Interleaved Multi-Bank Scratchpad Memories: A Probabilistic Description of Access Conflicts

Andreas Tretter, Pratyush Kumar and Lothar Thiele
Computer Engineering and Networks Laboratory, ETH Zürich, 8092 Zürich, Switzerland
firstname.lastname@tik.ee.ethz.ch

ABSTRACT

Shared on-chip memory is common on state-of-the-art multi-core platforms. In a number of designs, memory throughput is enhanced by providing multiple independent memory banks and spreading consecutive memory addresses to these (interleaving). This can reduce, but not eliminate, the number of access conflicts. In this paper, we statically analyse the probabilities and frequencies of these access conflicts and calculate the expected throughput for various hardware configurations and software applications. Using two techniques – the classic occupancy distribution and a Markov model – we are able to explain most of the underlying conflict mechanisms and to provide accurate estimations. We present the practical consequences for hardware and software design and establish an intuitive understanding of the characteristics of interleaved memory architectures.

1. INTRODUCTION

Recent multi- and many-core platforms show a clear trend towards the use of *shared* and *fast* on-chip memory, with multiple separately accessible banks. This memory, sometimes referred to as *scratchpad* or *L1* memory, is a powerful and energy-efficient alternative to caches. Examples of such memory architectures are found in the Epiphany [1], P2012 [3], Kalray MPPA, TI Freescale or ARM multicore platforms.

The sharing of the memory modules, i.e. having a common address space for all processors, has some distinct advantages. Primarily, it improves the programmability, especially in terms of efficient communication between processors. It also efficiently allows for different processors to have different memory footprints. However, this sharing introduces the problem of *interference* amongst the processors. One solution to the interference problem is to increase the number of banks: More the banks, the lower the expected probability of multiple processors accessing the same bank. Apart from having scalability issues, this solution by itself cannot solve the problem. We also need to optimize the *address mapping* – the mapping of the logical addresses to physical locations.

Since long, several standard approaches to address mapping have been studied. These include (a) *contiguous* mapping, (b) *pseudo-random* mapping, and (c) *sequentially interleaved* mapping. In contiguous mapping, a large block of contiguous addresses are mapped onto the same memory bank. As an example, this mapping has been adopted in the Epiphany chip. In pseudo-random mapping, the address map is computed by a pseudo-random hash function. In effect, there is no specific pattern in which the addresses are distributed spatially across the memory banks. As an example, the cache lines of Xeon Phi have a pseudo-random mapping. In sequentially interleaved memory (SIM), the address space is divided into chunks of a certain number of words, and each subsequent chunk is assigned to a subsequent bank. Typically, this means that the low-order bits of the word address are assigned to the bank address. As an example, the P2012 platform is configured with SIM.

For fast memory modules, contiguous mapping is effective if different processors access different banks. In many applications this is not the case – multiple processors execute parts of the same application with large blocks of shared data. In such cases, the interference can be reduced with a pseudo-random mapping. The aim here is to avoid any memory hot-spots, without exploiting any specific application profiles. However, it is often argued that frequently occurring sequential address access patterns in real applications can be taken advantage of with sequentially interleaved mapping (SIM) instead of a pseudo-random mapping. The idea is that the memory accesses of two threads will after one collision always be displaced, causing no further conflicts. However, a general quantification of this *synchronisation effect* is missing.

The aim of this work is to quantitatively evaluate the properties and characteristics of SIM systems. In particular, we study the metric of *average throughput of the memory subsystem*. This requires developing accurate models and analysis methods which can compute the metric with SIM for specific applications and architectures.

An application is characterized by its memory access pattern, i.e. which address is accessed and when. We propose to abstract this complex characterization by two *stochastic parameters*: p_a, which represents the probability of the application accessing the memory in any given cycle, and p_{seq}, which represents the probability that the application accesses the address subsequent to its previous address. While this first-order abstraction may seem inadequately approximate, we demonstrate that with specific analysis methods we can accurately predict the average throughput for many real applications.

Given the stochastic parameters of an application, we propose two methods to predict its memory throughput. First, we propose to apply a modified version of the *occupancy distribution*. This provides an analytical expression for the throughput, but is only accurate when memory bank conflicts are not a dominating factor in the system (e.g. few processors and many banks). To extend the applicability we propose a second method, which relies on modelling and solving a *Markov system*, which is specifically designed to model the working of a SIM-based memory. Both these methods advance known results on analysis of interleaved memory, and provide design guidelines on choice of parameters.

To evaluate the accuracy of the proposed abstraction and analysis methods, we perform several experiments. Using the gem5 simulator [6], we model a multi-banked memory module accessed by multiple ARM-based processors. We consider different applications that are common in several benchmarks. We profile these applications, compute their stochastic parameters, and then ana-

DAC '15, June 07 – 11, 2015, San Francisco, CA, USA

ACM 978-1-4503-3520-1/15/06 ...$15.00
http://dx.doi.org/10.1145/2744769.2744861

lyse them with the two methods. Across experiments, we find that the Markov-based method computes a throughput that is within 2.5 % deviation of the measured throughput. For settings containing twice as many banks as cores, i.e. where waiting accesses do not play a dominating role, the error of the occupancy distribution method is within 5 %. We also observe that the SIM memory has a throughput up to 5% higher than the pseudo-random mapping. These experimental results weaken the argument that the synchronization effect alone is effective enough to overcome the disadvantages of SIM memory vis-a-vis pseudo-random mapping.

