# A Compiler-in-the-Loop Framework to Explore Horizontally Partitioned Cache Architectures

Aviral Shrivastava†
Aviral.Shrivastava@asu.edu

Ilya Issenin
isse@ics.uci.edu

Nikil Dutt
dutt@ics.uci.edu

†Compiler and Microarchitecture Labs
Department of Computer Science and Engineering
Arizona State University
Tempe, AZ, USA

Center for Embedded Computer Systems
Department of Information and Computer Science
University of California Irvine
Irvine, CA, USA

## ABSTRACT

*Horizontally Partitioned Caches (HPCs) are a promising architectural feature to reduce the energy consumption of the memory subsystem. However, the energy reduction obtained using HPC architectures is very sensitive to the HPC parameters. Therefore it is very important to explore the HPC design space and carefully choose the HPC parameters that result in minimum energy consumption for the application. However, since in HPC architectures, the compiler has a significant impact on the energy consumption of the memory subsystem, it is extremely important to include compiler while deciding the HPC design parameters. While there has been no previous approaches to HPC design exploration, existing cache design space exploration methodologies do not include the compiler effects during DSE. In this paper, we present a Compiler-in-the-Loop (CIL) Design Space Exploration (DSE) methodology to explore and decide the HPC design parameters. Our experimental results on HP iPAQ h4300-like memory subsystem running benchmarks from the MiBench suite demonstrate that CIL DSE can discover HPC configurations with up to 80% lesser energy consumption than the HPC configuration in the iPAQ. In contrast, tradition simulation-only exploration can discover HPC design parameters that result in only 57% memory subsystem energy reduction. Finally our hybrid CIL DSE heuristic saves 67% of the exploration time as compared to the exhaustive exploration, while providing maximum possible energy savings on our set of benchmarks.*

## I. INTRODUCTION

Embedded systems design is fascinating owing to the interplay of multi-dimensional design considerations applicable to them. On one hand high performance is required of embedded systems, while on the other hand, the weight of the embedded system should be low and the battery life (energy consumption) long. While a custom ASIC design might have been a good solution at this point, shortening time to market and the frequent upgrade-ability and flexibility requirements shift embedded systems designers towards processor based embedded systems. In such systems the memory subsystem including the caches, buses, and the memory can use majority of the energy budget. Consequently memory subsystem energy reduction has been a very important design goal in the past two decades.

Although originally proposed for performance improvements, Horizontally Partitioned Cache (HPC) architectures have been found very effective in reducing the energy consumption of the memory subsystem [18]. In this popular technique, processor maintains multiple caches at the same level of memory hierarchy, and each memory page is mapped to exactly one of these caches. HPC is a popular cache architecture in embedded systems. For example, in the Intel XScale, the main cache is 32 KB, and is augmented by a 2 KB mini-cache, which is at the same level of memory hierarchy (namely L1). Henceforth in this paper we will call the additional cache as the *mini-cache* and the original cache as the *main cache*. The mapping of the page to the cache is specified as a page attribute and set while loading the application. The page mapping attribute is present in the Translation Look-aside buffer (TLB). On a cache access first a TLB lookup is performed to find out if the page is in the cache or not, and if yes, in which cache. Thus for each memory request only one cache lookup is performed. The idea of HPC architectures was originally proposed by Gonzalez et al. in [4] to improve the performance by separating and thus reducing the interference between the array and stack variables.

However HPCs also improve the energy consumption of the memory subsystem due to two main reasons. 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 directly translates into energy improvement. The second and the more important reason is that typically the mini-cache is smaller in size than the main cache; therefore the energy consumption per access of the mini-cache is smaller than the energy consumption per access of the main cache. As a result, diverting some memory accesses to the mini-cache leads to a decrease in the total energy consumption. In summary, HPC is a simple yet very effective technique to reduce the power consumption in embedded processors.

However, as we demonstrate in this paper, 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 the these parameters to suite the application. Furthermore, [18] demonstrated that compiler can have a very significant impact on the power

978-1-4244-1922-7/08/$25.00 ©2008 IEEE

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.

