed-up compared to the use of cycle-accurate simulation.

The design space, i.e., the set of candidate cache configurations is as follows. We assume a line size of 32 bytes. The size of the cache ranges from 1 to 32 kB with a range of the

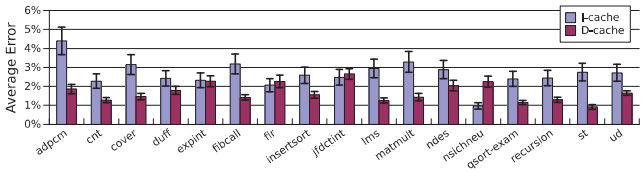

Fig. 3. Average missrate prediction error (over the complete range of cache configurations). The error bars show the standard deviation.

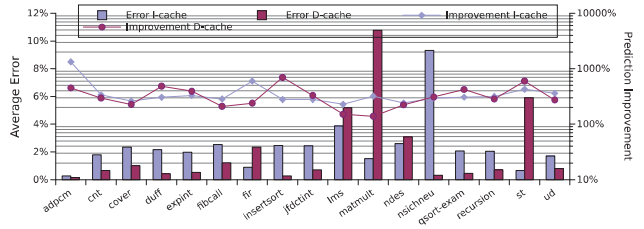

Fig. 4. Average error (over the complete range of cache configurations) in the predicted cycle count. The lines show the improvement over the naive prediction method from Eq. (8). Note that the right y-axis is in log scale.

associativity from 1 to 32. This results in 11 distinct numbers of caches sets and thus the computation of 11 stack distance histograms.

As benchmarks, we use the Mälardalen benchmark suite [18] (except for *bsort100* due to a segmentation fault and *sqrt* which lacks a main-function), which gives 33 benchmarks and thus 66 scenarios in total when separately considering instruction and data caches. Due to space limitations, the graphs in this section only show a representative selection of the benchmarks, which contains the corner cases and spikes of the results. This means that results for the missing benchmarks lie in between those presented in the graphs. The gem5 [12], [19] instruction set simulator (ISS) serves as the cycle-accurate simulation within the framework (i.e. to obtain the cycle counts for the perfect and minimal cache configuration) and provides the cycle counts for all benchmarks and all cache configurations to evaluate the precision of our estimates. The simulator models a 32-bit ARMv7-A with a 5-stage pipeline which supports out-of-order execution. All simulations have been performed on an Intel QuadCore i7-2600.

A. Misses

Out-of-order execution may influence the actual address trace, such that the order of accesses depends on the actual cache set-up. We therefore first evaluate the number of misses predicted by our method (which is based on the address trace of the perfect cache) compared to the actual number of misses derived by the simulation (assuming the specific cache setup). We calculate the error using $error = \left|1 - \frac{miss_{pc,\zeta}}{miss_{\zeta}}\right|$ where $miss_{pc,\zeta}$ is the number of misses predicted by the trace from the perfect cache. The average error is shown in Figure 3. The figure shows that the average error in the number of misses of our estimation framework is below 5%. It also shows that the I-cache is more strongly influenced by the out-of-order behaviour than the D-cache. We note that in an in-order processor, the order of the memory operations would not have changed and we would have an error of 0%.

B. Validity

In Figure 4, we have visualized the average error in our predicted number of cycles compared to the actual number of cycles obtained using simulation. For readablity we have

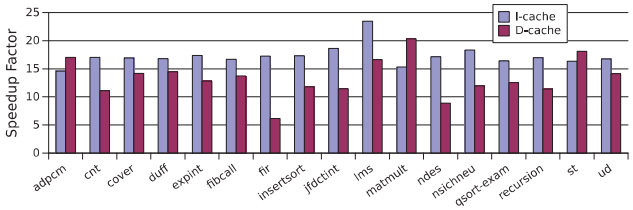

Fig. 5. Speed-up of our method compared to simulation.

omitted the error bars from Figure 4, which would have shown an average standard deviation of 2%.

We calculate the error using $error = \left|1 - \frac{cyc_{pc,\zeta}}{cyc_{\zeta}}\right|$ where $cyc_{pc,\zeta}$ is the number of cycles we predict. We have determined the $memory\ overlap$ using the smallest cache configuration (ζ_{min}) from our set of candidate configurations. This is a 1 kB cache with associativity 1, i.e., direct mapped.

The figure shows that for most benchmarks our predictions are close to the obtained values using simulation: From the 66 scenarios (of which only 34 are shown in the graphs due to the limited space), only 4 exhibit an average error of more than 5% and the majority (61) has an average error of below 3%. This indicates that our framework yields good results in predicting the number of required cycles.

The line graphs in Figure 4 show the improvement of our estimations compared to the naive approach from Eq. (8) on the secondary y-axis. From these results we can see that our approach is on average a factor 3.95 better for the prediction of the D-cache and a factor 6.95 for the I-cache.

