Compiler-in-the-Loop Design Space Exploration Framework for Energy Reduction in Horizontally Partitioned Cache Architectures

Aviral Shrivastava, Ilya Issenin, Nikil Dutt,
Sanghyun Park, and Yunheung Paek

***Abstract*—Horizontally partitioned caches (HPCs) are a power-efficient architectural feature in which the processor maintains two or more data caches at the same level of hierarchy. HPCs help reduce cache pollution and thereby improve performance. Consequently, most previous research has focused on exploiting HPCs to improve performance and achieve energy reduction only as a byproduct of performance improvement. However, with energy consumption becoming the first class design constraint, there is an increasing need for compilation techniques aimed at energy reduction itself. This paper proposes and explores several low-complexity algorithms aimed at reducing the energy consumption. Acknowledging that the compiler has a significant impact on the energy consumption of the HPCs, Compiler-in-the-Loop Design Space Exploration methodologies are also presented to carefully choose the HPC parameters that result in minimum energy consumption for the application.**

***Index Terms*—Compiler, design space exploration (DSE), energy reduction, horizontally partitioned cache (HPC), minicache, split cache.**

I. Introduction

Horizontally partitioned caches (HPCs) were originally proposed in 1995 by Gonzalez *et al.* [1] to improve the effectiveness of caches. An HPC architecture maintains multiple caches at the same level of hierarchy (usually a large main cache and a small minicache); however, each memory address is mapped to exactly one cache. HPCs are a very power-efficient microarchitectural feature and have been deployed in several current processors such as the Intel StrongARM [2] and the Intel XScale [3]. However, compiler techniques to exploit it are still in their nascent stages.

The original idea behind such cache organization is the observation that array accesses in loops often have a low-temporal locality. Each value of an array is used for a while and then not used for a long time. Such array accesses sweep the cache and evict the existing data (like frequently accessed stack data) out of the cache. The problem is worse for high-associativity caches that typically employ First-In First-Out page replacement policy. Mapping such array accesses to the small minicache reduces the pollution in the main cache and prevents thrashing, thus leads to performance improvements. Thus, an HPC is a simple, yet powerful architectural feature to improve performance. Consequently, most existing approaches for partitioning data between the HPCs aim at improving performance.

In addition to performance improvement, HPCs also result in a reduction in the energy consumption due to two effects. First is a direct consequence of performance improvement. Since partitioning the array and stack variable into different caches reduces the interference between them, resulting in performance improvement due to lesser number of cache misses, which is directly translated into energy improvement. The second and the more important reason is that typically the minicache is smaller in size than the main cache; therefore, the energy consumption per access of the minicache is smaller than the energy consumption per access of the main cache. Therefore, diverting some memory accesses to the minicache leads to a decrease in the total energy consumption. Note that the first effect is in-line with the performance goal and was therefore targeted by traditional performance improvement optimizations. However, the second effect is orthogonal to performance improvement. Therefore, energy reduction by the second effect was not considered by traditional performance oriented techniques. In fact, as we show in this paper, the second effect (of a smaller minicache) can lead to energy improvements even in the presence of slight performance degradation. Note that this is where the goals of performance improvement and energy improvement diverge.

The energy reduction obtained using HPCs is very sensitive on the HPC design parameters. Therefore, it is very important to explore the HPC design parameters and carefully tune these parameters to suite the application. Furthermore, as we demonstrate in this paper, compiler has a very significant impact on the power reduction achieved by the HPC architectures. Therefore, it is very important to include the "compiler effects" during the exploration and evaluating of HPC design parameters. However, traditional exploration techniques are *Simulation-Only* (SO) Design Space Exploration (DSE) techniques in which the compiler is used to compile the application once, and the executable generated is simulated on several architectural variations to find out the best one. SO DSE techniques are unable to include "compiler effects" in the exploration and evaluation of various HPC configurations. In this paper, we propose a *Compiler-in-the-Loop* (CIL) DSE methodology to systematically incorporate the compiler effect and, therefore, effectively explore and accurately evaluate HPC design parameters for energy reduction.

