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.**

I. 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 *Associativiy*. 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 Janapsatya 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.

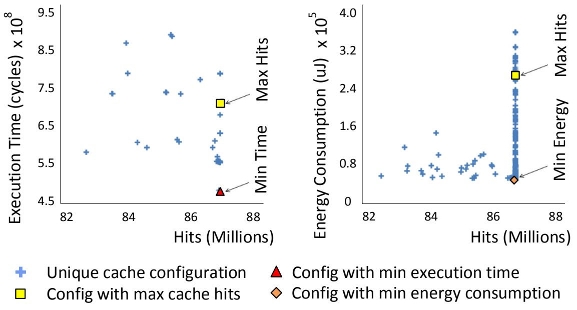

Fig. 1: (a) Execution time and (b) Energy consumption profiles for *G721 encoder* on a *Tensilica Xtensa* processor using different cache configurations, by Shwe et al. [1].

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-

Trace Size	12.46 Billion Accesses
Storage Space	129.4 GB
Time to Extract	**72 hours**
Simulation Time	**90 minutes**

Fig. 2: Software simulation of 44 cache configurations for MPEG2 encoder.

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 accesses 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], Janapsatya 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 Janapsatya'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 (upto 53 times faster than *Cheetah* simulator [6]). It is the fastest design tool presented yet to explore the cache configuration design space of 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 simualtion 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

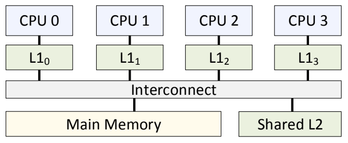

Fig. 3: Multiprocessor cache hierarchy with private L_1 caches and a shared L_2 cache.

This work focuses on shared memory multiprocessor systems with multi-level cache hierarchies containing private and shared caches, as depicted in Fig. 3. A hierachy 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 hierachy, 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 assisting 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.

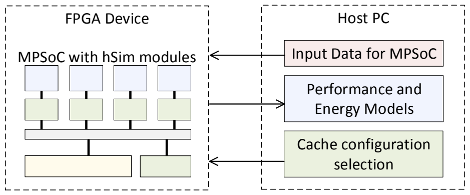

Fig. 4: Hybrid simualtion platform where cache hit rates are calculated on FPGA.

Initially, the MPSoC doesn't contain any cache. The cache hierarchy is explored in n stages, starting from the L_1 caches and moving down the hierarchy until the last cache level L_n. In the i^{th} stage, configurations for all m_i caches in level L_i 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 i^{th} 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.

$$T_{cache} = t_{access} + (1 - h_c) \times t_{miss} \tag{1}$$

$$E_{cache} = e_{access} + (1 - h_c) \times e_{miss} \tag{2}$$

The term t_{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_{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_{access} , representing the effects of the configuration, and the energy penalty for a cache miss is given by e_{miss}.

