PARE: A Power-Aware Hardware Data Prefetching Engine

Yao Guo Mahmoud Ben Naser Csaba Andras Moritz

Department of Electrical and Computer Engineering
University of Massachusetts, Amherst, MA 01003
{yaoguo, mbennase, andras}@ecs.umass.edu

ABSTRACT

Aggressive hardware prefetching often significantly increases energy consumption in the memory system. Experiments show that a major fraction of prefetching related energy degradation is due to the hardware history table related energy costs. In this paper, we present PARE, a Power-Aware pRefetching Engine that uses a newly designed indexed hardware history table. Compared to the conventional single table design, the new prefetching table consumes 7-11X less power per access. With the help of compiler-based location-set analysis, we show that the proposed PARE design improves energy consumption by as much as 40% in the data memory systems in 70-nm BTPM processor designs.

Categories and Subject Descriptors: B.3.2 [MEMORY STRUCTURES]: Design Styles; C.3 [SPECIAL-PURPOSE AND APPLICATION-BASED SYSTEMS].

General Terms: Design, Experimentation, Performance.

Keywords: Data Prefetching, Prefetch Engine, Low Power, Energy Efficiency.

1. INTRODUCTION

Data prefetching is one of the successful techniques to bridge the speed gap between processor and memory system. Although considerable research [16, 3, 13, 14, 5, 11, 9, 10] has been focused on improving the performance of prefetching mechanisms, the impact of prefetching techniques on processor energy efficiency has not yet been fully investigated.

In our previous work [7, 6], we have found that while aggressive prefetching techniques often help to improve performance, they could increase memory system energy consumption very significantly.

Hardware prefetching requires the help of a history table to record recent memory access instructions and set up relationships between them in order to make prefetching decisions and calculate prefetching addresses. The history tables are usually pretty large (normally 64-128 entries) [3, 13]. When implemented as a fully-associative CAM table, the energy cost of each table lookup or update operation could cost the amount of energy which is comparable to a read operation of a low-power cache. To make accurate prefetching decisions, the history table is accessed very frequently to update the recent information on all relevant load instructions, which makes this part of the energy overhead very significant.

In this paper, we introduce PARE - a new Power-Aware pRefetching Engine with a novel design of an indexed hardware history table. PARE is focused on improving the power-efficiency of hardware-based data prefetching. We show the detailed design of PARE for one hardware prefetching technique and compare its power dissipation with the currently used fully-associative table design. We show that with the help of compile-time location-set analysis [15], we can divide the memory accesses into different relationship groups, with each group consisting of memory accesses visiting only closely related location-sets. The compiler generated group numbers allow us to use the indexed history table in PARE.

In the proposed prefetch history table, we divide the total entries into multiple (e.g., 16 or more) smaller tables. Each memory access will be directed to one of the tables upon entering the prefetching engine according to their group numbers provided by the compiler. The prefetching engine will update the information within the group and will make prefetching decisions solely based on the information within this group. The compile-time location-set analysis is utilized to ensure that no information will be lost due to the partitioning of memory accesses. We can reduce the power consumption of each access to the prefetching tables by 7-11X with the proposed technique based on our HSPICE simulation.

To estimate power consumption in the memory system, we use state-of-the-art low-power cache circuits and simulate them using HSPICE. The SimpleScalar [4] simulation tool has been modified to implement the hardware prefetching technique and collect statistics on performance as well as switching activity in the memory system. The compiler passes are implemented using the SUIF infrastructure [17]. Our experiments show that the proposed technique improves the energy consumption by as much as 40% in the data memory system for a set of general-purpose programs. Our evaluation is based on 70-nm BPTM technology node and accounts for both active and leakage power.

The rest of this paper is organized as follows. Section 2 describes the energy overhead of data prefetching. The PARE power-aware prefetching engine is presented in Section 3. Section 4 gives an overview of the location set based group analysis. Section 5 presents the experimental assumptions. Section 6 shows the simulation results and we conclude with Section 7.