The rest of this paper is organized as follows. We define the system model and the problem statement in Section 3. In Section 4, the classic occupancy distribution and the model based on it will be presented. Section 5 does the same with the Markov based model. The models will then be evaluated experimentally in Section 6. Before, however, we will review other works on the same or related issues in the next section.

2. RELATED WORK

Interestingly, the main line of research in the domain of interleaved memory took place as early as the late 1960s and the 1970s, at the time using magnetic core memories. With memory accesses taking considerably longer than a processor clock cycle, the common simplification was to partition time into "access cycles", assuming that any access demand arrived at the beginning of a cycle and that every core accessed the memory in each such a cycle.

Some of the analysis models (e.g. [7, 8]) were conceived for architectures which we do not regard in this paper. Others (e.g.[16]) are computationally challenging and therefore only applicable to smaller clusters.

Two models turned out to be successful and were adopted in multiple publications. The first such model is that of Strecker [17], which assumes that every memory access will be to a random bank in each cycle, leaving out of consideration the pending accesses from previous cycles. This allows for a stateless analysis of the system and therefore for high analysability. [13] uses this model for further calculations.

The second popular model was presented in [2] and further detailed in [5] and [4]. The idea there is to use a Markov description of the system and to extract results from a steady state analysis. [14] provides a simpler and more accurate estimation for the results.

Many recent works use these two models for their applications; however, to the best of our knowledge, no publication has altered their basic assumptions. In this work, we revisit both the models and check if they can be applied to state-of-the-art multi-processor systems with fast on-chip memories. We take account of the fact that a processor will no longer access the memory in every cycle by extending the models by *memory access probabilities*. Also, we introduce a *sequential access probability* into the Markov model in order to reflect certain memory access patterns present in real applications. We believe that this is more relevant for systems with dedicated data memory than for the machines regarded in the aforementioned publications, which fetch instructions and data interchangeably from the same memory.

3. SYSTEM MODEL AND PROBLEM DEFINITION

3.1 Architecture Model

We consider a Harvard architecture, i.e. the data and program memory are separately stored and accessed. Our focus is only on the mapping and throughput of the data memory. We consider a platform with c processor cores and b independently accessible memory banks. Each memory bank takes 1 processor cycle to serve an access. Such fast memories are becoming common in cache-less many- and multi-core platforms [3, 1]. This memory model does not consider open-row effects in memories such as in SDRAM. There is no contention in the communication between the processors and the banks, i.e. there exist dedicated communication links (usually fully parallel crossbars) between the processors and the banks.

3.2 Application Model

By an application, we refer to a thread executing on a core. Each such application is characterized by a timed trace of memory accesses. We do not consider applications which are synchronized across processors.

For the analysis, we consider an *approximate stochastic* model of the application. This model characterizes each application by only two *stochastic parameters* – the access probability p_a and the sequential access probability p_{seq}. The access probability p_a represents the probability that in any given processor cycle the application will perform a memory access (assuming that perfect and contention-free memory access is guaranteed). The sequential access probability p_{seq} represents the probability that any two subsequent memory accesses will be to consecutive addresses in the local address space. Both of these mean quantities are obtained by profiling traces of the application: p_a as the ratio of cycles with memory accesses to the total number of cycles; and p_{seq}, counting the frequency of the individual address offsets between each pair of subsequent memory accesses from the trace, as the relative frequency of one-word offsets.

3.3 Problem Definition

The metric of interest is the throughput of the entire memory subsystem. This metric expresses the aggregated performance of all applications in the system. We do not regard how this performance is distributed between the individual applications; the problem of arbitration is orthogonal to the problem discussed here.

For the accurate application model, the mentioned metric can be measured using memory simulators like the gem5. For the approximate stochastic model, we aim to design analysis methods which can estimate the metric. To this end, we define two random variables: A and I, where A denotes the random number of accesses requested in any given cycle, and I represents the number of banks serving accesses in any given cycle. As an example, $\mathrm{P}(I=3)$ denotes the probability that there are exactly 3 banks serving an access in any given cycle. Then, the average throughput is given as the expectation of I, written as $\mathrm{E}[I]$. The distribution of I provides information about how much this throughput can vary and about the best and worst cases to expect.

This defines the problem definition which we will study in the next two sections: *Under the approximate stochastic model, given c, b, p_a, and p_{seq}, compute the distribution of the number I of memory banks serving accesses.*

For a small number of memory accesses (p_a close to zero), it can be expected that every access request is served immediately ($I=A$ in every cycle). With p_a growing, however, the probability increases that two processors try to access the same memory bank. Since each bank can only serve one access at a time, this may lead to $I<A$. At the same time, a bank might also serve a pending access request from a previous cycle, such that $I>A$ is possible as well. While it must hold that $I \leq b$ and $I \leq c$ and steady state considerations yield that $\mathrm{E}[I]=\mathrm{E}[A]$, the exact distribution of I, given c, b and p_a only, is far from being obvious.

These considerations are further complicated by the influence of sequentiality in memory accesses. It is clear that if all accesses are always sequential ($p_{\text{seq}}=1$) one would – after a first phase of collisions and waiting – expect all accesses to happen in lock-step with no further collisions (synchronisation effect). Yet, for smaller values of p_{seq}, it is not clear a priori if this effect still plays an important role. In particular, only the comparison with $p_{\text{seq}}=0$ can show if it is significant at all.