II. Compiler for HPC

A. Experimental Framework

The problem of energy optimization for HPCs can be translated into a data partitioning problem. The data memory that the program accesses is divided into pages, and each page can be independently and exclusively mapped to exactly one of the caches. The compiler's job is then to find the mapping of the data memory pages to the caches that leads to minimum energy consumption.

As shown in Fig. 1, we first compile the application and generate the executable. The *Page Access Information Extractor* calculates the number of times each page is accessed during the execution of the program. The *Data Partitioning Heuristic* finds the best mapping of the pages to the caches that minimizes the energy consumption of the target embedded platform. The *Data Partitioning Heuristic* can be tuned to obtain the best performing, or minimal energy data partition by changing the cost function *Performance/Energy Estimator*. The *Page Access Information Extractor* is implemented with the modified *sim-safe* simulator from SimpleScalar toolset [4].

Manuscript received August 1, 2007; revised January 24, 2008 and October 1, 2008. Current version published February 19, 2009. This work was supported in part by a generous grant from Microsoft, by IDEC, by Korea Science and Engineering Foundation (KOSEF) NRL Program grant funded by the Korea government (MEST) [R0A-2008-000-20110-0], and by Engineering Research Center of Excellence Program of Korea Ministry of Education, Science and Technology/Korea Science and Engineering Foundation [R11-2008-007-01001-0]. This paper was recommended by Associate Editor V. Narayanan.

A. Shrivastava is with the Department of Computer Science and Engineering, Arizona State University, Tempe, AZ 85281 USA (e-mail: Aviral.Shrivastava@asu.edu).

I. Issenin and N. Dutt are with the Center for Embedded Computer System, School of Information and Computer Science, University of California at Irvine, Irvine, CA 92697 USA (e-mail: isse@ics.uci.edu; dutt@ics.uci.edu).

S. Park and Y. Paek are with the School of Electrical Engineering and Computer Science, Seoul National University, Seoul 151-600, Korea (e-mail: shpark@optimizer.snu.ac.kr; ypaek@snu.ac.kr).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2009.2013275

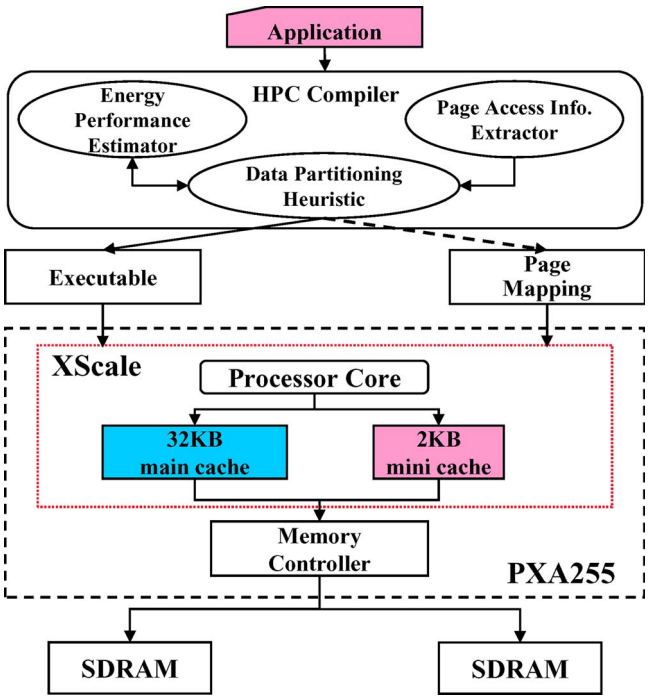

Fig. 1. Experimental Framework.

We use the memory latency as the performance metric and obtain the latency numbers using *sim-cache* simulator [4], modified to model HPCs. We use the memory subsystem energy consumption obtained by eCACTI [5] as the energy metric. The detailed parameters for the performance and the energy model can be found in [6].

We perform our experiments on applications from the MiBench suite [7] and an implementation of the H.263 encoder [8]. In the H.263 encoder, we encoded just one frame due to the simulation time, with the Quarter Common Intermediate Format resolution and 30 ft/s. To compile our benchmarks, we used Gnu Compiler Collection with all optimizations turned on.