The 2005 International Symposium on Low Power Electronics and Design (ISLPED'05), August 8–10, 2005, San Diego, California, USA

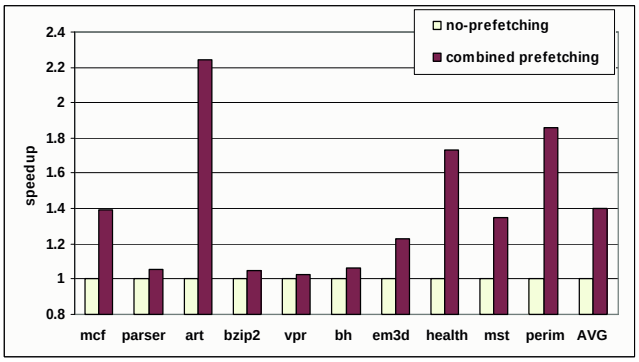

Figure 1: Performance speedup for different benchmarks.

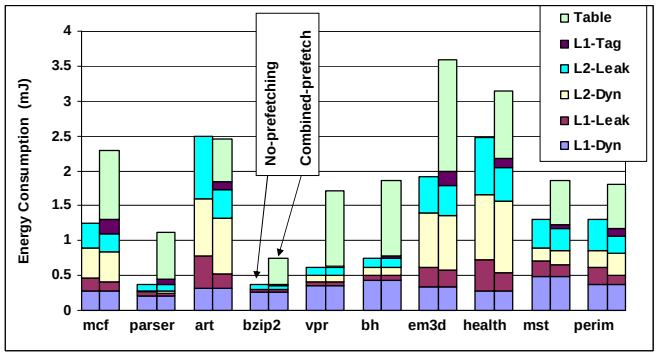

Figure 2: Total energy consumption for cache and prefetching history table.

2. MOTIVATION

In our previous studies [7, 6], we have evaluated the energy perspective of both hardware and software based data prefetching techniques. We have found that while aggressive prefetching techniques often help to improve performance, in most of the applications, they significantly increase the total energy consumption.

One of the techniques which produces the best performance speedup while resulting in the worst energy consumption is the combined stride and pointer prefetching technique [7]. The combined prefetching technique integrates the benefits from both stride prefetching [3] and dependence-based prefetching [13], such that it works on general-purpose programs that often use both array and pointer structures.

The performance improvement of the combined prefetching technique is shown in Figure 1. The first five benchmarks are from SPEC2000 benchmarks; the last five are Olden benchmarks which contain many pointer structures.

We can see from the figure the combined prefetching achieves a performance speedup averaging about 40% and it works consistently among all the benchmarks, including both array and pointer intensive programs.

However, hardware prefetching requires the help of history tables to record the recent memory access instructions and set up relationships between them in order to make prefetching decisions and calculate prefetching addresses. To achieve the prefetching benefit on both array-intensive and pointer-intensive programs, combined prefetching has to act aggressively, accessing the prefetching history table more frequently and issuing more prefetching requests. The naive design of the combined prefetching technique would use the hardware tables of both stride prefetching and dependence prefetching, simply by putting them together and accessing both of them. Instead, to make fair assumptions in a power-efficient system, we use an integrated 64-entry table as our baseline, which is shown later in Section 3. The integrated table already utilizes some of the low-power features as we will show later, but it is still energy hungry because of its large size and high associativity. As opposed to caches where banking is used to reduce power consumption, in prefetching all entries have to be searched at runtime.

We calculate the total energy consumption in the memory system for the prefetching technique based on HSPICE simulation. The results are shown in Figure 2: we show the energy breakdown (from bottom to top for each bar) for L1 dynamic energy, L1 leakage, L2 dynamic energy, L2 leakage, L1 tag lookups due to prefetching, and prefetch hardware table accesses.

The results in Figure 2 show that power consumption increases for the combined prefetching by more than 50% for many applications. This is mainly due to the prefetch table accesses and the extra L1 tag lookups due to prefetching. Note that a low power L1 data cache has a banked design that makes its CAM tag significantly more power efficient (e.g., 32-entry with 23 bits) during an access compared to the prefetching history table (64 entry by 32 bits as will be shown). The prefetch table accesses account for more than 70% of the total prefetching related energy overhead, and is the component that we are targeting to improve in this paper.

3. POWER-AWARE PREFETCHING DESIGN

As we mentioned earlier, the combined stride and pointer prefetching technique [7] integrates the mechanisms from both stride prefetching [3] and dependence-based prefetching [13].