For all the configurations in the design space, the values of t_{access} and e_{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 used for this purpose depending on the designer's requirement, and the performance and accuracy of the model used 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 cache $L_i^j$ in level $L_i$ where j:=1 to $m_i$ do
3          Calculate hit rates ($h_c$) for all the configurations $c_{L_i^j}$ using real-time
           extracted memory access traces. (Done in parallel on the FPGA)
4      for each cache $L_i^j$ in level $L_i$ where j:=1 to $m_i$ do
5          for each configuration $c_{L_i^j}$ do
6              Estimate $t_{access}$ and $e_{access}$ for $c_{L_i^j}$
7              if $i < n$ then
8                  Estimate $t_{miss}$ and $e_{miss}$ for $c_{L_i^j}$, using $t_{fetch}$ and
                   $e_{fetch}$ values of $L_{i+1}$
9              else if $i = n$ then
10                 Estimate $t_{miss}$ and $e_{miss}$ for $c_{L_i^j}$, using $t_{fetch}$ and
                   $e_{fetch}$ values of DRAM
           // Find $c_{L_i^j-selected}$ for the cache $L_i^j$
11         if best performance then
12             Select $c_{L_i^j}$ with minimum $T_{cache}$
13         else if least energy consumption then
14             Select $c_{L_i^j}$ with minimum $E_{cache}$
15         Include a cache $L_i^j$ with configuration $c_{L_i^j-selected}$, into the MPSoC
16         Re-synthesize the MPSoC on the FPGA
```

beyond the scope of this work. Since h_c is provided by the simulation done in hardware, the only unknown terms are t_{miss} and e_{miss}.

$$t_{miss} = t_{fetch} + t_{write} \tag{3}$$

$$e_{miss} = e_{fetch} + e_{write} \tag{4}$$

Equations (3) and (4) describes time and energy penalties incurred in a cache miss. Terms t_{fetch} and e_{fetch} respectively represent time and energy spent on retrieving the missing cache block from the next level in the memory subsystem, while t_{write} and e_{write} include the time and energy to write the fetched block to the cache. For the caches in the i^{th} level of the hierarchy (L_i), miss penalties depend on the properties of the next cache level (L_{i+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 L_i, there are no caches existing in the level L_{i+1}. Using the timing and energy values from the DRAM for t_{fetch} and e_{fetch} in a level L_i (where $1 \leq i < n$) yield unrealistic values for T_{cache} and E_{cache}. 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 L_i will be a hit in level L_{i+1}, only to calculate t_{fetch} and e_{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_{fetch} and e_{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 caches in the level L_i. 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

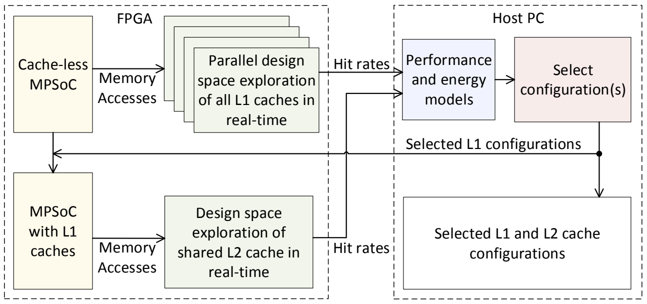
FPGA
Cache-less MPSoC
Memory Accesses
Parallel design space exploration of all L1 caches in real-time
Hit rates
Host PC
Performance and energy models
Select configuration(s)
Selected L1 configurations
MPSoC with L1 caches
Memory Accesses
Design space exploration of shared L2 cache in real-time
Hit rates
Selected L1 and L2 cache configurations

Fig. 5: Overview of the simulation methodology used to explore the design space of a multiprocessor cache hierarchy and determine suitable configurations.