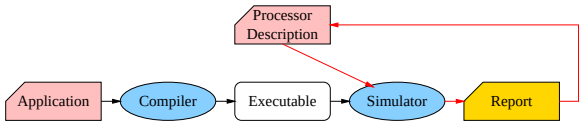


Fig. 1. Simulation-only Exploration

However traditional exploration techniques are *Simulation-Only* (SO) Design Space Exploration (DSE) techniques. Figure 1 describes the simulation only exploration methodology, in which the compiler is used to compile the application once. 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 and therefore effectively explore and accurately evaluate HPC design parameters for energy reduction.

In CIL DSE the application is first compiled for the given HPC configuration. The executable generated is then simulated on the same HPC configuration to accurately estimate the power, and performance corresponding for that HPC configuration.

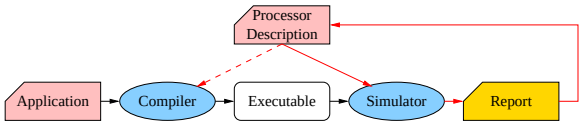


Fig. 2. Compiler-in-the-Loop Exploration

To demonstrate the need and usefulness of our approach, we perform experiments on an HP iPAQ 4300-like [6] memory subsystem. The HP iPAQ 4300 is intended for the wireless and handheld market. Consequently we perform experiments by running benchmarks from the MiBench suite [5], which are also intended to represent applications of the same domain.

The main results of this paper are:

- First, we demonstrate that as compared to the default configuration of the Intel XScale, our exploration can find HPC design parameters that can reduce the memory subsystem energy consumption by 80% .
- Second, we perform the traditional SO DSE, and find out that SO DSE can discover HPC design parameters that reduces the memory subsystem energy consumption by only 57%. In comparison with SO DSE, CIL DSE can discover HPC design parameters that are 30% superior in memory subsystem energy consumption.
- Finally, we develop a hybrid heuristic for CIL DSE, which can save 67% exploration time as compared to exhaustive CIL DSE scheme, while achieving maximum possible energy savings on our set of benchmarks.

## II. Related Work

### A. Horizontally Partitioned Caches

Caches are one of the major contributors of not only to the system power and performance, but also of the embedded processor area and cost. As a result, several techniques have been proposed to improve the effectiveness of caches.

Horizontally Partitioned Caches (HPCs) is an architectural technique was proposed by Gonzalez et al. [4] to separate and thus reduce the interference between the array and stack variables. The compiler's job in such architectures is to partition the data exclusively between the main cache and the mini-cache. The performance and energy improvements obtained using HPCs are highly sensitive to the compilation technique used. Consequently most earlier research focuses on achieving performance improvements using horizontally partitioned caches. Early techniques divided the data into main cache and the mini-cache based on reuse history. While some approaches use effective address reuse information [11, 15], others use program counter reuse information [19]. Software based approaches attempt to improve performance primarily by coarse-grain region based scheduling [12, 20]. Such schemes simply map the stack data, or the scalar variables to the mini-cache. Xu et al. [21] proposed a profile-based technique to partition the virtual pages to different caches.

However other than achieving energy reduction just due to performance improvements, HPC architectures reduce energy due to the lower energy per access to the smaller cache. Note that this energy reduction mechanism is not in line with performance improvements. Energy reduction by this mechanism requires mapping more pages to the smaller cache, which may result in performance penalty. Existing techniques that focus on performance improvements could not tap into this reason of energy reduction. [18] demonstrated that with a minimal performance penalty the energy consumption of the memory subsystem can be greatly reduced.

As compared to all these previous works, which try to improve power, and/or performance using an HPC configuration, this work deals with deciding on HPC design parameters for an application.

### B. Design Space Exploration

To start with, we are not aware of any DSE methodology for HPCs. However DSE methodologies are as old as the computer architecture. However, traditionally software and hardware development have been separated using the concept of *Instruction Set Architecture*. Consequently, the Compiler-assisted design has only been popular in the design of ISAs, and not in the design of the processor microarchitecture. However recently, several microarchitectural techniques have been proposed that operate under software control, e.g., software-controlled prefetching, software controlled power management. Such techniques blurs the hardware-software interface, and need to be explored while considering compiler effects. Consequently Compiler-in-the-Loop exploration frameworks have been recently proposed to explore processor design space, e.g., [17] and [3] propose compiler assisted DSE frameworks to explore the bypasses in processors. Similarly Halambi02 pro-

poses a compiler-assisted DSE framework to explore reduced Instruction Set Architectures (rISA).

