Hardware-Based Fast Exploration of Cache Hierarchies in Application Specific MPSoCs

Isuru Nawinne  Josef Schneider  Haris Javaid  Sri Parameswaran
School of Computer Science and Engineering, The University of New South Wales, Sydney, Australia
Email: isurun,jschneider,harisj,sridevan@cse.unsw.edu.au

Abstract—Multi-level caches are widely used to improve the memory access speed of multiprocessor systems. Deciding on a suitable set of cache memories for an application specific embedded system’s memory hierarchy is a tedious problem, particularly in the case of MPSoCs. To accurately determine the number of hits and misses for all the configurations in the design space of an MPSoC, researchers extract the trace first using Instruction set simulators and then simulate using a software simulator. Such simulations take several hours to months. We propose a novel method based on specialized hardware which can quickly simulate the design space of cache configurations for a shared memory multiprocessor system on an FPGA, by analyzing the memory traces and calculating the cache hits and misses simultaneously. We demonstrate that our simulator can explore the cache design space of a quad-core system with private $L_1$ caches and a shared $L_2$ cache, over a range of standard benchmarks, taking as less as $0.106$ seconds per million memory accesses, which is up to $456$ times faster than the fastest known software based simulator. Since we emulate the program and analyze memory traces simultaneously, we eliminate the need to extract multiple memory access traces prior to simulation, which saves a significant amount of time during the design stage.

1. INTRODUCTION

Memory subsystem is a major deciding factor when considering the performance of processor-based systems. Recent advancements in processor architectures and manufacturing technologies have enabled processors to operate at increasingly high frequencies. Unfortunately the same cannot be said for memory systems. Accessing the main memory consumes a large number of processor clock cycles, making it a performance bottleneck. The most widely used solution is employing faster but smaller cache memories to hold the most recently used data close to the processor for efficient access. Caches improve performance based on two attributes of application programs: a memory block is likely to be repeatedly accessed (i.e. temporal locality); and adjacent memory blocks are likely to be accessed in sequence (i.e. spatial locality).

In application specific embedded Multi-Processor System on Chips (MPSoC), where processors are optimized for applications, the caches in the memory hierarchy also need to be optimized. Such optimizations have been enabled by customizable processors such as ARM Cortex and Tensilica Xtensa. A designer of such systems needs to identify suitable configurations for different caches in the hierarchy, with respect to Block Size, Set Size and Associativity. This requires the exploration of a large design space, resulting from private and shared caches arranged in a multi-level hierarchy.

Simply selecting the largest available cache configuration to achieve a high hit rate does not necessarily provide better performance, contrary to what intuition suggests, as demonstrated by Shwe et al. in [1] and Janapsaty et al. in [2]. Figure 1 presents the performance and energy consumption variations when using different cache configurations, and shows that the largest configuration (with maximum hits) provides neither best performance nor least energy. Infact, large caches can often result in slower memory accesses in addition to power and chip die overheads. Therefore, it’s essential to explore the design space of the cache hierarchy, accurately find the hit rates of different configurations and use the results to obtain estimates for performance and energy consumption.

Individually testing the behaviour of different cache configurations for the system under design and counting their cache hits is not a feasible solution. Instead, the literature contains a large body of research work towards software simulation methods of multiple cache configurations, mostly for a single cache in a uniprocessor system. Some methods [3, 4, 5] rely on mathematical analyses to make guesses at cache hit rates; while a majority [6, 7, 2, 8, 9, 10, 11] aims at exact simulation of a set of cache configurations, where precise hit and miss rates are calculated for a given application from its memory access trace. Even though precise simulations provide accurate hit rates, the simulation time taken in software is significantly high. For the memory access trace detailed in Fig. 2, the fastest available software simulator by Sugumar et al. [6] takes 90 minutes to cover the design space of a single $L_1$ cache (consisting of 44 configurations). A more acute problem in software simulation is the extraction of memory access trace, which is a painstakingly slow process. For example, Fig. 2 reports that 72 hours were spent in trace extraction of an MPEG2 encoder executing on a single core for an input of 24 low resolution video frames.

