Design and Implementation of Correlating Caches

Arindam Mallik, Matthew C. Wildrick, Gokhan Memik
Department of Electrical and Computer Engineering
Northwestern University
Evanston, IL 60208

ABSTRACT

We introduce a new cache architecture that can be used to increase performance and reduce energy consumption in Network Processors. This new architecture is based on the observation that there is a strong correlation between different memory accesses. In other words, if load X and load Y are two consecutively executed load instructions, the offset between the source addresses of these instructions remain usually constant between different iterations. We utilize this information by building a correlating cache architecture. *This architecture consists of a Dynamic Correlation Extractor, a Correlation History Table, and a Correlation Buffer. We first show simulation results investigating the frequency of correlating loads. Then, we evaluate our architecture using SimpleScalar/ARM. For a set of representative applications, the correlating cache architecture is able to reduce the average data access time by as much as 52.7% and 36.1% on average, while reducing the energy consumption of the caches by as much as 49.2% and 25.7% on average.*

CATEGORIES & SUBJECT DESCRIPTORS

B.3 Memory Structures; B.8 Performance and Reliability

GENERAL TERMS

Performance, Design

1. INTRODUCTION

Traditional processing elements in networks are either general-purpose processors or ASIC solutions. The limitations of these traditional approaches led to the design of Network Processors (*NPs*). By being tailored towards a specific application domain and utilizing application-specific optimizations, NPs achieve performance comparable to ASIC solutions.

In this paper, we introduce a cache architecture that reduces the energy consumption of the local caches as well as overall energy consumption. Our new architecture is proposed to replace the local data caches in the execution cores of NPs. Overall; the proposed techniques reduce the average data access times by improving the hit rates in level 1 data caches. In addition, the energy consumption of the local caches is also reduced.

In the heart of our technique lies the observation that the load instructions in most networking applications are correlated. First, during the execution of the applications, if a $load_X$ (meaning the load operation at PC X) is followed by a $load_Y$ (meaning the load operation at PC Y) once, the probability that $load_X$ is followed by $load_Y$ in the next iteration is very high. We call this relation the *precedence correlation*. Second, in such a precedence correlation, we observed that the source addresses are also correlated which is defined as *source correlation*. In other words, if $load_X$ accesses src_{X_1} in its first iteration and src_{X_2} in its second while $load_Y$ accesses src_{Y_1} and src_{Y_2} in the same order, there is a high probability that the following will hold:

$$| src_{X\_1} | - | src_{Y\_1} | = | src_{X\_2} | - | src_{Y\_2} |$$

In this paper, we design a cache architecture that takes advantage of this knowledge. Particularly, our contributions in this paper are:

1) We introduce load source correlation,
2) We show that there is a strong source correlation in a variety of networking applications,
3) We design a memory subsystem that utilizes the correlation information, and
4) We present simulation results indicating that our proposed architecture can be efficiently used in Network Processors to reduce the energy consumption and execution time.

Energy consumption has traditionally been one of the primary design criteria for mobile systems. Although NPs are initially designed for wired systems, there is an increasing motivation to utilize them in wireless systems. Table 1 presents characteristics of several commercial NPs.

The rest of the paper is organized as follows. In the next section, we discuss the definitions and present simulation results investigating the correlation between load operations. Section 3 presents the details of our correlating cache architecture. Section 4 discusses the simulation environment. In Section 5, we present the simulation results. Section 6 overviews the related work. We conclude the paper with a summary in Section 7.

Table 1. Representative Network Processor Products:

Product	Power [W]	Tech. [μ]
Agere PayloadPlus	12	0.18
AMMC (MMC) nP7510	4	0.18
Clearwater CNP810SP	12	0.15
IBM Rainier	20	0.18
Intel IXP2850	27.5	0.13
Intel IXP1200	5	0.28
Motorola C-5	20	0.18
PMC-Sierra RM9000	5	0.18

2. DEFINITIONS AND MOTIVATIONAL RESULTS

In the previous section, we have defined the two important concepts in our work: precedence correlation and source correlation. Figure 1 presents the execution of a loop to illustrate the correlation relations. Figure 1 (a) presents the code of the loop labeled start. Within the loop, there are two load operations (*ld*X and *ld*Y). Figure 1 (b) and (c) depict the source addresses accessed by the *ld*X and *ld*Y for two different scenarios. In (b), we see that the *ld*X and *ld*Y exhibit *precedence correlation*. This sets a high probability that after *ld*X is executed, *ld*Y will be executed as the next load instruction. Figure 1 (c), on the other hand, depicts a sequence of source address sequences, where there is strong source correlation, i.e. the offset from the address accessed by *ld*X and the address accessed by *ld*Y is the same for all the iterations of the

ISLPED'04, August 9–11, 2004, Newport Beach, California, USA.

loop (which is equal to 0x200. Similar to precedence correlation, we associate a probability with a *source correlation*.