HPC is one such microarchitectural feature, in which software has to decide the page partitioning between the two caches. Since the compiler has such a significant impact on the effectiveness of HPC architecture, we propose a CIL DSE method to decide on the HPC parameters.

## III. HPC Exploration Framework

To perform exploration of the HPC design space, we have developed a *Compiler-in-the-Loop* Design Space Exploration (DSE) framework, as depicted in Figure 3. The CIL DSE framework is centered around a textual description of the processor. In specific, for our purposes, the processor description contains information about i) Horizontally Partitioned Cache parameters, ii) the memory subsystem energy models, and iii) the processor and memory delay models.

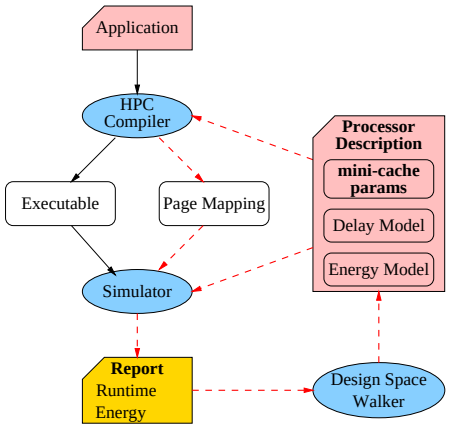


Fig. 3. Compiler-in-the-Loop methodology to explore the design space of HPCs

### A. HPC Compiler

We use the compilation technique OMN proposed in [18] as our HPC compiler, 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.

### B. HPC Simulator

The executable and the page mapping are both fed into a simulator which estimates the runtime and the energy consumption of the memory subsystem. Our simulator models the processor and memory subsystem of an HP iPAQ 4300. The iPAQ uses the Intel PXA255 processor [7] with the XScale core [8], which has a 32KB main cache and 2KB mini-cache. The system also has 64 MBytes of external SDRAM. We modify the *sim-cache* simulator of the simplescalar suite [1] and tune the sim-cache to model this memory system. The simulator provides us with the runtime of the applications in processor cycles ($rt$), and the number of accesses to the main cache ($aMc$), the number of accesses to the mini cache ($amc$)and the number of accesses to the memory ($M$).

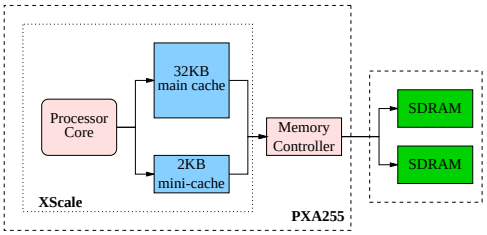


Fig. 4. Modeled memory subsystem

### C. Energy Models

Figure 4 shows the memory subsystem of the iPAQ that we have modeled to obtain performance and energy estimates. To model the energy consumption of the on-chip caches we use energy models from eCACTI [13] at 0.18 $u$ technology. As compared to CACTI [16], eCACTI provides better energy estimates, especially for high associativity caches, since it models sense-amps more accurately and scales device widths according to the capacitive loads. Thus we get the energy consumption per access of the main cache ($eMc$), and the energy consumption per access of the mini cache ($emc$) using the power model in eCACTI.

We find out the energy per burst access to the memory, $eM$ using models of external SDRAM memory from the low-power 32MB Micron MT48V8M32LF [14]. We estimate the energy consumed in the buses per access burst, $eb$ using power models from the PXA255 and Intel 440 MMX chipset power models [7,9,10]. The total power consumption of external memory and buses is $eM + eb = 42$ nJ for a 32-byte read/write burst.

We can then compute the energy consumed by the memory subsystem as $(eMc \times aMc) + (emc \times amc) + aM \times (em + eb)$.

### D. Benchmarks

We perform our experiments on applications from MiBench suite [5] and an implementation of H.263 encoder [2]. We choose this set of benchmarks, as they are representative of the wireless and handheld applications that is the target of the HP iPAQ 4300.