B. Scope of Energy Reduction

To study the maximum scope of energy reduction achievable by page partitioning, we performed an exhaustive exploration of all possible page mappings and estimate their energy consumption. The five black bars in Fig. 3 (labeled as OPT) show that as compared to the case when all pages are mapped to the main cache, the scope of energy reduction is 55% on these set of benchmarks. Encouraged by the effectiveness of the page mapping, we develop several heuristics to partition the pages and see if it is possible to achieve high degrees of energy reduction using much faster techniques.

C. Page Partitioning Heuristics

We developed and examined three heuristics described in Fig. 2. Each heuristic takes a list containing pages sorted in the decreasing order of accesses and greedily maps pages to the main cache (M) and the minicache (m). The heuristic OP2A and OPA uses $evaluatePartitionCost(M, m)$ which performs simulation to estimate the performance or the energy consumption for a given partition. OA heuristic defines $k = ((minicache\_size)/(page\_size))$ as a cost metric, then the first k pages with the maximum number of accesses are mapped to the minicache, and the rest of the pages are mapped to the main cache. The time complexity of heuristic OP2A is $O(P^2A)$ due to the one level of backtracking, and the complexities of OPA and OA are $O(PA)$ and $O(A)$, respectively.

Heuristic OP2A(Page List PL)
01: $M = \phi, m = \phi, U = PL$
02: **while** $(U \neq \phi)$
03: $p = U.pop()$
04: $M1 = M + p, m1 = m, U1 = U$
05: **while** $(U1 \neq \phi)$
06: $p' = U1.pop()$
07: $cost1' = evaluatePartitionCost(M1 + U1 + p', m1)$
08: $cost2' = evaluatePartitionCost(M1 + U1, m1 + p')$
09: **if** $(cost1' \leq cost2')$ $M1+ = p'$ **else** $m1+ = p'$
10: **endwhile**
11: $M2 = M, m2 = m + p, U2 = U$
12: **while** $(U2 \neq \phi)$
13: $p' = U2.pop()$
14: $cost1' = evaluatePartitionCost(M2 + U2 + p', m2)$
15: $cost2' = evaluatePartitionCost(M2 + U2, m2 + p')$
16: **if** $(cost1' \leq cost2')$ $M2+ = p'$ **else** $m2+ = p'$
17: **endwhile**
18: $cost1 = evaluatePartitionCost(M1 + U, m1)$
19: $cost2 = evaluatePartitionCost(M2 + U, m2)$
20: **if** $(cost1 \leq cost2)$ $M+ = p$ **else** $m+ = p$
21: **endwhile**
22: **return** M, m

Heuristic OPA(Page List PL)
01: $M = \phi, m = \phi, U = PL$
02: **while** $(U \neq \phi)$
03: $p = U.pop()$
04: $cost1 = evaluatePartitionCost(M + U + p, m)$
05: $cost2 = evaluatePartitionCost(M + U, m + p)$
06: **if** $(cost1 \leq cost2)$ $M+ = p$ **else** $m+ = p$
07: **endwhile**
08: **return** M, m

Heuristic OA(Page List PL)
01: $M = \phi, m = \phi, U = PL$
02: **for** $(i = 0; i < \frac{mini-cache\_size}{page\_size}; i++)$
03: $m+ = U.pop()$
05: **endFor**
06: $M = U$
07: **return** M, m

Fig. 2. Heuristics: OP2A, OPA, and OA.

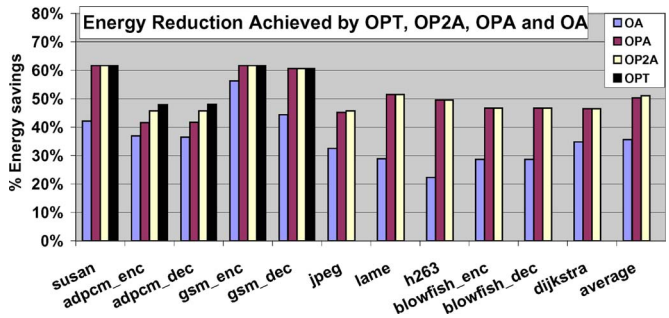

Fig. 3. Energy Reduction achieved by Exhaustive algorithm, OP2A, OPA, and OA.