Figure 2 and Figure 3 present simulation results investigating the correlation factors (precedence and source) in various networking applications. We simulate a processor similar to StrongARM SA-110. Note that the size and associativity of the caches do not affect the correlation factors. The applications and the simulation parameters for these simulations are identical to those discussed in Section 4. Figure 2 plots the results for simulations measuring the precedence correlation.

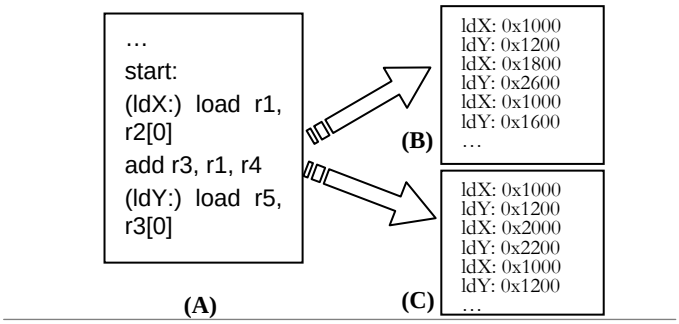

Figure 1. Correlation example: (A) is the code segment executed, (B) is a possible sequence of source addresses accessed by ldX and ldY, which exhibits precedence correlation, but no source correlation, (C) is another sequence of source addresses which exhibits source correlation between ldX and ldY.

For each application we measure the fraction of loads that has a certain probability of precedence correlation. Particularly, more than 95% of the loads exhibit an 80% or more precedence correlation. Figure 3 plots the results for source correlation. Similar to precedence correlation, the results indicate a strong source correlation. Particularly, 91% of the executed load instructions exhibit an 80% or more source correlation. Although not presented due to lack of space, the offset values between the accesses range from 4 bytes to 13848 bytes. The average offset observed in the simulated applications is 27.4 bytes.

3. CORRELATING CACHES

We designed the correlating cache architecture to take advantage of strong source correlation among load operations. The overview of the architecture is depicted in Figure 4. There are three additional structures compared to a traditional memory subsystem: the *Dynamic Correlation Extractor (DCE)*, *Correlation History Table (CHT)* and the *Correlation Buffer (CB)*. The DCE aims to detect the source correlation among different load instructions. The PC of the preceding load operation is stored in the CHT along with the corresponding offset. If the same load instruction is executed, the CHT captures the correlation and starts to prefetch. If the currently executed PC is not in the CHT, no prefetching is performed. Regardless of the prefetching, the current request only accesses the correlating buffer. If the accessed block does not exist in the CB, the DL1 cache is accessed and the block is promoted to the CB.

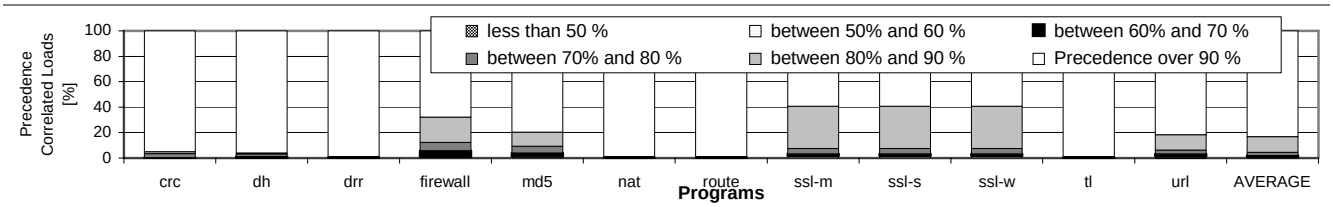

Figure 2. Fraction of precedence correlated load instructions.

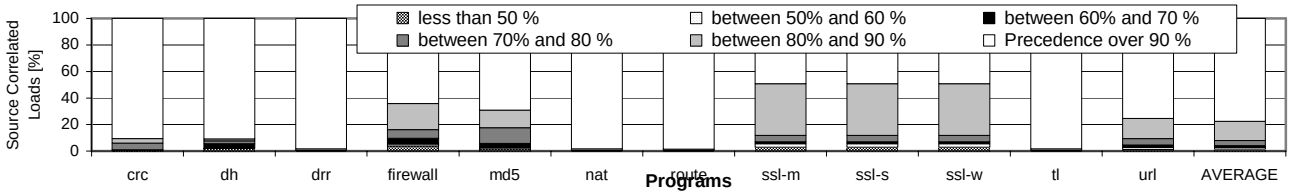

Figure 3. Fraction of source correlated loads for NetBench applications.

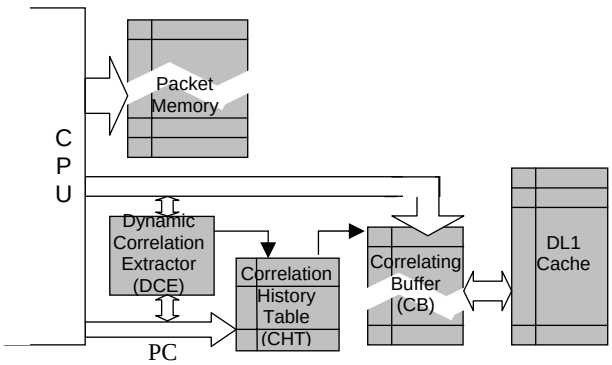

Figure 4. Correlating Cache Architecture.