### E. Design Space Walker

The Design Space Walker performs HPC design space exploration by updating the HPC design parameters in the processor description. The mini-cache, that is configured by Design Space Walker, is specified using two attributes, the mini-cache size and the mini-cache associativity. For our experiments, we vary cache size from 256 bytes to 32 KB, in exponents of 2. We explore the whole range of mini-cache associativities, i.e., from direct mapped to fully associative. We do not model mini-cache configurations which cannot be modeled by eCACTI. We set the cache line size to be 32 bytes as in the Intel XScale architecture. In total we explore 33 mini-cache configurations for each benchmark.

The Design Space Walker implements various DSE algorithms described in Section VI. In the rest of this paper we develop and explore several DSE schemes employed in the Design Space Walker. The techniques differ in the exploration

```
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
```

Fig. 5. Exhaustive Exploration Algorithm

time and the minimum energy consumption HPC configuration they are able to discover.

## IV. Exhaustive Exploration

We first present experiments to estimate the importance of exploration of Horizontally Partitioned Caches. To this end, we perform exhaustive CIL exploration of HPC design space, and find out the minimum energy HPC design parameters.

Figure 5 describes the exhaustive exploration algorithm. The algorithm estimates the energy consumption for each mini-cache configuration (line 02), and keeps track of the minimum energy. The function *estimate_energy*, estimates the energy consumption for a given mini-cache size and associativity.

Figure 6 compares the energy consumption of the memory subsystem with 3 cache designs. The leftmost bar represents the energy consumed by the memory subsystem when the system has only a 32 KB main cache (no mini-cache is present.) The second, or the middle bar shows the energy consumed when there is a 2 KB mini-cache in parallel with the 32 KB cache, and the application is compiled to achieve minimum energy. The third and the rightmost bar represents the energy consumed by the memory subsystem, when the mini-cache parameters (size and associativity) are chosen using exhaustive CIL exploration. All the energy values are normalized to the case when there is a 2 KB mini-cache (the Intel XScale configuration.) The last set of bars is the average over the applications.

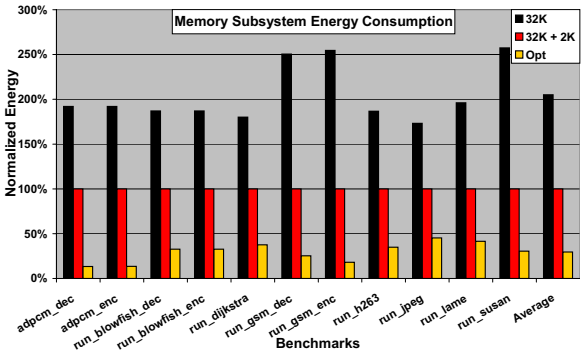


Fig. 6. Energy savings achieved by exploration

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 using not using any mini-cache, using default mini-cache (the default mini-cache 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 mini-cache parameters. Exhaustive CIL exploration of the mini-cache DSE to find the minimum energy mini-cache results in additional 80% energy reduction, thus reducing the energy consumption to just 20% of the case with a 2 KB mini-cache.

Furthermore, the performance of the energy-optimal HPC configuration is very close to the performance of the best performing HPC configuration. The performance degradation was no more than 5% and was 2% on average. Therefore energy-optimal HPC configuration achieves high energy reductions at minimal performance cost.

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

TABLE I
Optimal mini-cache parameters

Figure I shows the energy optimal mini-cache configuration for each benchmark. The table suggests that low-associativity mini-caches are good candidates to achieve low-energy solutions.

## V. Importance of Compiler-in-the-Loop Exploration

Our next set of experiments are to show that although SO DSE can also find HPC configurations with lesser memory subsystem energy consumption, it does not do as well as CIL DSE.