Stride prefetching captures the static strides between memory accesses (mainly array accesses), and requires a history table to record the address of the instruction (PC), previously accessed address, and the predicted stride. In comparison, the dependence-based prefetching requires two history tables to record the potential candidates of instructions and the correlations which include PC, previously generated addresses, and predicted offset values.

As we will show later, the tables for both techniques could be combined together into a single table, each entry attached with one bit to indicate the prefetching type. We will also use two bits to indicate the prefetching status, which will help us to track whether the relationship is steady (status>1) or not. Prefetching requests will be issued only after the relationship is established, i.e., it is steady.

Next, we will show the design of our baseline prefetching history table, which is a 64-entry fully-associative table with many circuit-level low-power features. Following that we present the design of the proposed indexed history table for PARE, and compare the power dissipation, including both dynamic and leakage, of the two designs.

3.1 Baseline History Table Design

The baseline prefetching table design is a 64-entry fully-associative table shown in Figure 3. In each table entry,

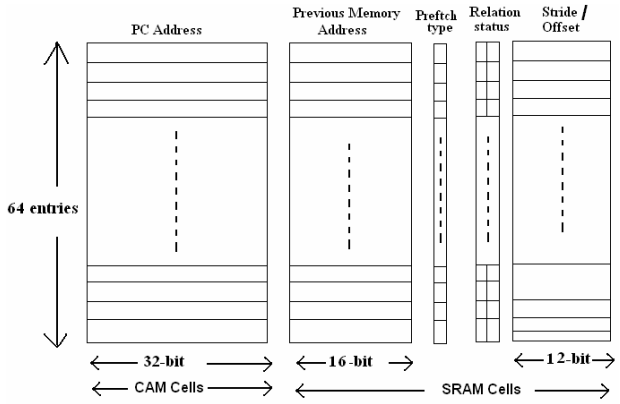

Figure 3: The baseline design of hardware prefetch table.

we store a 32-bit program counter (the address of the instruction), the lower 16 bits of the previously used memory address (we do not need to store the whole 32 bits because of the locality property in prefetching). We also use one bit to indicate the prefetching type and two bits for status, as mentioned previously. Finally, each entry also contains the lower 12 bits of the predicted stride/offset value.

In our design, we use Content Addressable Memory (CAM) for the PCs in the table, because CAM provides a fast and power-efficient data search function, accessing data by its content rather than its memory location.

The memory array of CAM cells logically consists of 64 by 32 bits. The rest of the history table is implemented using SRAM arrays. During a search operation, the reference data are driven to and compared in parallel with all locations in the CAM array. Depending on the matching tag, one of the wordlines in the SRAM array is selected and read out.

The prefetching engine will update the table for each load instruction and check whether steady prefetching relationships have been established. If there exists a steady relation, the prefetching address will be calculated according to the relation and data stored in the history table. A prefetching request will be issued in the following cycle.

3.2 PARE History Table Design

Each access to the table in Figure 3 still consumes significant power because all 64 CAM entries are activated during a search operation. We could reduce the power dissipation in two ways: reducing the size of each entry and partitioning the large table into multiple smaller tables.

First, because of the program locality property, we do not need the whole 32 bits PC to distinguish between different memory access instructions. If we use only the lower 16 bits of the PC, we could reduce roughly half of the power consumed by each CAM access.

Next, we break up the whole history table into 16 smaller tables, each containing only 4 entries, as shown in Figure 4. Each memory access will be directed to one of the smaller tables according to their group numbers provided by the compiler when they enter the prefetching engine . The prefetching engine will update the information within the group and will make prefetching decisions solely based on the information within this group. The compile-time location-set analysis is utilized to ensure that no information will be lost due to the partitioning of memory accesses. The group number can be accommodated in future ISAs that target energy efficiency and can be added easily in VLIW/EPIC type of designs. We also expect that many optimizations that would use compiler hints could be combined to reduce the impact on the ISA. The approach can reduce power significantly even with fewer tables (requiring fewer bits in the ISA) and could also be implemented in current ISAs by using some bits from the offset. Embedded ISAs like ARM that have 4 bits for predication in each instruction could trade off less predication bits (or none) with perhaps more bits used for compiler inserted hints. The compiler analysis will be presented in the next section.

In the PARE history table shown in Figure 4, during a search operation, only one of the 16 tables will be activated based on the group number provided by the compiler. We only perform the CAM search within the activated table, which is a fully-associative 4-entry CAM array.