The aforementioned precise software simulation methods are targeted at uniprocessor systems. The considerable time consumption of these methods multiplies when the design problem is extended to multiprocessor systems with complex cache hierarchies (see Fig. 3). Multiprocessor cache hierarchies bring additional concerns into the simulation problem. For example, most last level caches are shared among processors, and their dimensioning must consider interleaving of memory accesses from all processors. If the target system executes communicating applications and includes a coherency controller for the private caches, then the effects of cache block invalidations due to maintaining coherency must also be taken into account for precise hit rate calculations. Simulation methods such as [12] which combine these factors tend to be several times slower than uniprocessor cache design space exploration methods. The extraction of memory trace in multi-level multiprocessor cache hierarchy becomes more challenging as multiple memory access traces from various points in the MPSoC memory subsystem need to extracted to accurately capture the memory access behavior. This process become very slow and practically infeasible when traces have to be extracted repeatedly to perform consecutive simulations of large cache hierarchies. The authors of [13] recently presented a hardware based cache simulation core to mitigate the problems of software simulation through the use parallel processing offered by FPGA logic. However, [13] explored the design space of cache in a uniprocessor only.

In this paper, we present the first ever hardware based methodology to rapidly perform exploration of the cache design space for a multi-
level multiprocessor cache hierarchy containing private and shared caches. We eliminate the need for repeated extraction of memory access traces by obtaining the memory access traces in real-time from different points of MPSoC memory subsystem. Those memory access traces are processed in hardware cache simulator core to precisely calculate the cache hits and miss rates. A demonstration is presented targeting a two-level cache hierarchy with private level 1 data caches and a shared level 2 data cache for a quad-core system executing non-communicating applications on an FPGA. Our novel contributions are:

- We present the first ever hardware based rapid design space exploration of multiprocessor cache hierarchies containing private and shared caches, to select suitable cache configurations.

- We propose a method to flexibly connect many instances of a hardware simulator core to different points in the memory hierarchy of an MPSoC in an FPGA, thereby enabling extraction of memory access traces in real-time as experienced by individual caches. The memory accesses are processed in parallel to the MPSoC execution, to calculate precise cache hit and miss rates for different cache configurations. Finally, a suitable cache hierarchy is selected based upon the hit and miss rates.

The rest of this paper is organized as follows: Section II presents a concise analysis of the literature; Section III identifies the target MPSoC architectures of the proposed method; the methodology is detailed in Section IV; hardware implementation details are presented in Section V; Sections VI and VII present a demonstration of the proposed hardware based method.

### II. RELATED WORK

Exact simulation methods to explore the cache configuration design space have evolved with many advancements over the years. They are characterized with providing the precise hit rate for many cache configurations at once, given a memory access trace. Several different approaches have been proposed to accelerate the software based simulation methods focusing on \( L_1 \) caches for uniprocessor systems. *Dinero IV* by Hill [14] is one of the most widely used simulators which analyses the hit rate for a single cache configuration at a time by using a memory access trace. Based on the *Forest Simulation* technique introduced by Hill et al. in [7], Janapasuya et al. proposed a method to simultaneously assess hits and misses for a large group of cache configurations [2]. There, collections of binomial tree structures representing different cache configurations are traversed top down for each memory access in the trace, and correlation properties between cache configurations are exploited to improve the simulation speed. Subsequently, Tojo et al. proposed enhancements [15] to Janapasuya’s algorithm by introducing more correlation properties between cache configurations. Out of many methods, *Cheetah* simulator [6] by Sugumar et al. and *SuSeSim* [9] by Haque et al. are two of the fastest implementations to date which are based on the forest of binomial tree structures.

Viana et al. tackled the problem using a different approach in their work [16]. They determined whether a memory access is a hit or a miss by keeping track of how many unique addresses were accessed using stack and table data structures. In [10], Zang et al. extended the same concept to exclusive two level caches, where the content of the two caches are disjoint sets. The assumption of exclusivity allowed Zang et al. to view the two cache levels as one single cache and use a single memory access trace rather than extracting different access traces.