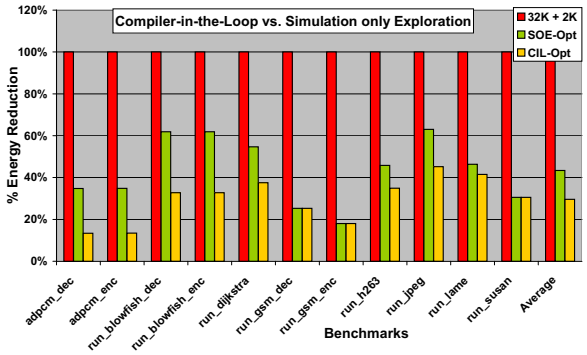


Fig. 7. CIL versus SO exploration

To this end, we perform SO DSE of HPC design parameters. We compile once for the 32KB/2KB (i.e. the original XScale cache configuration) to obtain an executable and the minimum energy page mapping. While keeping these two same, we explored all the HPC configurations to find the HPC design parameters which minimize the memory subsystem energy consumption. Figure 7 plots the the energy consumption of the HPC configuration found by the SO DSE (middle bar) and CIL DSE (right bar), and the original Intel XScale HPC configuration (left bar) for each benchmark. The rightmost set of bars represent the average over all the benchmarks. All the energy

consumption values are normalized to energy consumption of the 32KB/2KB configuration.

The important observation to make from this graph 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.

## VI. HPC DSE Heuristics

We have demonstrated that CIL DSE of HPC design parameters is very useful and important to achieve significant energy savings. However since the mini-cache design space is very large, exhaustive exploration may consume a lot of time. In this section we explore heuristics for effective and efficient HPC DSE.

The rightmost bar in Figure 8 plots the energy consumption of the optimal configuration, as compared to the energy consumption when the XScale default 32-way, 2K mini-cache is used. The rightmost bar in Figure 9 plots the time (in hours) required to explore the design space using the exhaustive algorithm.

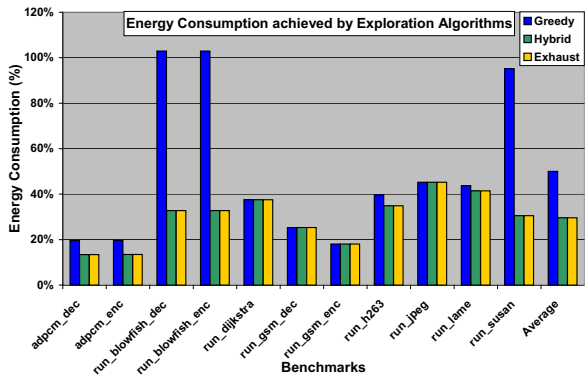


Fig. 8. Relative energy consumption achieved by exploration algorithms

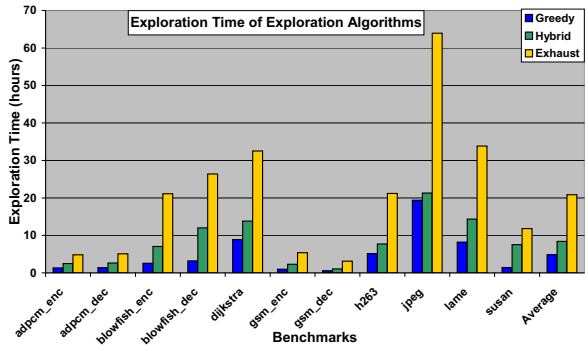


Fig. 9. Exploration time of exploration algorithms

As we can see, although greedy algorithm is able to discover extremely low-energy solutions, it may take 10s of hours to perform the exploration. To reduce the exploration time, we fist developed a greedy DSE heuristic.

**GreedyExploration()**
01: $size = MIN\_SIZE, assoc = MIN\_ASSOC$

// greedily find the size
02: **while** $(betterNewConf(size \times 2, assoc, size, assoc))$
03: $\quad size = size \times 2$
04: **endWhile**

// greedily find the assoc
05: **while** $(betterNewConf(size, assoc \times 2, size, assoc))$
06: $\quad assoc = assoc \times 2$
07: **endWhile**

08: **return** $estimateEnergy(size, assoc)$

**betterNewConf(size', assoc', size, assoc)**
01: **if** $(!existsCacheConfig(size', assoc'))$
02: $\quad$ **return** $false$
03: $energy = estimateEnergy(size, assoc)$
04: $energy' = estimateEnergy(size', assoc')$
05: **return** $(energy' < energy)$

Fig. 10. Greedy Exploration Algorithm

### A. Greedy Algorithm

In an attempt to reduce the runtime of the exhaustive algorithm, we developed a very simple **greedy exploration algorithm** to explore the mini-cache design space.