The schematic of each small table is shown in Figure 5. Each small table consists of a 4x16 bits CAM array containing the program counter, a sense amplifier and a valid bit for each CAM row, and the SRAM array on the right which contains the data.

We use one of the most power-efficient CAM cell design proposed in [18]. The cell uses ten transistors that contain an SRAM cell and a dynamic XOR gate used for comparison. It separates search bitlines from the write bitlines in order to reduce the capacitance switched during a search operation.

For the row sense amplifier, we are using a single ended alpha latch to sense the match line during the search in the CAM array. The activation timing of the sense amplifier was determined with the case where only one bit in the word has a mismatch state.

Each word has the valid bit which indicates whether the data stored in the word will be used in search operations. A match line and a single ended sense amplifier are associated with each word. A hit/miss signal is also generated: its *high* state indicating a *hit* or *multiple hits* and the *low* state indicating *no hits* or *miss*.

Finally, the SRAM array is the memory block that holds the data. Low-power memory designs typically use a six-transistor (6T) SRAM cell. Writes are performed differentially with full rail voltage swings.

The power dissipation for each successful search is the power consumed in the decoder, CAM search and SRAM read. The power consumed in a CAM search includes the power in the match lines and search lines, the sense amplifiers and the valid bits.

The new hardware prefetch table has the following benefits compared to the baseline design:

- The dynamic power consumption is dramatically reduced because of the partitioning into 16 smaller tables;
- The CAM cell power is also reduced because we use only the lower 16 bits of the PC instead of the whole 32 bits;
- Another benefit of the new table is that since the table is very small (4-entry), we do not need a column sense amplifier. This also helps to reduce the total power consumed.

However, some overhead is also introduced by the new

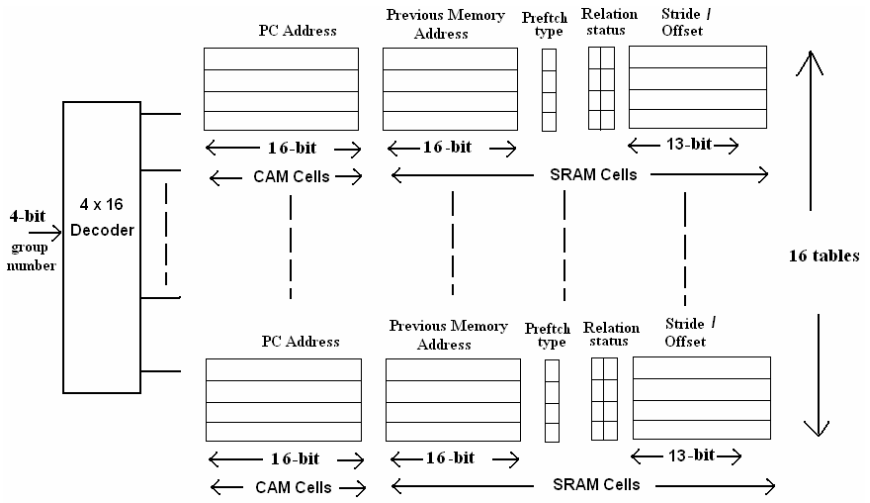

Figure 4: The overall organization of our hardware prefetch table.

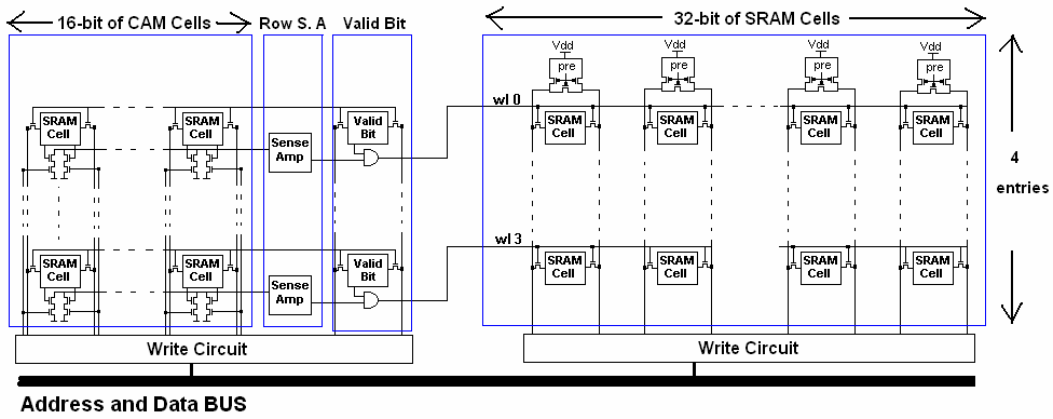

Figure 5: The schematic for each small history table.