By incorporating an FPGA device in the simulation process, the authors of [13] designed a hardware simulator core to accelerate exploring of cache configuration design space for a single cache in a uniprocessor system. It uses LRU (least recently used) replacement policy for set-associative cache configurations. The configurable logic allows several cache configurations to be analysed in parallel for each memory access, significantly reducing the simulation time (up to 53 times faster than *Cheetah* simulator [6]). It is the fastest design tool presented yet to explore the cache configuration design space for a uniprocessor cache. The simulator core itself operating in hardware enables the possibility of real-time extraction of memory access information from a processor working in the same FPGA, eliminating the tedious process of extracting the memory access trace beforehand. The work in [13] is targeted at reducing the logic footprint of the hardware simulator core, to encompass more cache configurations into the design space.

With many computing systems adopting multiprocessors, recent literature looks at methods to explore the design space of cache hierarchies for such systems. *DIMSim* [12] by Haque et al. presents a two stage methodology to find the suitable cache configurations for a system as in Fig. 3, containing private \( L_1 \) caches and a shared \( L_2 \) cache. They extract the combined memory access trace as observed by the main memory and use that to derive separate memory accesses contributed by different processors. The first stage of the simulation explores the design space for the shared \( L_2 \) cache using the combined trace, and the second stage simulates the configurations for each \( L_1 \) cache. Assuming that the two cache levels are inclusive, the misses in the selected \( L_2 \) configuration are considered to be misses in all \( L_1 \) configurations. However, once the selected \( L_1 \) caches are in the system, the accesses seen by the \( L_2 \) cache change. Therefore, the method does not guarantee the optimal configuration. In [17] Haque et al. perform a simulation starting from \( L_1 \) caches, and then combining the traces to simulate configurations for a shared \( L_2 \) cache. Rawlins et al. present a dynamic reconfiguration of \( L_1 \) caches based on a control system for a dual-core processor in their work [18]. It is a run-time approach and doesn’t provide the designer with information on all concerned cache configurations.

To the best of our knowledge, no work has been presented yet to explore the design space of multi-level cache hierarchies for multiprocessor systems using specialised hardware to accelerate the exploration process.

### III. TARGET MULTIPROCESSOR SYSTEM ARCHITECTURE

![Fig. 3: Multiprocessor cache hierarchy with private \( L_1 \) caches and a shared \( L_2 \) cache.](image)

This work focuses on shared memory multiprocessor systems with multi-level cache hierarchies containing private and shared caches, as depicted in Fig. 3. A hierarchy can contain \( n \) levels of caches and each level \( L_i \) can contain \( m_i \) caches. The system used for demonstrations contains four processor cores with four private \( L_1 \) caches and one shared \( L_2 \) cache. We assume that the memory accesses produced by the processors are blocking, and that the caches used in the system do not implement advanced techniques such as block pre-fetching, similar to the prior works [6, 2, 10, 12]. These assumptions allow deterministic simulation of cache hits and misses. It should be noted that we consider a cache hierarchy where no coherency controlling is performed (and it is out future work). Unlike the works in [10] and [12], we do not make assumptions as to inclusiveness or exclusiveness between cache levels which makes our method applicable in a broader range of problem instances.

### IV. METHODOLOGY

In this section we present the proposed methodology to explore the design space of a multiprocessor cache hierarchy, highlighting the use of hardware modules to accelerate the simulation.
A. Hybrid Simulation Platform

Many state-of-the-art design time methods tend to use Hybrid Simulation where the repetitive and time consuming portions in the design process are accelerated using hardware components [19]. The hybrid simulation methodology presented in this paper utilizes an FPGA device connected to a host PC, as illustrated in Fig. 4. Target MPSoC exists in the FPGA, with hardware cache simulation modules (hSim). Input data for the applications running on the MPSoC are provided by the Host PC. The hSim modules extract the memory accesses generated by the MPSoC in real-time, calculate the hit rates of different configurations for each cache, in parallel to the execution of the applications. The resulting hit rates are sent to the Host PC where analytical models are used to estimate the timing and energy measures. Details about the hSim modules and how the memory access extraction is done in real-time are presented in Section V.

Fig. 4: Hybrid simulation platform where cache hit rates are calculated on FPGA.

Initially, the MPSoC doesn’t contain any cache. The cache hierarchy is explored in stages, starting from the L1 caches and moving down the hierarchy until the last cache level Ln. In the ith stage, configurations for all m_i caches in level Li of the hierarchy are explored in parallel to calculate the hit rates. After the results (the hit rates for all the configurations in the design space, for each cache in level i) are sent to the Host PC, timing and energy values are estimated for all configurations and a selection is made based on minimum energy or maximum performance. Afterwards, caches with selected configurations are put into the ith level in the cache hierarchy and the system is re-synthesized before simulation moves on to level i + 1.