4. CLASSIC OCCUPANCY BASED MODEL

In this section, we will analyse memory bank access conflicts using the *classic occupancy distribution*. This necessitates the simplification from [17] that pending accesses from previous memory cycles are ignored.

We will first summarise the results from [17] (which assume $p_a = 1$) and then extend the model for different values of p_a. Finally, we will evaluate the limitations of this model.

4.1 The classic occupancy distribution

This section summarises the existing approach to the problem. If the number of actual memory accesses a in a cycle is known (the usual assumption is $a=c$), the throughput in the cycle follows the so-called *classic occupancy distribution*. This distribution is defined as follows. n balls are distributed at random into m urns. How many urns will contain at least one ball?

Setting $n=a$ and $m=b$, one can reuse the known results for this distribution [9, 11]:

$$\mathrm{P}^{occ}_{a,b}(I=i)=\left\{ {a \atop i} \right\}\cdot\frac{b^{\underline{i}}}{b^a} \quad \text{with} \quad \left\{ {a \atop i} \right\}=\frac{1}{i!}\sum_{k=0}^{i}(-1)^{i-k}\binom{i}{k}k^a, \quad (1)$$

where $b^{\underline{i}}$ is the falling factorial $b\cdot(b-1)\cdots(b-i+1)$ and $\left\{ {a \atop i} \right\}$ is the *Stirling number of the second kind*, sometimes also written as $\mathrm{S}(a,i)$. It represents the number of possibilities of partitioning a distinct elements into i non-empty sets.

The expected throughput is

$$\mathrm{E}^{occ}_{a,b}[I]=b-b\cdot\left(1-\frac{1}{b}\right)^a. \quad (2)$$

4.2 Adding access probabilities

Previously, we have seen how to predict the throughput if the total number a of simultaneous accesses is known. This, however, is not the case for $p_a<1$. To extend the model accordingly, we therefore also have to describe a.

We thus model the actual number of accesses as a random variable A, which, according to our definition follows the binomial distribution $P(A=a)=\binom{c}{a}\cdot {p_a}^a\cdot(1-p_a)^{c-a}$. Combining this with (1), we obtain

$$\mathrm{P}^{occ}_{c,b,p_a}(I=i)=\sum_{a=0}^{c}\binom{c}{a}\cdot {p_a}^a\cdot(1-p_a)^{c-a}\cdot\left\{ {a \atop i} \right\}\cdot\frac{b^{\underline{i}}}{b^a}. \quad (3)$$

The expected throughput for this distribution is

$$\mathrm{E}^{occ}_{c,b,p_a}[I]=b-b\cdot\left(1-\frac{p_a}{b}\right)^c. \quad (4)$$

This can be shown similarly to (2) in [11]. For each bank, the probability of being accessed by one given core is $p_a\cdot\frac{1}{b}$, and thus the probability *not* to be accessed by *any* core is $\left(1-\frac{p_a}{b}\right)^c$. This is also the expected number of banks *out of this one bank* which are not accessed by any core. As the expected values for the different banks can be just added up, multiplication with b and subtraction from b (to get the number of banks that are accessed) yields the result above.

Since usually $p_a \ll b$, we can introduce the following simplification. If the ratio $r=\frac{c}{b}$ of cores and banks is constant, the approximation

$$\mathrm{E}^{occ}_{c,b,p_a}[I]=b-b\cdot\left(1-\frac{p_a\cdot r}{c}\right)^c\approx b-b\cdot e^{-p_a\cdot r}, \quad (5)$$

holds with $(1-\frac{\lambda}{x})^x\approx e^{-\lambda}$ for $\lambda\ll x$, e being the Euler number.

4.3 Limitations of the model

While the model allows interesting insights, it also has some shortcomings. Firstly, since it is stateless, sequential access patterns of the applications (i.e. $p_{\mathrm{seq}} \neq 0$) cannot be taken into account. Secondly, as discussed earlier, it ignores the fact that accesses that cannot be immediately served are served in subsequent cycles, then interfering with new accesses. As long as the number of such accesses having to wait is small, the occupancy distribution can therefore be regarded as an approximation for the real system. With a higher number of conflicts, however, the numbers can be expected to deviate substantially from the real values.

5. MARKOV MODEL

In this section, we will analyse memory bank access conflicts using a Markov model. This is more calculation intensive, but also more accurate than the occupancy model.

We will again start with the known model described in [2], i.e. $p_a=1$ and $p_{\mathrm{seq}}=0$. We will then extend the model to include sequential accesses and finally, we will introduce the memory access probability again.

5.1 Known model

In the following, it will be explained how Markov model steady states are used in general to calculate certain distributions and how this is done in [2] for modelling interleaved memory throughput.

In general, a Markov model consists of a set of k states $S=\{\boldsymbol{s}^1,...,\boldsymbol{s}^k\}$ and a transition probability function $f:S\times S\to[0,1]$. $f(\boldsymbol{s},\boldsymbol{t})=\mathrm{P}(\boldsymbol{s}\to\boldsymbol{t})$ is the probability of a transition from state $\boldsymbol{s}$ to $\boldsymbol{t}$.