design. First, we need an address decoder to select one of the 16 tables. The total leakage power is increased (in a relative sense only) because while one of the smaller tables is active, the remaining 15 tables will be leaking. Fortunately, as we will show next, the new PARE design overcomes all these disadvantages.

3.3 Power Evaluation

The hardware history table was designed using the 70-nm BPTM technology and simulated using HSPICE with a supply voltage of 1V. Both leakage and dynamic power are measured. Figure 6 summarizes our results showing the breakdown of dynamic and leakage power at different temperatures for both baseline and PARE history table designs.

From the figure, we can see that leakage power is very sensitive to temperature. The leakage power, which is initially 10% of the total power for the PARE design at room temperature (25℃), increases up to 50% as the temperature goes up to 100℃. This is because scaling and higher temperature cause subthreshold leakage currents to become a large component of the total power dissipation.

The new PARE table design proves to be much more power efficient than the baseline design. Although the leakage power consumption of PARE has more than doubled compared to the baseline design (this is because a smaller fraction of transistors are switching and a larger fraction are idle), the dynamic power of PARE is reduced dramatically, from 13mW to 1.05mW. As the result, the total power con-

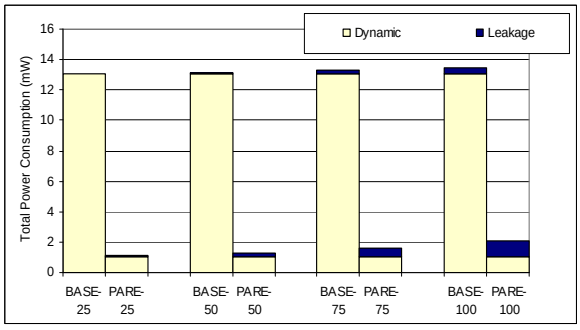

Figure 6: Power consumption for each history table access for PARE and baseline designs at different temperatures(℃).

sumption is reduced by 7-11X. For our simulation, we used the power consumption result at 75℃, which is the typical temperature of a chip.

4. LOCATION-SET BASED GROUP ANALYSIS

Due to space constraints, we cannot get into details of the compiler analysis algorithms. We instead give a brief overview of how it helps to partition the memory accesses into different groups such that we can use the new proposed PARE history table design.

Figure 7 shows the flow diagram of our compiler procedures. SUIF compiler infrastructure [17] is used to perform the analysis on intermediate files. Our location-set analysis pass is performed after the high-level SUIF passes.

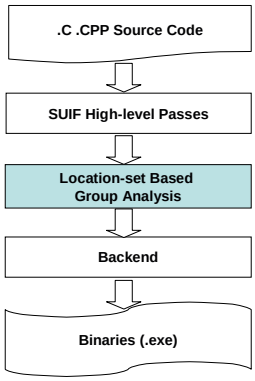

Figure 7: The flow diagram for the compiler passes.

Location-set analysis is a compiler analysis similar to pointer alias analysis [15]. By specifying locations for each memory objects allocated by the program, a location set is calculated for each memory instruction. A key difference in our work is that we use an approximative runtime-biased analysis [8] that has no restrictions in terms of complexity. Each location set contains the set of possible memory locations which could be accessed by the instruction.

The location-sets for all the memory accesses are grouped based on their relationships and their potential effects on the prefetching decision-making process. The reason why we can group the memory accesses while not losing the accuracy of prefetching is because of the regular properties of the prefetching techniques: stride prefetching is based on the relationship within an array structure, while dependence-based pointer prefetching is based on the relationship between linked data structures.

The results of the location-set analysis, along with type information captured during SUIF analysis, give us the ability to group the memory accesses which relate during the prefetching decision-making process into the same group. For example, memory instructions which access the same location-set will be put in the same group, while the instructions accessing the same pointer structure will also be put in the same group.