considered. In the case of the last level cache, the access time and energy of the main memory is used. Making the selection of a cache configuration is given in Lines 11 to 14 and line 15 represents event of putting the selected cache into the MPSoC on the FPGA. Once all the caches in the level L_i are configured in this manner, the system is re-synthesized and the process moves to the level L_{i+1}. Figure 5 shows the flow of the exploration process for a 2-level cache hierarchy with private L1 caches and a shared L2 cache (Fig 3).

V. Implementation

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 implemeted in Altera Qsys system integration tool [21] is shown in Fig. 7. The ports can be adapted to other interconnect architectures using bridging components.

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 L_1 caches in parallel by using addresses extracted at real-time, making the

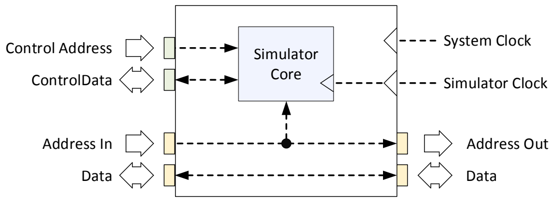

Fig. 6: Operation and interfacing overview of the hSim module.

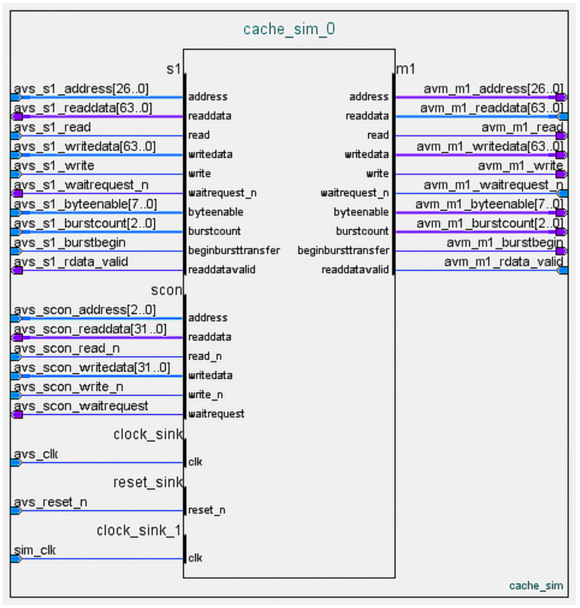

Fig. 7: Detailed schematic symbol showing all signals for the hSim module as implemeted in Altera Qsys system integration tool [21]. Widths of the address and data signals are configurable.

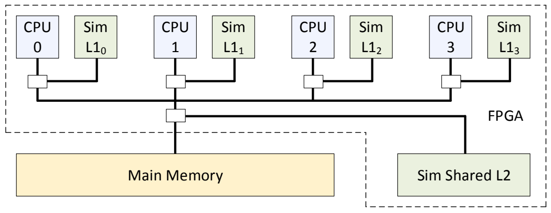

Fig. 8: Multiple hSim modules connected to a multiprocessor system on FPGA, illustarting how the module can be used to simulate private/shared/level1/level2 cache configurations.

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 demostrate 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 coherency, separate application programs were executed on the four processors. Even though data aren't shared between programs, the sharing of L_2 cache affects the hit rates of different configurations on L_1 and L_2. The MPSoC was built using Altera Qsys system integration tool [21], with four Nios II/f 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 L_1 cache configurations for the four processors in parallel. Each of the L_1 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

Core	Experiment 1			Experiment 2		
	Application	Input size (KB)	Memory acesses	Application	Input size (KB)	Memory acesses
0	rijndael aes	17	99,299,728	lame encode	3	259,070,294
1	bzip2 de-compress	18	66,365,689	bzip2 compress	131	143,691,935
2	jpeg	769	53,597,119	rijndael aes	41	238,329,483
3	lame decode	25	70,702,236	jpeg	769	53,597,114

TABLE II: CONFIGURATIONS FOR PRIVATE L_1 CACHES AND SHARED L_2 CACHE

27 configurations for each private L1 cache			180 configurations for shared L2 cache		
Block size (Bytes)	Set size	Associativity	Block size (Bytes)	Set size	Associativity
4, 16, 32	1 - 256	1	32 - 256	1 - 256	1 - 16

the hit rates obtained from the hSim modules, and L_1 caches were put into the MPSoC based on the selected configurations (we used minimum E_{cache} for experiment 1 and minimum T_{cache} for experiment 2). A single hSim module was connected to the MPSoC after the L_1 caches, to simulate the cache hits for the shared L_2 cache. L_2 simulator was parameterized to simulate 90 different configurations as described in Table II.

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 L_1 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 *rijndael_aes* application observes minimum E_{cache} with two different configurations, using different size inputs. When choosing L_1 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 L_1 configurations are shown in Table III.