One can now define a *transition matrix* $T\in\mathbb{R}^{k\times k}$ with $T_{i,j}=P(\boldsymbol{s}^j\to\boldsymbol{s}^i)$. If a vector $v\in\mathbb{R}^k$ contains the probabilities v_j for the system to be in a state $\boldsymbol{s}^j$ at a certain point, $T\cdot v$ will contain the state probabilities after one state transition. After a a large number of rounds, the system has reached the *steady state* described by the probability vector $\boldsymbol{\sigma}\in\mathbb{R}^k$. It holds that

$$T\cdot\boldsymbol{\sigma}=\boldsymbol{\sigma}. \quad (6)$$

If the transition probabilities are known, one can thus calculate the steady state probabilities by solving the mentioned eigenvector problem.

Since one is usually not interested in the probabilities of certain states but rather in the probabilities of certain quantities associated with the states, one will define a quantity function $q:S\to\mathbb{R}$ mapping the states to the quantities. Then one can extract the probability of a certain value q^* as the probability sum of all the concerned states

$$\mathrm{P}(Q=q^*)=\sum_{j\in J}\sigma_j \text{ with } J=\left\{j \;\middle|\; q(\boldsymbol{s}^j)=q^*\right\}. \quad (7)$$

To apply the tool of the Markov steady state to the interleaved memory throughput problem in order to calculate $\mathrm{P}^{mkv}_{c,b,p_a=1,p_{\mathrm{seq}}=0}(I=i)$ and $\mathrm{E}^{mkv}_{c,b,p_a=1,p_{\mathrm{seq}}=0}[I]$, one needs to define the set of states, derive the transition probabilities and provide a mapping from the states to the throughput. In the following, it will be shown how this is done in [2]. Accordingly, it is assumed that $a=c$ and that $p_{\mathrm{seq}}=0$.

State set. To provide better extensibility for later problems, we encode the states differently here than it is done in [2]. The states themselves, however, are still the same. The state vectors we use have the form

$$\boldsymbol{s}=(s_1,...,s_c),$$

where s_n is the number of banks having exactly n accesses in their queue. Summing up the queue lengths of all banks, one can see that the state set is

$$S=\left\{\boldsymbol{s}\in\mathbb{N}^c \;\middle|\; \sum_{j=0}^{c}j\cdot s_j=a\right\}. \quad (8)$$

We define $s_0=b-\sum_{j=1}^{c}s_j$, the number of idle memory banks for a state $\boldsymbol{s}$.

The number of states is equal to the number of partitions of a: As an example, for $a=16$, there are $P(a)=273$ different states.

Transition probabilities. The probability of attaining a state $\boldsymbol{s}$ out of the initial state $\boldsymbol{s}^0$ (with all banks idle) can be calculated

directly as [12]

$$\mathrm{P}(\boldsymbol{s}^0 \to \boldsymbol{s}) = \frac{a! \cdot b!}{b^a \cdot \prod_{j=0}^{c} [(j!)^{s_j} \cdot s_j!]}. \quad (9)$$

However, for attaining a target state $\boldsymbol{t}$ from an arbitrary state $\boldsymbol{s}$, there are multiple different access redistribution possibilities, which makes it hard to come up with a closed formula for calculating the associated probability. (With access redistributions, we model the fact that as soon as an access is served, the concerned processor will make a new access request to a new bank.) The problem can be solved by first determining the "stripped" state

$$\boldsymbol{s}' = (s_2, s_3, ..., s_c, 0), \quad (10)$$

in which one access request has been removed from each bank's queue, and by then enumerating the possibilities for distributing new access requests such that $\boldsymbol{t}$ is attained. One algorithm for this calculation is described in [5, 4].

State throughput mapping. For a state $\boldsymbol{s}$, the associated throughput is $i(\boldsymbol{s}) = b - s_0$.

5.2 Adding sequential access patterns

In order to analyse the access synchronisation effect for SIMs mentioned in the introduction, we now extend the model by a *sequential access probability* p_{seq}. We assume a (circular) order of the memory banks and that upon each access, with a probability of p_{seq} the memory bank assigned to the accessed address will not be random but instead the bank next to the previously accessed one.

Clearly, an exact solution to the problem would need distinction of the individual banks, since their relative position to each other has now become relevant. Such a distinction would, however, blow up the number of states to $\binom{a+b-1}{b-1}$, e.g. $1.5 \cdot 10^{12}$ states for the configuration $a=16$, $b=32$ (P2012), which is computationally difficult. We thus stick to the states introduced in the last section and simplifyingly assume that all combinations of relative bank positionings are equiprobable.

The redistribution of the accesses in this model can be regarded as consisting two distinct steps: First a sequential distribution (access requests to the "next" banks) of n_{seq} balls and then a random distribution of n_{rnd} accesses, with $n_{\mathrm{seq}} + n_{\mathrm{rnd}} = n_{\mathrm{redist}}$ the total number of accesses redistributed. (6) can then be rewritten as $T_{\mathrm{rnd}} \cdot (T_{\mathrm{seq}} \cdot \boldsymbol{\sigma}) = T_{\mathrm{rnd}} \cdot T_{\mathrm{seq}} \cdot \boldsymbol{\sigma} = \boldsymbol{\sigma}$, defining a new transition matrix $T = T_{\mathrm{rnd}} \cdot T_{\mathrm{seq}}$.