The bars in Figure 4 show a large error of our predicitions for two benchmarks ($matmult$ and $nsichneu$). We have observed for these two benchmarks significant performance differences for the minimal cache setup (ζ_{min}) compared to other cache setups (not shown). This results in a stark under- or over-approximation of the $memory\ overlap$, which in turn results in a larger error of our predictions. However, our framework still improves over the naive approach.

Note that we are agnostic to the target architecture and did not perform an in-depth analysis of the hardware. We only require the availablity of a cycle-acurate ISS to predict the number of required cycles.

C. Fidelity

As we have seen, our estimates are very close to the actual number of misses and execution cycles, but not exact. However, for early DSE it is often equally important to preserve the fidelity than to have precise predictions [20]. To show the fidelity of the results we use two common metrics: Spearman's ρ [21] and Kendall's τ [22]. In short, Spearman's ρ compares the rank of elements between two datasets. Where a value of 1 means the ranks match and a value of -1 means the ranks are reversed. Kendall's τ compares the relation of one item in the set to another. Again a value of 1 means the ordering matches and a value of -1 means a reverse ordering.

For the I-cache, the average value of Spearman's τ for the whole benchmark suite is 0.994 with a standard deviation of 0.004. The average of Kendal's ρ is 0.983 with a standard deviation of 0.017. For the D-cache, the average of Spearman's ρ is 0.956 with a standard deviation of 0.038. The average of Kendal's τ is 0.924 with a standard deviation of 0.062.

Our method is an approximation and is subject to an error margin, as can be seen from the figures. For cache setups that

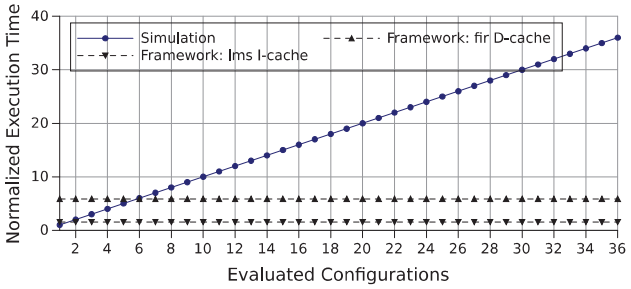

Fig. 6. DSE with our high-level estimation versus simulation-based DSE.

perform within 1% of each other, our method can therefore not provide a proper judgement. This, however, is acceptable for early DSE as other design-criteria (such as hardware costs or energy consumption) are likely to outweigh such a minor difference. To take this into account, we also investigated the effect of allowing a 1% error in the number of calculated cycles in the calculation of the fidelity metrics. This means we allow shuffling of the ordering if the results are within a 1% range of each other. For the I-cache predictions this improves Spearman's ρ to an average of 0.999 with a standard deviation of 0.003 and Kendal's τ to an average of 0.990 with a standard deviation of 0.012. The D-cache experiments show similar behaviour with Spearman's ρ improved to an average of 0.991 with a standard deviation of 0.015. Kendal's τ is improved to an average of 0.954 with a standard deviation of 0.042. These results indicate that there are several cache configurations with very similar performance.

D. Speed-up

Finally, we evaluate the execution time of our approach compared to exhaustive simulation using gem5. We therefore compare the total execution time of the simulation of the 36 cache setups with the execution time of our framework, including both the setup-phase (executed once in total) and DSE-phase (executed once for each cache setup). In Figure 5, the speed-up of our approach is shown per benchmark. We have measured speed-ups between $6x$ and $23x$ with an average of $15x$. This shows that for all benchmarks, we are significantly faster than a complete simulation of all cache configurations.

Figure 6 visualizes the point at which the overhead of the setup-phase amortizes and our approach is faster than simulation. The dotted blue line shows the normalized execution time when exploring cache configurations using simulation. The line with the upward pointing arrows represents the normalized time our framework requires for the benchmark fir when predicting D-cache, i.e., the minimal speed-up. The line with the downward pointing arrows represents the benchmark lms when predicting I-cache, i.e., the maximum speedup. The break-even points show the number of cache configurations where the prediction using our framework and using the simulations require the same execution time. The break-even points are at 6 cache configurations in case of fir and D-cache exploration, and at 2 cache configurations in case of lms and I-cache exploration. The average break-even point lies at 2.37, which is to be expected as our framework requires two runs of the simulator and limited additional overhead to compute the metrics.

VI. Conclusion

In this paper, we have presented a framework to estimate the performance of systems with out-of-order processors using caches. Our framework consists of two phases: a setup and a DSE phase. The setup-phase performs two cycle-accurate simulations and a trace analysis to extract relevant information metrics, after which the DSE-phase then derives accurate performance estimations of the target application for architectures with different cache setups within negligible execution time. Evaluation has shown that the framework is precise both in terms of estimated number of cache misses (average error of less than 4%) and estimated execution cycles (average error of less than 3.5%). Our framework improves upon a naive estimation based purely on the number of misses by a factor of 3.95 in case of I-caches and by a factor of 6.95 in case of D-caches. The fidelity in the ordering of candidate cache setups is near optimal with an error of around 5%. We furthermore have observed an average speed-up of $15x$ compared to cycle-accurate simulation, which represents the only precise alternative. Our framework thus enables quick yet accurate performance estimation for out-of-order processors with caches to be used for system-level DSE.