In our analysis, group numbers are assigned within each procedure, and will be reused on a round-robin basis if necessary. The group numbers then will be written as annotations to the instructions and transferred to the SimpleScalar simulator via the binaries.

5. EXPERIMENTAL ASSUMPTIONS

5.1 Experimental Framework

We implement the hardware prefetching techniques by modifying the SimpleScalar [4] simulator. We use SUIF [17] to implement the compiler passes for PARE, generating annotations of group information which we later transfer to assembly codes. The binaries input to the SimpleScalar simulator are created using a native Alpha assembler. The parameters we use for the simulations are listed in Table 1.

Table 1: Baseline parameters

Processor speed	1GHz
Issue	4-way, out-of-order
L1 D-cache	32KB, CAM-tag, 32-way, 32bytes cache line
L1 I-cache	32KB, 2-way, 32bytes cache line
L1 cache latency	1 cycle
L2 cache	unified, 256KB, 4-way, 64bytes cache line
L2 cache latency	12 cycle
Memory latency	100 cycles latency + 10 cycles/word

The benchmarks evaluated are listed in Table 2. The SPEC2000 benchmarks [1] use mostly array-based data structures, while the Olden benchmark suite [12] contains pointer-intensive programs that make substantial use of linked data structures. We randomly select a total of ten benchmark applications, five from SPEC2000 and five from Olden. For SPEC2000 benchmarks, we fast forward the first one billion instructions and then simulate the next 100 million instructions. The Olden benchmarks are simulated to completion except for one (perimeter), since they complete in relatively short time.

Table 2: SPEC2000 & Olden benchmarks

Benchmark	Description
SPEC2000	
181.mcf	Combinatorial Optimization
197.parser	Word Processing
179.art	Image Recognition / Neural Nets
256.bzip2	Compression
175.vpr	Versatile Place and Route
Olden	
bh	Barnes & Hut N-body Algorithm
em3d	Electromagnetic Wave Propagation
health	Colombian Health-Care Simulation
mst	Minimum Spanning Tree
perimeter	Perimeters of Regions in Images

5.2 Cache Energy Modeling

To accurately estimate power and energy consumption in the L1 and L2 caches, we perform circuit-level simulations using HSPICE. We base our design on a recently proposed low-power circuit [18] that we implement in 70-nm BPTM technology. Our L1 cache includes the following low-power features: low-swing bitlines, local word-line, CAM-based

Table 3: Cache configuration and power consumption

Parameter	L1	L2
size	32KB	256KB
tag array	CAM-based	RAM-based
associativity	32-way	4-way
bank size	1KB	4KB
# of banks	32	64
cache line	32B	64B
Power (mW)		
tag	6.5	6.3
read	9.5	100.5
write	10.3	118.6
leakage	3.1	23.0
reduced leakage	0.8	1.5

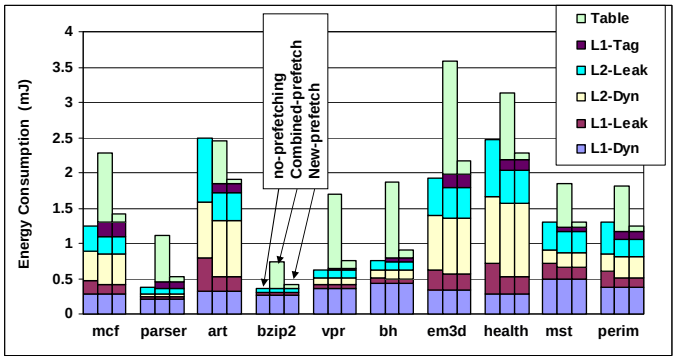

Figure 8: Energy consumption in the memory system after applying different energy-aware prefetching schemes.

tags, separate search lines, and a banked architecture. The L2 cache we evaluate is based on a banked RAM-tag design.

As we expect that implementations in 70-nm technology would have significant leakage, we apply a recently proposed circuit-level leakage reduction technique called asymmetric SRAM cells [2]. This is necessary because otherwise our conclusions would be skewed due to very high leakage power. The *speed enhanced cell* in [2] has been shown to reduce L1 data cache leakage by 3.8X for SPEC2000 benchmarks with no impact on performance. For L2 caches, we use the *leakage enhanced cell* which increases the read time by 5%, but can reduce leakage power by at least 6X. In our evaluation, we assume speed-enhanced cells for L1 and leakage enhanced cells for L2 data caches, by applying the different asymmetric cell techniques respectively.