Fig. 3 shows the energy reduction achieved by the minimum energy page partition found by our heuristics, as compared to the energy consumption when all the pages are mapped to the main cache. The main observation from Fig. 3 is that the minimum energy achieved by the exhaustive and the OP2A is almost the same. On average, the heuristics can achieve 52%, 50%, and 35% reductions in memory subsystem energy consumption, respectively. It should be noted that the heuristics OP2A and OPA can cover almost the maximum scope of the energy savings (55%) described in Section II-B, which demonstrates the quality of our page partitioning heuristics.

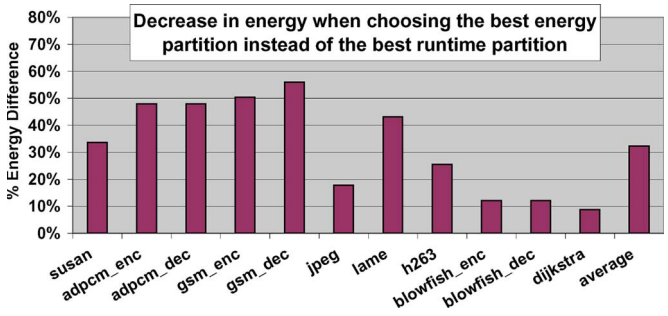

Fig. 4. Decrease in energy consumption, when we choose the best energy partition instead of the best performance partition.

TABLE I
POWER CONSUMPTION OF THE SYSTEM COMPONENT

System Component	Power Consumption
Memory Subsystem	450 mW
→ SDRAM	→ 332.5 mW
→ Cache (I + D)	→ 67.5 mW
→ Memory Bus (I + D)	→ 50 mW

D. Discussion

1) Optimizing for Energy Is Different From Optimizing for Performance: This experiment investigates the difference in optimizing for energy and optimizing for performance. We find the partition that results in the least memory latency, and the partition that results in the least energy consumption. Fig. 4 shows $(E_{\rm br} - E_{\rm be})/(E_{\rm br})$, where $E_{\rm br}$ is the memory subsystem energy consumption of the partition that results in the least memory latency, and $E_{\rm be}$ is the memory subsystem energy consumption by the partition that results in the least memory subsystem energy consumption. For the first five benchmarks (*susan—gsm_dec*), the number of pages in the footprint were small, so we could explore all the partitions. For the last six benchmarks (*jpeg—dijkstra*), we took the partition found by the OP2A heuristic as the best partition, as OP2A gives close-to-optimal results in the cases when we were able to search optimally. The graph essentially plots the decrease in energy if you choose the best energy partition as your design point. The decrease in the energy consumption is up to 56% and on average 32% for this set of benchmarks. In this experiment, the increase in memory latency was on average 1.7% and 5.8% in the worst case. It implies that choosing the best energy partition results in significant energy savings at a minimal loss in performance.

2) Total Processor Power Reduction: In this section, we analyze the total system power and see how the energy savings achieved by our algorithm is translated into the total processor energy reduction. Since we are only considering the data cache, data bus, and data memory, we assume that the power consumption to access the instruction and data is the same. This is a very conservative assumption in the sense that the instruction bus and the instruction memory are not used much since the instruction cache has very good performance due to its locality. It should be also noted that we consider the cache of the Intel XScale which consumes 15% of the processor power; the caches of the several other embedded processors consume up to 50% of the processor power. We summarized the memory subsystem power consumption of the system in Table I. We assume that the Intel XScale core (including caches) consumes 450 mW [3], [9].

As we described in Section II-C, OP2A heuristic achieves 52% reduction in memory subsystem energy consumption. According to our assumption, the power consumption only for the data memory subsystem is $(332.5 + 67.5 + 50)/2 = 225$ mW. Thus, we achieve 225 mW $*$ 52% $=$ 117 mW out of 832.5 mW, which is translated to 14% of the whole processor power.