Acknowledgements

This work was partially funded by the NWO SES Project Nr. 647.000.002 and the ICT COST Action IC1202 TACLe.

References

[1] J. Montanaro, R. T. Witek, K. Anne, A. J. Black, E. M. Cooper, D. W. Dobberpuhl, P. M. Donahue, J. Eno, and D. Kruckemyer, "A 160-mhz, 32-b, 0.5-w cmos risc microprocessor," *IEEE Journal of Solid-State Circuits*, vol. 31, pp. 1703–1714, 1996.

[2] S. Ghosh, M. Martonosi, and S. Malik, "Cache miss equations: A compiler framework for analyzing and tuning memory behavior," *ACM Trans. Program. Lang. Syst.*, vol. 21, no. 4, pp. 703–746, Jul. 1999.

[3] R. Mattson, J. Gecsei, D. Slutz, and I. Traiger, "Evaluation techniques for storage hierarchies," *IBM Systems Journal*, vol. 9, no. 2, pp. 78–117, 1970.

[4] B. T. Bennett and V. Kruskal, "Lru stack processing," *IBM Journal of Research and Development*, vol. 19, no. 4, pp. 353–357, July 1975.

[5] M. D. Hill and A. J. Smith, "Evaluating associativity in cpu caches," *IEEE Trans. Comput.*, vol. 38, no. 12, pp. 1612–1630, Dec. 1989.

[6] R. A. Sugumar and S. G. Abraham, "Set-associative cache simulation using generalized binomial trees," *ACM Trans. Comput. Syst.*, vol. 13, no. 1, pp. 32–56, Feb. 1995.

[7] G. Almási, C. Caşcaval, and D. A. Padua, "Calculating stack distances efficiently," *SIGPLAN Not.*, vol. 38, no. 2, pp. 37–43, Jun. 2002.

[8] J. J. Pieper, A. Mellan, J. M. Paul, D. E. Thomas, and F. Karim, "High level cache simulation for heterogeneous multiprocessors," in *DAC '04*, July 2004, pp. 287–292.

[9] J. Elder and M. Hill, "Dinero IV Trace-Driven Uniprocessor Cache Simulator," http://pages.cs.wisc.edu/ markhill/DineroIV/.

[10] W. Zang and A. Gordon-Ross, "T-spacs–a two-level single-pass cache simulation methodology," *IEEE Trans. on Computers*, vol. 62, no. 2, pp. 390–403, 2013.

[11] A. Janapsatya, A. Ignjatović, and S. Parameswaran, "Finding optimal l1 cache configuration for embedded systems," in *ASP-DAC '06*, Jan 2006, pp. 796–801.

[12] "The gem5 simulator system," http://www.gem5.org.

[13] Z. Wang and J. Henkel, "Fast and accurate cache modeling in source-level simulation of embedded software," in *DATE '13*, 2013, pp. 587–592.

[14] S. Stattelmann, G. Gebhard, C. Cullmann, O. Bringmann, and W. Rosenstiel, "Hybrid source-level simulation of data caches using abstract cache models," in *DATE '12*, March 2012, pp. 376–381.

[15] S. Chakravarty, Z. Zhao, and A. Gerstlauer, "Automated, retargetable back-annotation for host compiled performance and power modeling," in *CODES+ISSS'13*, Sept 2013, pp. 1–10.

[16] R. Plyaskin, T. Wild, and A. Herkersdorf, "System-level software performance simulation considering out-of-order processor execution," in *International Symposium on System-on-Chip*, Oct 2012.

[17] S. Eyerman, L. Eeckhout, and K. De Bosschere, "Efficient design space exploration of high performance embedded out-of-order processors," in *DATE '06*, vol. 1, March 2006, pp. 1–6.

[18] J. Gustafsson, A. Betts, A. Ermedahl, and B. Lisper, "The Mälardalen WCET benchmarks – past, present and future," B. Lisper, Ed. Brussels, Belgium: OCG, Jul. 2010, pp. 137–147.

[19] N. Binkert *et al.*, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[20] H. Javaid, A. Ignjatovic, and S. Parameswaran, "Fidelity metrics for estimation models," in *ICCAD '10*, Nov 2010, pp. 1–8.

[21] C. Spearman, "The proof and measurement of association between two things," *American Journal of Psychology*, vol. 15, pp. 88–103, 1904.

[22] M. Kendall and J. D. Gibbons, *Rank Correlation Methods*, 5th ed. A Charles Griffin Title, September 1990.