The greedy algorithm outlined in Figure 10 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 mini-cache parameters result in lower energy consumption than the old mini-cache parameters.

The middle bar in Figure 8 plots the energy consumption when the mini-cache configuration is chosen by the greedy algorithm, and the leftmost bar in Figure 9 plots the time that greedy exploration requires to explore the design space of the mini-cache. Although the greedy algorithm reduces the exploration time on an average by a factor of 5X, the energy consumption is on an average 2X more than what is achieved by optimal algorithm. Clearly, there is a trade-off between the exploration time and the energy reductions that can be obtained thereby.

### B. Hybrid Algorithm

To obtain best of both the worlds, we developed a hybrid algorithm that can achieve energy consumption very close to the optimal configuration, while consuming time closer to the greedy algorithm.

Figure 11 outlines the hybrid algorithm. The algorithm first greedily searches for the optimal mini-cache size (lines 02-04). Note however that it tries every alternate mini-cache size. Hybrid algorithm tries mini-caches sizes in exponents of 4, rather than 2 (line 03). Once it has found the optimal mini-cache size, then it explore exhaustively in the size-associativity neighborhood (lines 07-15) to find out a better size-associativity configuration.

The leftmost bar in Figure 8 plots the energy consumption when the mini-cache configuration is chosen by the hybrid algorithm, and the leftmost bar in Figure 9 plots the time that hybrid exploration requires to explore the design space of the mini-cache. Our hybrid algorithm is able to find the optimal

```
HybridExploration()
01: size = MIN_SIZE, assoc = MIN_ASSOC

// greedily find the size
02: while (betterNewConf(size × 4, assoc, size, assoc))
03:     size = size × 4
04: endWhile

// search in the neighbourhood
05: done = false
06: while (!done)
07:     if (betterNewConf(size × 2, assoc, size, assoc))
08:         size = size × 2
09:     else if (betterNewConf(size, assoc × 2, size, assoc))
10:         assoc = assoc × 2
11:     else if (betterNewConf(size ÷ 2, assoc, size, assoc))
12:         size = size ÷ 2
13:     else if (betterNewConf(size ÷ 2, assoc ÷ 2, size, assoc))
14:         size = size ÷ 2, assoc = assoc ÷ 2
15:     else if (betterNewConf(size ÷ 2, assoc × 2, size, assoc))
16:         size = size ÷ 2, assoc = assoc × 2
17:     else
18:         done = true
19: endWhile

08: return estimateEnergy(size, assoc)
```

Fig. 11. Hybrid Exploration Algorithm

mini-cache configuration in all of our benchmarks, while it takes about 3X less time than the optimal algorithm.

## VII. Summary

Horizontally Partitioned Caches (HPC) are an effective architectural mechanism to achieve energy reduction. However the energy reduction obtained using HPCs is very sensitive on the HPC design parameters. Furthermore the compiler has a significant impact on the energy reduction obtained using HPCs, therefore it is important to include compiler effects during exploration. In this paper, we present a Compiler-in-the-Loop (CIL) Design Space Exploration (DSE) framework to decide on optimal HPC parameters for an application.

Our experiments on an HP iPAQ h4300-like memory subsystem executing benchmarks from the MiBench suite show that our CIL DSE can find HPC design parameters that result in 80% reduction of the memory subsystem energy consumption, thus motivating for the need of DSE for HPCs. We also show that traditional SO DSE can find HPC configurations which result in 57% memory subsystem energy reduction, but CIL DSE can achieve 30% on top of that, demonstrating the need and usefulness of CIL DSE. Finally our hybrid heuristic is able to save on an average of 67% of exploration time as compared to the exhaustive exploration, while providing maximum possible energy savings on our set of benchmarks.

## References


[1] D. Burger and T. M. Austin. The simplescalar tool set, version 2.0. *SIGARCH Comput. Archit. News*, 25(3):13–25, 1997.