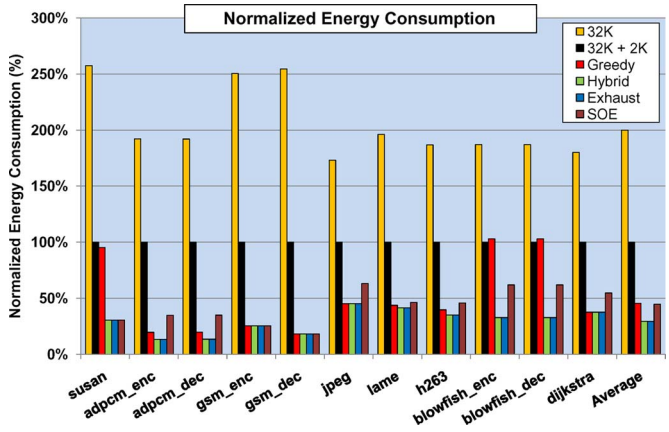

Fig. 5. Energy consumption normalized to the Intel XScale configuration.

III. CIL HPC DESIGN

A. Experimental Framework

So far, we have seen that HPC is a very effective microarchitectural technique to reduce the energy consumption of the processor. Another factor that has a significant impact on the energy consumption is the HPC configuration. Since the energy consumption per cache access significantly depends on the cache parameters (i.e., cache size and associativity), there has been an extensive research on the DSE of cache parameters aimed at reducing the energy consumption.

In the traditional DSE techniques, e.g., SO DSE, the binary and the page mapping is kept the same and the binary with the page mapping is executed on different HPC configurations. This strategy is not useful for HPC DSE, since it does not make sense to use the same page mapping after changing the HPC parameters. On the other hand, in our CIL DSE, we use an EXPRESSION processor description [10] which contains information about 1) HPC parameters, 2) the memory subsystem energy models, and 3) the processor and memory delay models. During the exploration, the processor information keeps changing along with the cache parameters. It should be also noted that we do not explore the design space of different cache line sizes and the replacement policies since it does not bring significant changes in the energy reduction. We set the cache line size to be 32 B as in the Intel XScale architecture. In total, we explore 33 minicache configurations for each benchmark.

We use the OPA page partitioning heuristic and generate binary executable along with the page mapping. The page mapping specifies to which cache (main or mini) each data memory page is mapped. The compiler is tuned to generate page mappings that lead to minimum memory subsystem energy consumption. The executable and the page mapping are both fed into a simulator which estimates the runtime and the energy consumption of the memory subsystem.

B. Scope of Energy Reduction by DSE

We first present experiments to estimate the importance of exploration of HPCs. To this end, we perform exhaustive CIL exploration of HPC design space and find out the minimum energy HPC design parameters. The first algorithm in Fig. 6 describes the exhaustive exploration algorithm. The algorithm estimates the energy consumption for each minicache configuration (line 02) and keeps track of the minimum energy. The function *estimate_energy* estimates the energy consumption for a given minicache size and associativity.

The bars labeled as "$32K$, $32K + 2K$ and *Exhaust*" in Fig. 5 compare the energy consumption of the memory subsystem with three cache designs, which are normalized to the case when there is a 2-kB minicache (the Intel XScale configuration).

TABLE II
OPTIMAL MINICACHE PARAMETERS

Benchmark	mini-cache conf.	Benchmark	mini-cache conf.
susan	2K, 4-way	lame	8K, 2-way
adpcm_enc	4K, 2-way	h263	8K, 2-way
adpcm_dec	8K, 2-way	blowfish_enc	16K, 2-way
gsm_dec	2K, direct mapped	blowfish_dec	16K, 2-way
gsm_enc	2K, direct mapped	dijkstra	8K, 2-way
jpeg	16K, 2-way		