Using the selected configurations for private L_1 caches in the

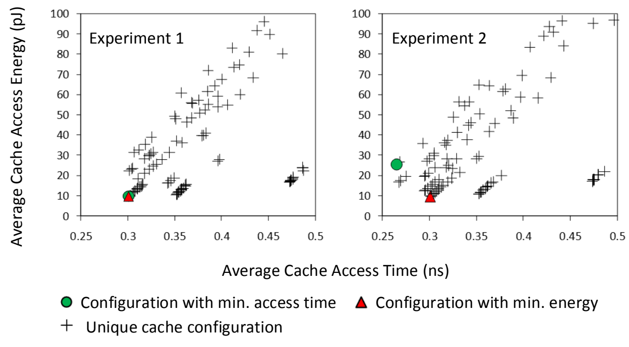

Fig. 9: Energy Consumption against Access Time for shared L_2 cache configurations.

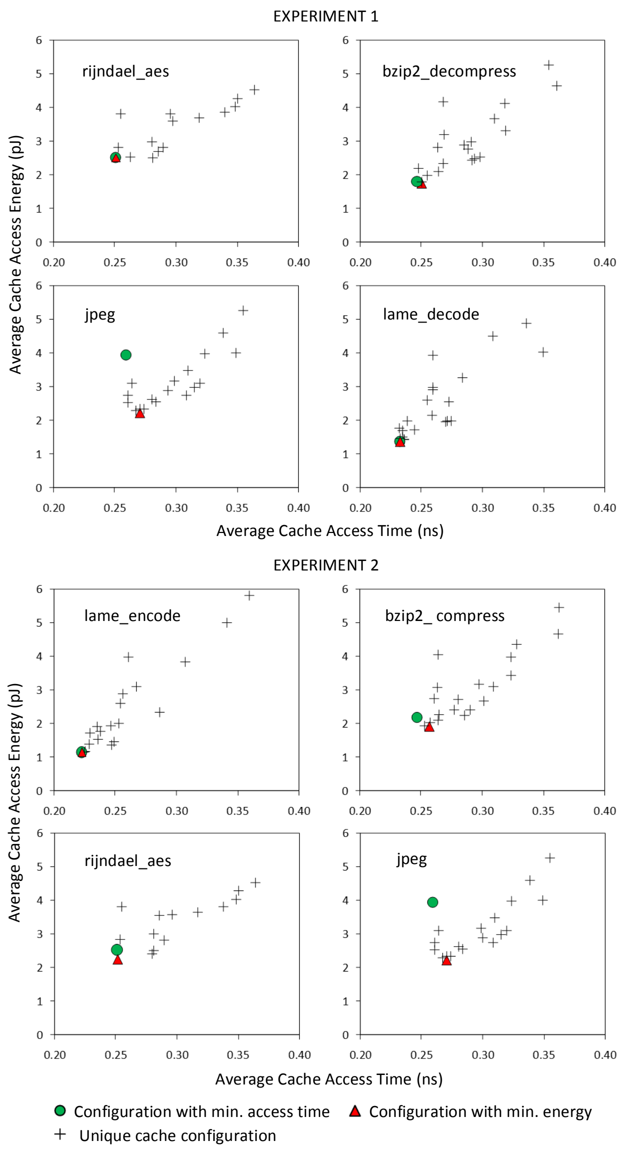

Fig. 10: Energy Consumption against Access Time for private L_1 cache configurations.

MPSoC, Fig. 9 reports the results obtained for the shared L_2 cache. Details of L_2 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 L_1 caches took 30.6 seconds with 289.9 million accesses processed in total by the four hSim modules. With the selected L_1 configurations (giving minimum energy estimates) in the MPSoC, the combined trace of L_1 misses was 42.3 million accesses. The hSim module simulating 180 shared L_2 cache configurations took 6.3 seconds to process this trace. In experiment 2, four hSim modules (each simulating 27 L_1 configurations) took 84.3 seconds to process a total of 694.7 million memory accesses. After the L_1 caches with minimum access time estimates were put into the MPSoC, the hSim module simulating 180 shared L_2 configurations observed just 23 million accesses. The L_2 simulation in experiment 2 took 3.2 seconds to calculate the hit rates. The time and trace size details are tabulated

TABLE III: PRIVATE L_1 CACHE CONFIGURATIONS WITH MINIMUM ACCESS TIME AND ENERGY FROM EXPERIMENTS 1 AND 2

Core	Application	Experiment 1										Application	Experiment 2									
		L_1 Config. with min. E_{cache}					L_1 Config. with min. T_{cache}						L_1 Config. with min. E_{cache}					L_1 Config. with min. T_{cache}				
		Block size	Set size	Assoc	Cache size	Hit rate	Block size	Set size	Assoc	Cache size	Hit rate		Block size	Set size	Assoc	Cache size	Hit rate	Block size	Set size	Assoc	Cache size	Hit rate
0	rijndael aes	32B	128	1	4KB	99.5%	32B	128	1	4KB	99.5%	lame encode	32B	16	1	512B	98.3%	32B	16	1	512B	98.3%
1	bzip2 de-compress	16B	32	1	512B	93.9%	32B	32	1	1KB	95.6%	bzip2 compress	16B	32	1	512B	92.7%	32B	64	1	2KB	96.9%
2	jpeg	16B	32	1	512B	90.3%	32B	256	1	8KB	99.2%	rijndael aes	32B	4	1	128B	86.8%	32B	128	1	4KB	99.4%
3	lame decode	32B	16	1	512B	96.6%	32B	16	1	512B	96.6%	jpeg	16B	32	1	512B	90.3%	32B	256	1	8KB	99.2%

TABLE IV: SHARED L_2 CACHE CONFIGURATIONS WITH MINIMUM ACCESS TIME AND ENERGY FROM EXPERIMENTS 1 AND 2

Experiment	Shared L_2 Config. with min. E_{cache}					Shared L_2 Config. with min. T_{cache}				
	Block size	Set size	Assoc	Cache size	Hit rate	Block size	Set size	Assoc	Cache size	Hit rate
1 (Group A)	256B	4	16	16KB	99.8%	256B	4	16	16KB	99.5%
2 (Group B)	128B	8	16	16KB	99.8%	128B	1	16	2KB	98.9%

TABLE V: SIMULATION TIMES TO CALCULATE HIT RATES IN HARDWARE

Simulation	Time (s)	Total Memory Accesses	Time per Million Accesses (s)
Experiment 1 – 4 private L_1 caches	30.6	289,964,772	0.106
Experiment 2 – 4 private L_1 caches	84.3	694,688,826	0.121
Experiment 1 – shared L_2 cache	6.3	42,360,645	0.149
Experiment 2 – shared L_2 cache	3.2	23,943,761	0.133

in Table V. These values represents time taken purely for simulation as in the prior works, excluding overheads of re-synthesizing the system on FPGA. The observed time per million memory accesses taken by hSim modules is from 0.106 seconds to 0.149 seconds. The software based method in [12] takes 68 seconds per million accesses to explore the design space of a similar multiprocessor cache hierarchy, with a total time of 9468 seconds for 139 millions of memory accesses. In comparison, our hardware based method took 87.5 seconds and 36.9 seconds for the two experiments demonstrated here, while processing larger memory access traces. Therefore the hardware based calculation of cache hit rates using hSim modules in the experiments is upto 456 times faster than software based simulation, owing to the parallel simulation in hardware.