B. Selection of Cache Configurations

The calculated hit rates for different cache configurations that are provided by the hSim modules are used for analysis of Average Cache Access Time (T_cache) and Average Access Energy Consumption (E_cache). T_cache and E_cache are normalized values per single access to the cache. These two measures are used to select a suitable cache configuration depending on the requirement. The model for T_cache is described in (1) and provides a time estimate for accessing a cache with a given configuration.

\[ T_{\text{cache}} = \alpha_{\text{access}} + (1 - h_c) \times t_{\text{miss}} \]  

(1)

\[ E_{\text{cache}} = \alpha_{\text{access}} + (1 - h_c) \times e_{\text{miss}} \]  

(2)

The term \( t_{\text{access}} \) represents the time required to make a single access to the cache, which encompasses the parameters of the cache configuration such as associativity and set size. Hit rate for the configuration is given by \( h_c \) and the time penalty for a cache miss is given by \( t_{\text{miss}} \). Similarly, (2) provides an average energy measure for accessing a cache with a given configuration. Energy required to make a single cache access is given by \( E_{\text{access}} \), representing the effects of the configuration, and the energy penalty for a cache miss is given by \( e_{\text{miss}} \).

For all the configurations in the design space, the values of \( t_{\text{access}} \) and \( E_{\text{access}} \) are obtained by using the detailed cache analysis tool CACTI 6.5 [20] by Muralimanohar et al. It should be noted that any analytical model can be used for this purpose depending on the designer’s requirement, and the performance and accuracy of the model used is beyond the scope of this work. Since \( h_c \) is provided by the simulation done in hardware, the only unknown terms are \( e_{\text{miss}} \) and \( e_{\text{miss}} \).

\[ t_{\text{miss}} = t_{\text{fetch}} + t_{\text{write}} \]  

(3)

\[ e_{\text{miss}} = e_{\text{fetch}} + e_{\text{write}} \]  

(4)

Equations (3) and (4) describes time and energy penalties incurred in a cache miss. Terms \( t_{\text{fetch}} \) and \( e_{\text{fetch}} \) respectively represent time and energy spent on retrieving the missing cache block from the next level in the memory subsystem, while \( t_{\text{write}} \) and \( e_{\text{write}} \) include the time and energy to write the fetched block to the cache. For the caches in the ith level of the hierarchy (Li), miss penalties depend on the properties of the next cache level (Li+1). Penalties for the misses occurring at the last cache level (L_n) depends on the properties of the DRAM. However, while configuring the caches in Li, there are no caches existing in the level Li+1. Using the timing and energy values from the DRAM for Li, using \( t_{\text{fetch}} \) and \( e_{\text{fetch}} \) in a level Li (where 1 ≤ i < n) yield unrealistic values for Ti and Ei. This is because a next level cache will exist in the final system, and access times and energy for a DRAM is several orders higher compared to a cache. Therefore we assume a miss in the current cache level Li will be a hit in level Li+1, only to calculate \( t_{\text{fetch}} \) and \( e_{\text{fetch}} \). Values for the four parameters in (3) and (4) are also obtained through CACTI 6.5 tool, which takes the contention when accessing a shared memory device into account. Calculation of \( t_{\text{fetch}} \) and \( e_{\text{fetch}} \) need to be done only once for all the configurations of a particular cache, since the penalties of fetching from the next level will be the same for all current level configurations.