We make two important observations from this graph. The first is that HPC is very effective in reducing the memory subsystem energy consumption. As compared to not using any minicache, using default minicache (the default minicache is 2-kB 32-way set associative) leads to an average of 2X reduction in the energy consumption of the memory subsystem. The second important observation is that the energy reduction obtained using HPCs is very sensitive on the minicache parameters. Exhaustive CIL exploration of the minicache DSE to find the minimum energy minicache parameters results in additional 80% energy reduction, thus reducing the energy consumption to just 20% of the case with a 2-kB minicache. This drastic energy reduction can be explained with Table II. Table II shows the energy optimal minicache configuration our exhaustive exploration found for each benchmark. Except for *susan* and *gsm* benchmark, the optimal minicache sizes are more than 2 kB, and this is the first source of the energy reduction. Since our page mapping technique is able to map more pages to the minicache, the minicache utilization increases, and the energy of the memory subsystem is reduced. The second and more interesting reason is the low associativity. The table suggests that low-associativity minicaches are good candidates to achieve low-energy solutions. This is because for several of these applications, the increase in energy is not compensated by the associativity. The energy consumption per access increases with the associativity. Thus, high associativity is beneficial to the energy consumption only if it is able to reduce the cache misses drastically and thereby reduce the number of accesses to the data memory. However, the benchmarks we used are multimedia applications and there is not much data reuse. Most of the misses in the caches are not conflict misses but compulsory misses, and the number of misses is not reduced even though the associativity increases. Thus, it turns out the default XScale cache configuration overconsumes the energy with its high associativity, and the minicache configuration our exhaustive exploration found results in impressive energy reduction.

C. CIL Exploration Algorithms

The first heuristic we develop for HPC CIL DSE is a pure greedy algorithm, outlined in Fig. 6. The greedy algorithm first greedily finds the cache size (lines 02–04), and then greedily finds the associativity (lines 05–07). The function *betterNewConfiguration* tells whether the new minicache parameters result in lower energy consumption than the old minicache parameters.

The bars labeled as "*Greedy*" in Fig. 5 plot the energy consumption when the minicache configuration is chosen by the greedy, normalized to the energy consumption when the default 32-kB main cache, and 2-kB minicache configuration is used. The plot shows that for most applications, greedy exploration is able to achieve good results, but for *blowfish*, and *susan*, the greedy exploration is unable to achieve any energy reduction; in fact, the solution it has found consumes even more energy than the base configuration. It comes from the separation of steps to find the cache size and the associativity. Our greedy exploration heuristic first increases the cache size with the minimal associativity (direct-mapped). Thus, it stops increasing the size if a bigger size cache with the minimal associativity is not beneficial even though a bigger size cache with higher associativity brings much more energy reduction. For *susan*, the greedy heuristic stops its exploration with the configuration of 512 B and two-way set associativity. However, on an average, the greedy CIL HPC DSE can reduce the energy consumption of the memory subsystem by 50%.

Since our greedy exploration algorithm fails to find the optimal configuration for some of the benchmarks, we developed a hybrid algorithm outlined in Fig. 6, to achieve the energy consumption close to the optimal configurations. The "*Hybrid*" bars in Fig. 5 plots the energy consumption when the minicache configuration is chosen by the hybrid algorithm. When compared to the energy reductions achieved with the greedy and exhaustive explorations, the graph shows that the hybrid exploration can always find out the optimal HPC configuration for our set of benchmarks.

```
ExhaustiveExploration()
01: minEnergy = MAX_ENERGY
02: foreach (c ∈ C)
03:   energy = estimateEnergy(c)
04:   if (energy < minEnergy)
05:     minEnergy = energy
06:   endIf
07: endFor
08: return minEnergy

GreedyExploration()
01: size = MIN_SIZE, asc = MIN_ASSOC
02: while (betterNewConf(size × 2, asc, size, asc))
03:   size = size × 2
04: endWhile
05: while (betterNewConf(size, asc × 2, size, asc))
06:   asc = asc × 2
07: endWhile
08: return estimateEnergy(size, asc)

HybridExploration()
01: size = MIN_SIZE, asc = MIN_ASSOC
02: while (betterNewConf(size × 4, asc, size, asc))
03:   size = size × 4
04: endWhile
05: done = false
06: while (!done)
07:   if (betterNewConf(size × 2, asc, size, asc))
08:     size = size × 2
09:   else if (betterNewConf(size, asc × 2, size, asc))
10:     asc = asc × 2
11:   else if (betterNewConf(size ÷ 2, asc, size, asc))
12:     size = size ÷ 2
13:   else if (betterNewConf(size ÷ 2, asc ÷ 2, size, asc))
14:     size = size ÷ 2, asc = asc ÷ 2
15:   else if (betterNewConf(size ÷ 2, asc × 2, size, asc))
16:     size = size ÷ 2, asc = asc × 2
17:   else
18:     done = true
19: endWhile
20: return estimateEnergy(size, asc)

betterNewConf(size', asc', size, asc)
09: if (!existsCacheConfig(size', asc'))
10:   return false
11: energy = estimateEnergy(size, asc)
12: energy' = estimateEnergy(size', asc')
13: return (energy' < energy)
```

Fig. 6. Exhaustive Exploration Algorithm.