Additionally, simulation in hardware uses the real memory access traces observed by caches in a multi-level hierarchy, as opposed to software simulation where memory access traces are derived. It should also be noted that simulation times are directly related to the number of memory accesses processed. In hardware based simulation, increasing the number of cache configurations to explore requires more FPGA logic elements, rather than increasing the simualtion time.

VIII. Conclusion

In this paper, we presented a hardware based design space exploration methodology to determine the cache configurations for a multi-level cache hierarchy in an application specific MPSoC. The proposed method significantly reduces the time taken during the design stages by rapidly calculating the cache hits for all configurations using specialized hardware. The hSim modules designed for this purpose can be flexibly connected to different points in an MPSoC cache hierarchy on an FPGA device, to extract and analyse the memory access information at those points in real-time. With such fast and flexible simulation made possible, designers can explore the cache hierarchy design space with ease. An additional benefit offered by using hSim modules is that actual memory accesses generated by processors can be observed at real-time. This enables the possibility of accurately emulating coherent cache behaviours. We aim to extend this work to examine the coherence of caches.

References

[1] S. M. M. Shwe, H. Javaid, and S. Parameswaran, "RExCache: Rapid Exploration of Unified Last-Level Cache," in *Proceedings of the 18th Asia and South Pacific Design Automation Conference (ASP-DAC'13)*. Ieee, Jan. 2013, pp. 582–587.

[2] A. Janapsatya and A. Ignjatovic, "Finding Optimal L1 Cache Configuration for Embedded Systems," in *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC'06)*, 2006, pp. 1–6.

[3] A. Agarwal, J. Hennessy, and M. Horowitz, "An Analytical Cache Model," *ACM Transactions on Computer Systems (TOCS)*, vol. 7, no. 2, pp. 184–215, May 1989.

[4] J. J. Pieper, A. Mellan, J. M. Paul, D. E. Thomas, and F. Karim, "High Level Cache Simulation for Heterogeneous Multiprocessors," in *Proceedings of the 41st annual Design automation Conference - (DAC'04)*. New York, New York, USA: ACM Press, 2004, p. 287.

[5] S. Ghosh, M. Martonosi, and S. Malik, "Cache Miss Equations: An Analytical Representation of Cache Misses," in *Proceedings of the 11th international conference on Supercomputing (ICS'97)*, 1997, pp. 317–324.

[6] R. A. Sugumar and S. G. Abraham, "Set-Associative Cache Simulation Using Generalized Binomial Trees," *ACM Transactions on Computer Systems (TOCS)*, vol. 13, no. February, pp. 32–56, 1995.

[7] M. Hill and A. Smith, "Evaluating Associativity in CPU Caches," *IEEE Transactions on Computers*, vol. 38, no. 12, pp. 1612–1630, 1989.

[8] M. S. Haque, J. Peddersen, A. Janapsatya, and S. Parameswaran, "SCUD : A Fast Single-pass L1 Cache Simulation Approach for Embedded Processors with Round-robin Replacement Policy," in *Proceedings of the Design Automation Conference (DAC'10)*, 2012, pp. 356–361.