Note that there has to be a higher number of "intermediary" states between applying T_{seq} and T_{rnd}, since a subset of the accesses has not yet been redistributed and thus (8) must be weakened to

$$\sum_{j=0}^{c} j \cdot s_j \leq a, \quad (11)$$

increasing the number of intermediary states to $\sum_{j=0}^{a} P(a)$, e.g. 915 states for $a=16$. In this case, it holds that $T_{\mathrm{rnd}} \in \mathbb{R}^{273 \times 915}$ and $T_{\mathrm{seq}} \in \mathbb{R}^{915 \times 273}$.

The algorithm from [4] mentioned in the previous section can also be used to calculate T_{rnd}. Note that T_{rnd} does not depend on p_{seq}, it can thus be reused when testing different values of p_{seq}.

T_{seq} can be calculated as follows. For a transition from a state $\boldsymbol{s}$ to an intermediary target state $\boldsymbol{t}$, one calculates the "stripped state" $\boldsymbol{s}'$ according to (10) and then the *upgrade vector* $\boldsymbol{u} \in \mathbb{N}^{0 \dots c}$ as

$$u_j = \sum_{\nu=0}^{j} u_\nu^* \quad \text{with} \quad \boldsymbol{u}^* = \boldsymbol{s}' - \boldsymbol{t},\ u_0^* = s_0' - t_0. \quad (12)$$

The idea is that if in $\boldsymbol{s}'$, s_0' banks are idle and in $\boldsymbol{t}$, only t_0 banks should be idle, $u_0 = s_0' - t_0$ banks have to be *upgraded* by assigning one access to them. Now there are $s_1' + u_0$ banks with a queue length of exactly one; if only t_1 of these are required, $t_1 - (s_1' + u_0) = u_1$ of them have to be upgraded again etc. If any element of $\boldsymbol{u}$ is negative, a transition is not possible, i.e. the transition probability is zero. Also, the transition is only possible if $\forall j,\ s_j' \geq u_j$, since no bank can receive more than one access through sequential redistribution.

For each upgrade class j, there are now u_j out of s_j' banks that must be upgraded; in total, there are $n_{\mathrm{seq}} = \sum_{j=1}^{c} (t_j - s_j')$ accesses to be sequentially distributed and thus n_{seq} out of b banks to be upgraded. Together with the (binomial) probability that indeed n_{seq} accesses are redistributed sequentially, the joint probability for the sequential transition from $\boldsymbol{s}$ to $\boldsymbol{t}$ is equal to

$$\mathrm{P}(\boldsymbol{s} \xrightarrow{\mathrm{seq}} \boldsymbol{t}) = \binom{n_{\mathrm{redist}}}{n_{\mathrm{seq}}} \cdot {p_{\mathrm{seq}}}^{n_{\mathrm{seq}}} \cdot (1 - p_{\mathrm{seq}})^{n_{\mathrm{rnd}}} \cdot \frac{\prod_{j=0}^{c} \binom{s_j'}{u_j}}{\binom{b}{n_{\mathrm{seq}}}}. \quad (13)$$

5.3 Adding access probabilities

In this section, we will remove the previous assumption $a=c$ and introduce the memory access probability p_a for each core.

The solution to this problem is a combination of different approaches already shown in this paper. It is clear that dropping the assumption $a=c$ increases the number of possible states; in fact, since each state $\boldsymbol{s}$ has to fulfil the requirement

$$\sum_{i=0}^{c} i \cdot s_i \leq c, \quad (14)$$

which is similar to (11), the new states are identical to the intermediary states from the last section.

Like in Section 4.2, the different possible values of a and thus of n_{redist} for the target state have to be considered separately:

$$\left(\sum_{a=0}^{c} {T_{\mathrm{rnd}}}^{A=a} \cdot {T_{\mathrm{seq}}}^{A=a} \right) \cdot \boldsymbol{\sigma} = \boldsymbol{\sigma},$$

For each possible transition, the probability that $A=a$ must be included as well. This can be done by multiplying it into ${T_{\mathrm{seq}}}^{A=a}$, i.e.

$${T_{\mathrm{seq}}}_{i,j}^{A=a} = \binom{n_{\mathrm{avail}}}{n_{\mathrm{idle}}} \cdot {p_a}^{(n_{\mathrm{avail}} - n_{\mathrm{idle}})} \cdot (1 - p_a)^{n_{\mathrm{idle}}} \cdot \mathrm{P}(\boldsymbol{s}^j \xrightarrow{\mathrm{seq}} \boldsymbol{s}^i),$$

with $n_{\mathrm{avail}} = b - s_0$ the number of cores available for requesting new accesses and $n_{\mathrm{idle}} = c - a$ the number of cores that are not going to request an access.

For calculating ${T_{\mathrm{rnd}}}^{A=a}$, the algorithm from [4] mentioned in the previous sections can be used again.

6. EXPERIMENTAL EVALUATION

In this section we compare the estimates based on the analysis of the approximate model with the simulated results of different SIM configurations with real benchmark applications. First we describe the setup and then the obtained results. We also draw conclusions from the models and give hints for system designers.