The complete flow of the selection process is described in Algorithm 1, which iteratively explores the cache hierarchy. In a single iteration (lines 1-16) the algorithm finds the suitable configuration for all \( m_i \) levels in the cache level Li. Parallel exploration of design spaces for all caches in a given level, to calculate cache hit rates for different configurations, is given in lines 2 and 3. This step is carried out in the FPGA using hSim modules, simultaneous to the execution of application in the MPSoC. Lines 5-10 describe the estimation of energy and performance measures using analytical models. When assessing the miss penalties, access time and energy of the next cache level is

### Algorithm 1: Configuring an n-level Cache Hierarchy with \( m_i \) caches at level \( L_i \)

1. for each cache level \( L_i \), where \( i=1 \) to \( n \) do
2.   for each configuration \( c_{L_i} \) do
3.     Calculate hit rates \( h_{c_{L_i}} \) for all the configurations \( c_{L_i} \) using real-time extracted memory access traces. (Done in parallel on the FPGA)
4.   for each cache level \( L_j \) in level \( L_i \), where \( j=1 \) to \( m_i \) do
5.     Estimate \( t_{\text{access}} \) and \( E_{\text{access}} \) for \( c_{L_j} \)
6.     if \( i < n \) then
7.       Estimate \( t_{\text{miss}} \) and \( e_{\text{miss}} \) for \( c_{L_j} \) using \( t_{\text{fetch}} \) and \( e_{\text{fetch}} \) values of \( L_{i+1} \)
8.     else if \( i = n \) then
9.       Estimate \( t_{\text{miss}} \) and \( e_{\text{miss}} \) for \( c_{L_j} \) using \( t_{\text{fetch}} \) and \( e_{\text{fetch}} \) values of DRAM
10. // Find \( c_{L_j}^{\text{selected}} \) for the cache \( L_j 
11. if best performance then
12.   Select \( c_{L_j} \) with minimum \( T_{\text{cache}} \)
13. else if least energy consumption then
14.   Select \( c_{L_j} \) with minimum \( E_{\text{cache}} \)
15. Include a cache \( L_j \) with configuration \( c_{L_j}^{\text{selected}} \) into the MPSoC
16. Re-synthesize the MPSoC on the FPGA
A multi-level cache hierarchy in an MPSoC requires memory accesses to be extracted from different points in the memory subsystem in order to carry out the simulation of cache configurations. Therefore, we design a hardware cache simulator module (hSim) using the simulation core by the authors of [13] such that it can be connected to different positions in the memory subsystem of the MPSoC operating on the FPGA device. The module was designed using VHDL. Figure 6 illustrates the interfacing details of hSim, which consists of three ports. The first port connects to the previous cache level of the memory hierarchy (to receive memory addresses) and the second port connects to the next level. The third port is used for control signals such as enabling and disabling of the hSim module. The control signals can be sent from any processor in the MPSoC. Address and data widths for the ports are parameterized and hence customizable upon requirement. With these ports, a number of hSim module can be flexibly connected to different points in the memory hierarchy. There are two clock signal inputs associated with the hSim module: main system clock for interface operations; and a separate clock for simulator core. The schematic symbol of an hSim module as implemented in Altera Qsys system integration tool [21]. Widths of the address and data signals are configurable.

The module can be connected in the place of a cache memory, to simulate different configurations for that particular cache in the hierarchy. The memory accesses, which are coming from the processor or the previous cache(s) in the hierarchy, are passed through to the next level cache. Accesses coming from different sources can be connected in combination to the input port of the hSim module, which enables it to simulate configurations for shared caches. Multiple instances of the hSim module can be connected to the MPSoC memory hierarchy as shown in Fig. 8, to simulate configurations for a set of private L1 caches in parallel by using addresses extracted at real-time, making the simulation accurate compared to software methods where each access trace is derived from a combined trace [12].

VI. EXPERIMENTAL SETUP

We used the target system of Fig. 3 in our experiments to demonstrate the process of cache exploration. Since we do not assume a hardware cache coherence mechanism to be present in the final system and hence do not calculate cache misses occurring due to maintaining coherence, separate application programs were executed on the four processors. Even though data aren’t shared between programs, the sharing of L2 cache affects the hit rates of different configurations on L1 and L2. The MPSoC was built using Altera Qsys system integration tool [21], with four Nios II/ embedded processor cores [22] at 200MHz, and was deployed in a Stratix V GX FPGA on an Altera DE5-NET board [23]. We used 1 gigabyte of DDR3 SDRAM at 800MHz on the DE5-NET board as the main memory for the set of processors.

We used 6 benchmark applications from SPEC2006 benchmark suite (bzip2 compression, bzip2 de-compression) and MiBench suite (lame mp3-encoding, lame mp3-decoding, rijndael aes-encryption, jpeg) to create two groups of four applications each. Table I shows the two groups of applications, the sizes of the data inputs used and the memory accesses generated by each application.