[9] M. S. Haque, A. Janapsatya, and S. Parameswaran, "SuSeSim : A Fast Simulation Strategy to Find Optimal L1 Cache Configuration for Embedded Systems," in *Proceedings of the 7th IEEE/ACM International Conference on Hardware/Software Co-Design and System Synthesis (CODES+ISSS'09)*, 2009, pp. 295–304.

[10] W. Zang and A. Gordon-Ross, "T-SPaCS A Two-Level Single-Pass Cache Simulation Methodology," *IEEE Transactions on Computers*, vol. 62, no. 2, pp. 390–403, 2011.

[11] Y. Chen, J. Cong, and G. Reinman, "HC-Sim : A Fast and Exact L1 Cache Simulator with Scratchpad Memory Co-simulation Support," in *Proceedings of the 9th IEEE/ACM/IFIP International Conference on Hardware/Software Co-Design and System Synthesis (CODES+ISSS'11)*, 2011, pp. 295–304.

[12] M. Haque, R. Ragel, A. Ambrose, S. Radhakrishnan, and S. Parameswaran, "DIMSim : A Rapid Two-level Cache Simulation Approach for deadline-based MPSoCs," in *Proceedings of the 8th IEEE/ACM/IFIP International Conference on Hardware/Software Co-Design and System Synthesis (CODES+ISSS'12)*, 2012, pp. 151–160.

[13] J. Schneider, J. Peddersen, and S. Parameswaran, "A Scorchingly Fast FPGA-Based Precise L1 LRU Cache Simulator," in *Asia and South Pacific Design Automation Conference (ASP-DAC'14)*. IEEE, Jan. 2014, in press.

[14] M. D. Hill. Dinero IV Trace-Driven Uniprocessor Cache Simulator. [Online]. Available: http://pages.cs.wisc.edu/~markhill/DineroIV/

[15] N. Tojo, N. Togawa, M. Yanagisawa, and T. Ohtsuki, "Exact and fast L1 cache simulation for embedded systems," in *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC'09)*. Ieee, Jan. 2009, pp. 817–822.

[16] P. Viana, A. Gordon-Ross, E. Barros, and F. Vahid, "A Table-based Method for Single-pass Cache Optimization," in *Proceedings of the 18th ACM Great Lakes symposium on VLSI (GLSVLSI'08)*. New York, New York, USA: ACM Press, 2008, p. 71.

[17] M. S. Haque, A. Kumar, Y. Ha, Q. Wu, and S. Luo, "TRISHUL: A Single-pass Optimal Two-level Inclusive Data Cache Hierarchy Selection Process for Real-time MPSoCs," in *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC'13)*, 2013, pp. 320–325.

[18] M. Rawlins and A. Gordon-Ross, "CPACT - The conditional parameter adjustment cache tuner for dual-core architectures," in *IEEE 29th International Conference on Computer Design (ICCD)*. Ieee, Oct. 2011, pp. 396–403.

[19] E. S. Chung, M. K. Papamichael, E. Nurvitadhi, J. C. Hoe, and K. Mai, "PROTOFLEX: Towards Scalable, Full-System Multiprocessor Simulations Using FPGAs," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 2, no. 2, 2009.

[20] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Optimizing NUCA Organizations and Wiring Alternatives for Large Caches with CACTI 6.0," *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'07)*, pp. 3–14, Dec. 2007.

[21] Altera Qsys System Integration Tool. [Online]. Available: http://www.altera.com/products/software/quartus-ii/subscription-edition/qsys/qts-qsys.html

[22] Nios II/f Core: Fast for Performance-Critical Applications. [Online]. Available: http://www.altera.com/devices/processor/nios2/cores/fast/ni2-fast-core.html

[23] Altera DE5 Development and Education Board. [Online]. Available: http://www.altera.com/education/univ/materials/boards/de5/unv-de5-board.html