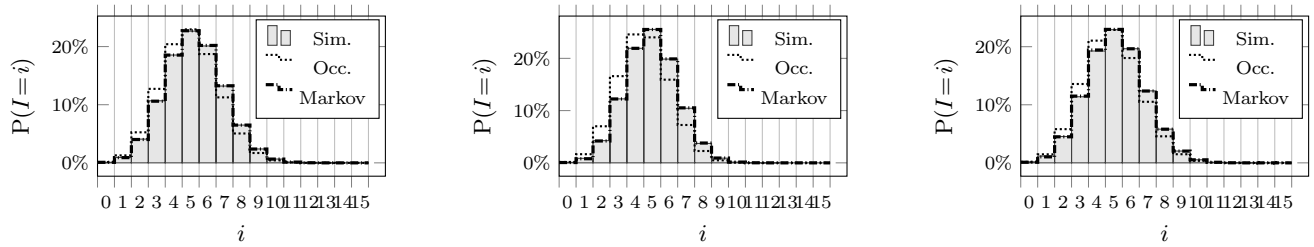
6.1 Experimental setup

All benchmarks were compiled for and run on the gem5 [6] ARM simulator. For every core (we simulated $c=16$), a separate memory access trace was created, recording the accessed addresses as well as the pauses between the accesses. Wherever possible, program parameters or inputs were varied in order to create different access patterns.

A bank access and collision simulation was then carried out on a custom simulator, replaying each of the traces with short, but random initial delays. The memory was configured either with $b=16$ or $b=32$ banks (the latter is the configuration of P2012) with round robin arbitration.

A selection of benchmarks from the MiBench embedded benchmark suite [10] was used for the experiments. The GSM, FFT, blowfish, string search and JPEG examples were chosen to obtain a high diversity in behaviour.

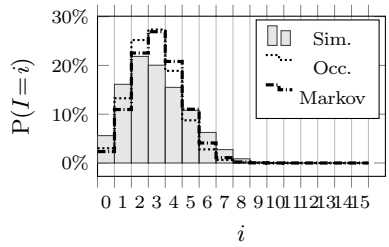
From the traces, p_a and p_{seq} were extracted and used as parameters for the occupancy and for the Markov model.



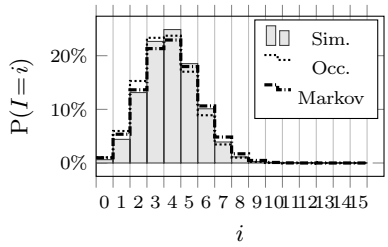
(a) Blowfish. $b=32$, $p_a=0.34$, $p_{\text{seq}}=0.27$. $\overline{I}^{sim}=5.23$, $\text{E}^{mkv}[I]=5.24$, $\text{E}^{occ}[I]=4.98$.

(b) GSM. $b=16$, $p_a=0.33$, $p_{\text{seq}}=0.07$. $\overline{I}^{sim}=4.93$, $\text{E}^{mkv}[I]=4.94$, $\text{E}^{occ}[I]=4.53$.

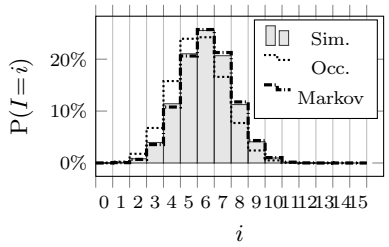
(c) GSM. $b=32$, $p_a=0.33$, $p_{\text{seq}}=0.07$. $\overline{I}^{sim}=5.12$, $\text{E}^{mkv}[I]=5.13$, $\text{E}^{occ}[I]=4.89$.



(d) FFT. $b=16$, $p_a=0.20$, $p_{\text{seq}}=0.49$. $\overline{I}^{sim}=3.02$, $\text{E}^{mkv}[I]=3.09$, $\text{E}^{occ}[I]=2.87$.



(e) String search. $b=32$, $p_a=0.25$, $p_{\text{seq}}=0.86$. $\overline{I}^{sim}=3.94$, $\text{E}^{mkv}[I]=3.95$, $\text{E}^{occ}[I]=3.75$.



(f) JPEG. $b=16$, $p_a=0.42$, $p_{\text{seq}}=0.14$. $\overline{I}^{sim}=6.01$, $\text{E}^{mkv}[I]=6.07$, $\text{E}^{occ}[I]=5.57$.

Figure 1: Simulation results for different benchmarks. The simulated systems have $c=16$ cores and either $b=16$ or $b=32$ memory banks. For each benchmark, the simulated throughput frequencies are plotted together with the predictions from the Markov and the occupancy model. Model parameters like b, p_a and p_{seq} are given as well as the simulated average throughput and $\overline{I}^{sim}$ and the expected throughput according to the Markov and the occupancy models.

6.2 Accuracy of the occupancy model

The occupancy model makes drastic simplifications, especially when a high number of waiting accesses is expected in a system. Fig. 2 compares the simulated throughput of the Blowfish benchmark for $c=16$ and $1\leq b\leq 64$ with the predicted value from the occupancy model. For a small number of banks, the throughput is likely to be close to that number and thus accurately computed. For a sufficiently large number of banks, the number of waiting accesses is small and thus the accuracy of the model is good as well. As expected, the deviation from the simulation is highest in between, i.e. around the value $b=p_a\cdot c$. The maximum error observed is about 12%. However, the trend of the throughput is still captured by the occupancy model.