Four instances of the hSim module, each operating at 100MHz, were connected to the cache-less MPSoC in order to simulate the cache hits for L1 cache configurations for the four processors in parallel. Each of the L1 hSim modules were parameterized to simulate 27 different configurations as described in Table II (27 configurations were used since Nios II cache module is direct mapped only). Energy and performance measures were calculated as described in Section IV using
TABLE I: APPLICATIONS USED IN OUR EXPERIMENTS

<table>
<thead>
<tr>
<th>Core</th>
<th>Application</th>
<th>Input size (kB)</th>
<th>Memory accesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>rjndael_aes</td>
<td>17</td>
<td>99,298,728</td>
</tr>
<tr>
<td>1</td>
<td>bzip2_decompress</td>
<td>18</td>
<td>66,365,689</td>
</tr>
<tr>
<td>2</td>
<td>jpeg</td>
<td>769</td>
<td>53,597,119</td>
</tr>
<tr>
<td>3</td>
<td>lame_decode</td>
<td>25</td>
<td>70,702,236</td>
</tr>
</tbody>
</table>

TABLE II: CONFIGURATIONS FOR PRIVATE L1 CACHES AND SHARED L2 CACHE

<table>
<thead>
<tr>
<th>Block size (Bytes)</th>
<th>Set size (Bytes)</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>4, 16, 32</td>
<td>1 - 256</td>
<td>1</td>
</tr>
</tbody>
</table>

Table of configurations used in experiment 1; and configurations showing minimum access times for applications.

Fig. 9: Energy Consumption against Access Time for shared L2 cache configurations.

VII. RESULTS

The results obtained from the simulations are presented in Fig. 9 and Fig. 10, with the Average Cache Access Energy ($E_{cache}$) on vertical axis plotted against Average Cache Access Time ($T_{cache}$) on horizontal axis. Each plot displays a subset of configurations in the respective design space, with low energy and access time values. Crosses represent different cache configurations explored. The configuration giving the least energy in the design space is marked with a red triangle, whereas the configuration giving the fastest access time is marked with a green circle.

Figure 10 reports the results for all private L1 caches. A designer can decide which configuration to select depending on the requirement and constraints. For example, for jpeg, the configuration with block size = 32 bytes, set size = 256 (8KB cache) gives the fastest access time with 99.2% hit rate; while the configuration with block size = 16 bytes set size = 32 (512B cache) gives the lowest energy consumption, with only 90.3% hit rate. It is worthwhile noting that rjndael_aes application observes minimum $E_{cache}$ with two different configurations, using different size inputs. When choosing L1 caches in the two experiments, we selected configurations showing minimum energy for applications in experiment 1; and configurations showing minimum access times for applications in experiment 2. Details of these selected L1 configurations are shown in Table III.

Using the selected configurations for private L1 caches in the MPSoC, Fig. 9 reports the results obtained for the shared L2 cache. Details of L2 cache configurations with minimum energy and access time are shown in Table IV. Experiment 1 obtained minimum energy using a 16KB cache with 99.8% hit rate while minimum access time was achieved by a 2KB cache with 98.9% hit rate.

Since this work focuses on calculating hit rates for the cache configuration design space using specialized hardware, it is of importance to assess the time consumed by the hSim modules to produce the results. In experiment 1, the simulation of 27 configurations for each of the four private L1 caches took 30.6 seconds with 289.9 million accesses processed in total by the four hSim modules. With the selected L1 configurations (giving minimum energy estimates) in the MPSoC, the combined trace of L1 misses was 42.3 million accesses. The hSim module simulating 180 shared L2 cache configurations took 6.3 seconds to process this trace. In experiment 2, four hSim modules (each simulating 27 L1 configurations) took 84.3 seconds to process a total of 694.7 million memory accesses. After the L1 caches with minimum access time estimates were put into the MPSoC, the hSim module simulating 180 shared L2 configurations observed just 23 million accesses. The L2 simulation in experiment 2 took 3.2 seconds to calculate the hit rates. The time and trace size details are tabulated...
This enables the possibility of accurately emulating coherent cache behaviours. We aim to extend this work to examine the coherence of caches.

REFERENCES