The power consumption for our L1 and L2 caches are shown in Table 3.

6. RESULTS

We integrate PARE into the SimpleScalar simulator and calculate the energy consumption statistics based on the simulation.

Figure 8 shows the energy reduction with PARE applied. We can see that we could reduce more than 70% of the prefetching energy overhead in the data memory system for most of the applications. Overall, the energy consumption is improved by almost 40% in the data memory system compared to the baseline prefetching engine. Along with the leakage energy reduction due to the performance speedup of prefetching, we can see that the prefetching impact on energy is reduced to negligible or even improved for some applications with PARE applied.

7. CONCLUSION

This paper proposes a power-efficient prefetching engine called PARE to improve the power-efficiency of hardware-based data prefetching mechanisms. Our experiments show that the proposed techniques improve the data memory system energy consumption by as much as 40%. Although we have implemented PARE on one specific data prefetching technique, we believe that PARE could be applied on other hardware prefetching techniques as well.

8. REFERENCES

[1] The standard performance evaluation corporation, 2000. http://www.spec.org.

[2] N. Azizi, A. Moshovos, and F. N. Najm. Low-leakage asymmetric-cell sram. In *ISLPED'02*, pages 48–51.

[3] J. L. Baer and T. F. Chen. An effictive on-chip preloading scheme to reduce data access penalty. In *Supercomputing 1991*, pages 179–186, Nov. 1991.

[4] D. C. Burger and T. M. Austin. The simplescalar tool set, version 2.0. Technical Report CS-TR-1997-1342, University of Wisconsin, Madison, June 1997.

[5] R. Cooksey, S. Jourdan, and D. Grunwald. A stateless content-directed data prefetching mechanism. In *ASPLOS-X*, pages 279–290, 2002.

[6] Y. Guo, S. Chheda, I. Koren, C. M. Krishna, and C. A. Moritz. Energy-aware data prefetching for general-purpose programs. In *Power-Aware Computer Systems(PACS'04), Micro-37*, Dec. 2004.

[7] Y. Guo, S. Chheda, I. Koren, C. M. Krishna, and C. A. Moritz. Energy characterization of hardware-based data prefetching. In *Intl. Conf. on Computer Design (ICCD'04)*, pages 518–523, oct 2004.

[8] Y. Guo, S. Chheda, and C. A. Moritz. Runtime biased pointer reuse analysis and its application to energy efficiency. In *Power-Aware Computer Systems(PACS'03) at Micro-36*, Dec. 2003.

[9] M. H. Lipasti, W. J. Schmidt, S. R. Kunkel, and R. R. Roediger. Spaid: software prefetching in pointer- and call-intensive environments. In *Micro-28*, pages 231–236, Nov. 1995.

[10] C.-K. Luk and T. C. Mowry. Compiler-based prefetching for recursive data structures. In *ASPLOS-VII*, pages 222–233, Oct. 1996.

[11] T. Mowry. *Tolerating Latency Through Software Controlled Data Prefetching*. PhD thesis, Dept. of Computer Science, Stanford University, Mar. 1994.

[12] A. Rogers, M. C. Carlisle, J. H. Reppy, and L. J. Hendren. Supporting dynamic data structures on distributed-memory machines. *ACM TOPLAS*, 17(2):233–263, Mar. 1995.

[13] A. Roth, A. Moshovos, and G. S. Sohi. Dependence based prefetching for linked data structures. In *ASPLOS-IIV*, pages 115–126, oct 1998.

[14] A. Roth and G. S. Sohi. Effective jump-pointer prefetching for linked data structures. In *ISCA-26*, pages 111–121, 1999.

[15] R. Rugina and M. Rinard. Pointer analysis for multithreaded programs. In *PLDI'99*, pages 77–90.

[16] A. J. Smith. Sequential program prefetching in memory bierarchies. *IEEE Computer*, 11(12):7–21, Dec. 1978.

[17] R. Wilson and et. al. SUIF: A parallelizing and optimizing research compiler. Technical Report CSL-TR-94-620, Computer Systems Laboratory, Stanford University, May 1994.

[18] M. Zhang and K. Asanovic. Highly-associative caches for low-power processors. In *Kool Chips Workshop, Micro-33*, Dec. 2000.