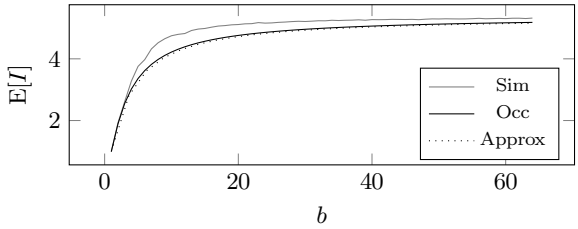


Figure 2: Accuracy of the occupancy model. The simulated average throughput for the Blowfish benchmark ($p_a=0.34$) and the expected throughput according to the occupancy model are plotted against the number of memory banks for $c=16$. The approximation (5) is plotted as well. The maximum relative error is of 12.0% for $b=8$.

6.3 Benchmark Results

Fig. 1 shows the results of the simulations for all the benchmark applications. The grey pillars show simulated throughput frequencies over all the cycles, the dotted and dash-dotted lines give the probabilities calculated using the occupancy and the Markov model. Next to each plot, the extracted memory access and sequential access probabilities are given.

In general, both the occupancy and the Markov models reflect well the trends that emerge in the simulation. As expected, the Markov model fits particularly well for small values of p_{seq} and for $p_{\text{seq}}\to 1$. In these settings, the Markov results are nearly congruent with the simulation results. For the string search example, the deviations can be explained with the inexactness resulting from the simplified sequentiality model as described in Section 5.2.

Only in the FFT example, the simulation shows a clearly large variance compared to the Markov model. We believe this is because of the specific access pattern of the FFT algorithm. The algorithm recursively splits arrays with a dimension of a power of two into two sub-arrays. It switches accesses between these arrays and thus produces multiple accesses in a row on the same memory bank. This is a well-documented effect (cf., e.g., [15]) in which multiple processors with similar memory access patterns frequently access the same banks, thus repeatedly blocking each other. A similar effect is observed when multiplying matrices with powers of two as their dimensions. Customized programming is required to avoid such instances on SIM memories. The FFT benchmark was the worst fit from all the benchmarks we tested.

In summary, the simulations show that for several benchmark applications the simple probabilistic model of an access and a sequential access probability accurately estimates the throughput of an interleaved memory system.

6.4 Conclusions from the occupancy model

The advantage of the occupancy model is the analytical form of its result. We now draw some conclusions from these results, which hold only under the conditions as described in Section 6.2. In (5), the expected throughput of a system was approximated as $\text{E}[I]\approx b-b\cdot e^{-p_a\cdot r}$ with $r=\frac{c}{b}$. This simplification yields the following conclusions.

- As long as the ratio of banks and cores is constant, a system can be arbitrarily scaled without changing the throughput expectation per bank or per core.
- The throughput converges exponentially with the product $p_a\cdot r$ to the maximum value b. Note that $p_a\cdot r=\frac{1}{b}\text{E}[A]$ is the expected number of accesses per memory bank.
- For $p_a\cdot r<0.3$, the throughput can be regarded as growing approximately linearly with p_a.

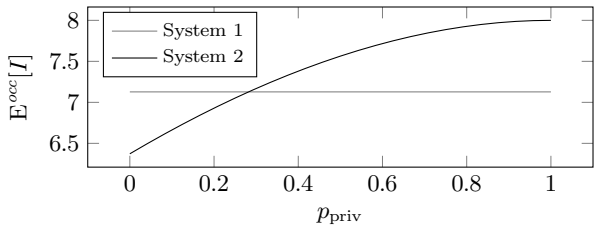


Figure 3: Comparison of two systems: System 1 with fully interleaved memory and System 2 with partially interleaved, partially private memory.

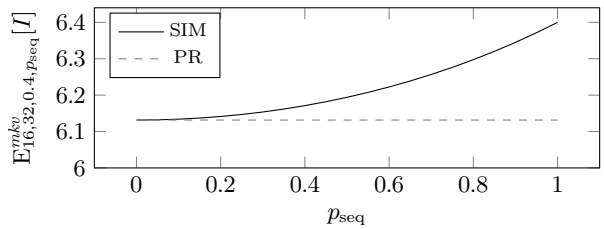


Figure 4: Evaluation of the synchronisation effect in SIM systems and systems with pseudo-random mapping (PR). The throughput according to the Markov model is plotted as a function of p_{seq} for $c{=}16$, $b{=}32$, $p_a{=}0.4$.

6.5 Application example: System design

The occupancy model can also be used to answer simple design questions. Consider the comparison between two different systems, each with $c{=}16$ cores and $b{=}32$ banks. System 1 uses complete interleaving over all 32 banks. System 2 uses interleaving for 16 banks and provides each core with one "private" memory bank. Let now p_{priv} be the probability that an access goes to the private memory bank on System 2. The question is which setting is better, and for which values of p_{priv}.

Since every access to the private memory is immediately served, the expected throughput for System 2 is

$$\mathrm{E}^{occ}_{\mathrm{System2}}[I]{=}c{\cdot}p_a{\cdot}p_{\mathrm{priv}}{+}\mathrm{E}^{occ}_{c,\frac{b}{2},p_a{\cdot}(1-p_{\mathrm{priv}})}[I],$$

whereas (2) can be used for System 1. Both expected throughputs are shown for $p_a = 0.5$ as a function of p_{priv} in Fig. 3. For System 1, the throughput is constant as p_{priv} does not play a role there. For System 2, the worst case is $p_{\mathrm{priv}}{=}0$ with effectively only 16 banks, the private banks not being used. The throughput increases with p_{priv} increasing, the maximum value being attained for $p_{\mathrm{priv}}{=}1$, in which case no conflicts occur as each core only uses its private memory. Comparing both systems, System 2 performs better for $p_{\mathrm{priv}}{>}0.2875$. Many other such comparisons can be made with the analytical results from the occupancy model.