[2] K. L. et al. *H.263 test model simulation software.* Telenor R&D, 1995.

[3] K. Fan, N. Clark, M. Chu, K. V. Manjunath R. Ravindran, M. Smelyanskiy, and S. Mahlke. Systematic register bypass customization for application-specific processors. In *Proc. of ASSAP*, 2003.

[4] A. Gonzalez, C. Aliagas, and M. Valero. A data cache with multiple caching strategies tuned to different types of locality. In *ICS '95: Proceedings of the 9th international conference on Supercomputing*, pages 338–347, New York, NY, USA, 1995. ACM Press.

[5] 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 *IEEE Workshop in workload characterization*, 2001.

[6] Hewlett Packard, http://www..hp.com. *HP iPAQ h4000 Series - System Specifications*.

[7] Intel Corporation, http://www.intel.com/design/pca/ applicationsprocessors/manuals/278693.htm. *Intel PXA255 Processor: Developer's Manual*.

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

[9] Intel Corporation, http://www.intel.com/design/mobile/desguide/251012.htm. *LV/ULV Mobile Intel Pentium III Processor-M and LV/ULV Mobile Intel Celeron Processor (0.13u) /Intel 440MX Chipset: Platform Design Guide*, 2002.

[10] *IPC-D-317A Design Guidelines for Electronic Packaging Utilizing High-Speed Techniques*, 1995.

[11] T. L. Johnson and W. mei W. Hwu. Run-time adaptive cache hierarchy management via reference analysis. In *ISCA*, pages 315–326, 1997.

[12] H.-H. S. Lee and G. S. Tyson. Region-based caching: an energy-delay efficient memory architecture for embedded processors. In *CASES '00: Proceedings of the 2000 international conference on Compilers, architecture, and synthesis for embedded systems*, pages 120–127, New York, NY, USA, 2000. ACM Press.

[13] M. Mamidipaka and N. Dutt. eCACTI: An enhanced power estimation model for on-chip caches. In *Technical Report TR-04-28, CECS, UCI*, 2004.

[14] Micron Technology Inc., http://www.micron.com/products/drammobilesdram/. *MICRON Mobile SDRAM MT48V8M32LF Datasheet*, 2005.

[15] J. A. Rivers, E. S. Tam, G. S. Tyson, E. S. Davidson, and M. Farrens. Utilizing reuse information in data cache management. In *ICS '98: Proceedings of the 12th international conference on Supercomputing*, pages 449–456, New York, NY, USA, 1998. ACM Press.

[16] P. Shivakumar and N. Jouppi. Cacti 3.0: An integrated cache timing, power, and area model. In *WRL Technical Report 2001/2*, 2001.

[17] A. Shrivastava, N. Dutt, A. Nicolau, and E. Earlie. Pbexplore: A framework for compiler-in-the-loop exploration of partial bypassing in embedded processors. In *DATE '05: Proceedings of the conference on Design, Automation and Test in Europe*, pages 1264–1269, Washington, DC, USA, 2005. IEEE Computer Society.

[18] A. Shrivastava, I. Issenin, and N. Dutt. Compilation techniques for energy reduction in horizontally partitioned cache architectures. In *CASES '05: Proceedings of the 2005 international conference on Compilers, architectures and synthesis for embedded systems*, pages 90–96, New York, NY, USA, 2005. ACM Press.

[19] G. Tyson, M. Farrens, J. Matthews, and A. R. Pleszkun. A modified approach to data cache management. In *MICRO 28: Proceedings of the 28th annual international symposium on Microarchitecture*, pages 93–103, Los Alamitos, CA, USA, 1995. IEEE Computer Society Press.

[20] O. S. Unsal, I. Koren, C. M. Krishna, and C. A. Moritz. The minimax cache: An energy-efficient framework for media processors. In *HPCA '02: Proceedings of the Eighth International Symposium on High-Performance Computer Architecture (HPCA'02)*, page 131, Washington, DC, USA, 2002. IEEE Computer Society.

[21] R. Xu and Z. Li. Using cache mapping to improve memory performance of handheld devices. *Performance Analysis of Systems and Software, 2004 IEEE International Symposium on - ISPASS*, pages 106–114, 2004.