D. Discussion: Importance of CIL DSE

In this section, we demonstrate that although SO DSE can also find HPC configurations with lesser memory subsystem energy consumption, it does not do as well as CIL DSE.

We perform SO DSE of HPC design parameters in which we compile once for the 32 kB/2 kB (i.e., the original XScale cache configuration) to obtain an executable and the minimum energy page mapping. While keeping these two the same, we explored all the HPC configurations to find the HPC design parameters which minimize the memory subsystem energy consumption. The rightmost bars in Fig. 5 plots the energy consumption of the HPC configuration found by the SO DSE, which is normalized to energy consumption of the 32 kB/2 kB configuration. The important observation to make from this graph is that although even SO DSE can find out HPC configurations which result in on average 57% memory subsystem energy reduction, CIL DSE is much more effective and can uncover HPC configurations that result in 70% reduction in the memory subsystem energy reduction. It should be noted that the overhead of compilation time in CIL DSE is negligible, because simulation times are several orders of magnitude more than compilation times.

IV. Summary

HPCs are a simple yet powerful architectural feature of the modern embedded processors to improve the performance and energy consumption. Existing compilation techniques to exploit the minicache are complex and concentrate on performance improvements. In this paper, we proposed and evaluated several data partitioning heuristics aimed at energy reduction. We showed that simple greedy heuristics are effective and achieve up to 52% of the possible energy reduction, and that this trend extends to various cache configurations. Since the energy reduction obtained using HPCs is very sensitive on the HPC design parameters and the compiler has a significant impact on the energy reduction, we presented a CIL DSE framework to decide optimal HPC parameters for an application. Our experiments show that CIL DSE can find HPC design parameters that result in 70% reduction of the memory subsystem energy consumption, thus motivating for the need of DSE for HPCs.

References

[1] A. Gonzalez, C. Aliagas, and M. Valero, "A data cache with multiple caching strategies tuned to different types of locality," in *Proc. 9th ICS*, Barcelona, Spain, 1995, pp. 338–347.

[2] Intel Corporation, *Intel StrongARM SA-1110 Microprocessor Brief Datasheet*. [Online]. Available: http://www.datasheetcatalog.org/datasheets/90/361041_DS.pdf

[3] Intel Corporation, *Intel XScale(R) Core: Developer's Manual*. [Online]. Available: http://www.intel.com/design/intelxscale/273473.htm

[4] D. Burger and T. M. Austin, "The simplescalar tool set, version 2.0," *ACM SIGARCH Comput. Archit. News*, vol. 25, no. 3, pp. 13–25, 1997.

[5] M. Mamidipaka and N. Dutt, "eCACTI: An enhanced power estimation model for on-chip caches," CECS, UCI, Irvine, CA, Tech. Rep. TR-04-28, 2004.

[6] A. Shrivastava, I. Issenin, and N. Dutt, "Compilation techniques for energy reduction in horizontally partitioned cache architectures," in *Proc. Int. Conf. CASES*, San Francisco, CA, 2005, pp. 90–96.

[7] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *Proc. WWC*, 2001, pp. 3–14.

[8] ITU-T/SG16 Video Coding Experts Group, "Video codec test model near-term, version 11 (TMN11)," Document Q15-G16, 1999.

[9] G. Contreras, M. Martonosi, J. Peng, R. Ju, and G. Y. Lueh, "XTREM: A power simulator for the Intel XScale core," in *Proc. ACM SIGPLAN/SIGBED Conf. LCTES*, Washington, DC, 2004, pp. 115–125.

[10] A. Halambi, P. Grun, V. Ganesh, A. Khare, N. Dutt, and A. Nicolau, "Expression: A language for architecture exploration through compiler/simulator retargetability," in *Proc. Conf. DATE*, Munich, Germany, 1999, pp. 485–490.