6.6 Discussion of the synchronisation effect

The support for sequential access patterns in the Markov model allows to analyse the access synchronisation effect described earlier. For this, Fig. 4 shows a plot of the throughput of a SIM system with $c{=}16$ and $b{=}32$ as a function of p_{seq}. The memory access probability has been chosen rather high with $p_a{=}0.4$. With a pseudo-random address mapping, the throughput is equal to the case when $p_{\mathrm{seq}}{=}0$. Two points can be seen from the plot: (i) For $p_{\mathrm{seq}}{<}0.4$, the synchronisation effect is insignificant. (ii) Even the speedup from $p_{\mathrm{seq}}{=}0$ to $p_{\mathrm{seq}}{=}1$ is less than 5% in this system.

This small benefit is opposed by the danger of large performance losses due to the mutual blocking effect described earlier for the FFT benchmark. Software designers who appreciate code compatibility or who do not know their target platform in detail, may also on SIM systems want to randomise their applications' memory accesses. This could for example be achieved by running multiple different types of tasks on a cluster instead of having a homogeneous set of threads. We claim that there are only few cases in which performance is likely to be a decisive factor for opting for a SIM system rather than for pseudo-random mapping.

7. CONCLUDING REMARKS

In this work, interferences in sequentially interleaved memory systems have been theoretically analysed using a simple probabilistic abstraction. Two models – a fast occupancy distribution based model and a more accurate Markov model – have been presented and discussed. General principles and character traits of interleaved memory systems have been deduced theoretically from the models. Finally, the models have been validated in an extensive set of simulations.

Apart from many concrete results such as the small impact of the synchronisation effect, the most important result of this paper is that these systems can be adequately described by a simple probabilistic abstraction which only knows the memory access probability of the processes and the probability of accessing sequential memory banks. This gives room for extensive abstract analyses and, hopefully, further interesting insights.

8. ACKNOWLEDGEMENT

This work was supported in part by the UltrasoundToGo RTD project (no. 20NA21 145911), evaluated by the Swiss NSF and funded by Nano-Tera.ch with Swiss Confederation financing.

9. REFERENCES

[1] Adapteva. Epiphany Architecture Reference Manual. `http://adapteva.com/docs/epiphany_arch_ref.pdf`, 2008-2013.

[2] F. Baskett and A. J. Smith. Interference in Multiprocessor Computer Systems with Interleaved Memory. *Comm. ACM*, 19(6):327–334, June 1976.

[3] L. Benini et al. P2012: Building an Ecosystem for a Scalable, Modular and High-Efficiency Embedded Computing Accelerator. In *Proc. DATE*, pages 983–987, 2012.

[4] D. P. Bhandarkar. Analysis of memory interference in multiprocessors. *Computers, IEEE Transactions on*, 100(9):897–908, 1975.

[5] D. P. Bhandarkar and S. H. Fuller. Markov Chain Models for Analyzing Memory Interference in Multiprocessor Computer Systems. In *Proc. Symposium on Computer Architecture*, ISCA '73, pages 1–6. ACM, 1973.

[6] N. Binkert et al. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1–7, 2011.

[7] G. J. Burnett and E. G. Coffman, Jr. A Study of Interleaved Memory Systems. In *Proc. Spring Joint Computer Conference*, AFIPS '70 (Spring), pages 467–474. ACM, May 1970.

[8] G. J. Burnett and E. G. Coffman, Jr. A Combinatorial Problem Related to Interleaved Memory Systems. *J. ACM*, 20:39–45, Jan. 1973.

[9] W. Feller. *An introduction to probability theory and its applications*, volume 1. Wiley, 1968.

[10] M. R. Guthaus et al. MiBench: A free, commercially representative embedded benchmark suite. In *Workload Characterization, 2001. WWC-4. 2001 IEEE Int'l Workshop on*, pages 3–14. IEEE, 2001.

[11] N. Johnson and S. Kotz. *Urn models and their application: an approach to modern discrete probability theory*. Wiley, 1977.

[12] R. v. Mises. Über Aufteilungs- und Besetzungswahrscheinlichkeiten. *Revue de la Faculté de Sciences de l'Université d'Istanbul*, pages 145–163, 1938-39.

[13] S. Rabinowitz. A model for interference in multiprocessors. *Missouri Journal of Mathematical Sciences*, 3:12–19, 1991.

[14] B. R. Rau. Interleaved Memory Bandwidth in a Model of a Multiprocessor Computer System. *IEEE Trans. Computers*, 28(9):678–681, 1979.

[15] B. R. Rau. Pseudo-randomly Interleaved Memory. In *Proceedings of the 18th Annual International Symposium on Computer Architecture*, ISCA '91, pages 74–83. ACM, 1991.

[16] C. E. Skinner and J. R. Asher. Effects of storage contention on system performance. *IBM Systems Journal*, 8(4):319–333, 1969.

[17] W. D. Strecker. *An analysis of the instruction execution rate in certain computer structures*. PhD thesis, 1970.