3.1 DYNAMIC CORRELATION EXTRACTOR (DCE)

In the previous section, we have associated probabilities with each source correlation. The hardware requirement to implement it is too high. Instead, the *DCE* stores the last two offset values seen for the specific PC using the DCE table. The total data size of the DCE table is "# of entries in the DCE table * 2" bytes. DCE also uses two registers called *last source address (LSA)* and *last program counter (LPC)*. The overall procedure of the DCE is depicted in Figure 5. The DCE table is accessed using the LPC (last PC) as the address. If the DCE table returns a hit, the new offset is compared against the last_offset_1 and last_offset_2. If both comparisons indicate equality, the LPC value is entered into the CHT with the value "new offset". If the entry is not found in the DCE table, a new entry is generated. In either case, the LPC and LSA (last source adreess) values are updated once the remaining computations are completed.

```
new offset = current source address – LSA
access the DCE table using the LPC
  if hit
    if new offset == last_offset_1 == last_offset_2
```

```
      put the LPC into the CHT with offset value new offset
    else
      last_offset_2 = last_offset_1,  last_offset_1 = new offset
  else
    allocate a new entry for LPC with last_offset_1 = new offset
LPC = current PC
LSA = current source address
```

Figure 5. The procedure for the DCE.

3.2 *Correlation History Table*

The *Correlation History Table (CHT)* is a small cache structure used to store the source correlated loads captured by the DCE. For each memory access, the CHT is probed using the PC. If the PC is in the CHT, the corresponding offset value is read, and then added to the current source address. A prefetch signal to the Correlating Buffer (CB) is generated. If the accessed source address is not in the CHT, it is ignored. *Note that neither the DCE nor the CHT are in the critical path of the load access. Hence, their latency will not affect the overall performance.*

3.3 *Correlating Buffer*

The *Correlating Buffer (CB)* is used as the primary cache. It receives read/write requests from the CPU and prefetch signals from the CHT. For the read/write signals, the CB acts like a level 0 cache, i.e. if the block exists in the cache, the data is sent back to the CPU. If the block does not exist in the cache, DL1 cache is accessed to satisfy the request and the block is promoted to the CB.

3.4 *Discussion*

The energy consumption optimizations are due to the small energy consumption of the correlating cache structure. The correlating buffer is much smaller than the DL1 cache. Therefore, the energy consumption of the local cache is reduced. In addition to the reduction of energy consumption in the local caches, the number of accesses to the higher levels of memory subsystem (i.e. shared memory) is also reduced.

The only modification to the CPU is making the PC value available to the CHT and the DCE. The rest of the structures can be implemented by only modifying the DL1 cache.

4. Simulation Environment

We have performed several simulations to measure the effectiveness of the proposed techniques. The SimpleScalar/ARM simulator is used in the experiments. The base processor is modeled after the StrongARM SA-110 with in-order execution and an issue width of 2. The base processor has 8 KB, 2-way associative L1 data and instruction caches and a 128 KB, 4-way set-associative unified L2 cache. We used the CACTI 3.1 tool to find important characteristics of the caches. Then assuming a 1 GHz clock speed, the latency for L1 caches is set to 2 cycles, and the L2 cache latency is set to 12 cycles. We simulate the applications in the NetBench suite [1].

5. Experimental Results

We have performed two sets of simulations. In the first set, we investigate the effects of the DCE size and the CHT size on their performance. In the second set, we measure the energy and performance effects of our techniques.

5.1 *Analysis of Correlating Cache Structures*

Figure 6 plots the success of Dynamic Correlation Extractor (DCE) when its size is varied between 4 entries and 128 entries. Each data point in the figure corresponds to the average success of the particular configuration for 12 NetBench applications. The success of a DCE is measured in the number of extractions it makes divided by maximum possible extractions. Particularly, a 4-entry DCE captures 65% of the source correlated loads. A 32-entry DCE, on the other hand, captures 81% of the source correlated loads on average.

We have also performed a set of experiments to measure the success rate of the CHT. The results are summarized in Figure 7. In these simulations, the DCE size is set to 16-entries. To find the success rate of the CHT, we first set its size to "infinite" and measure the number of CHT hits. The success rate for any CHT size is the number of hits for the particular size divided by the number of hits to the CHT with infinite entries. We see that even a very small CHT is able to store most captured loads. Particularly, an 8-entry CHT is able to successfully capture 65% of the possible source correlations.

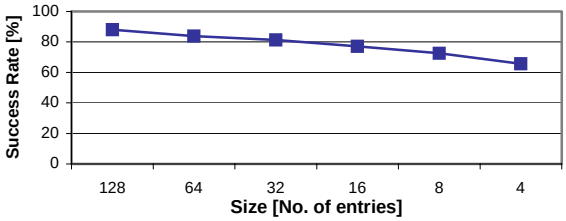

Figure 6. The success rate for different sizes of DCE.

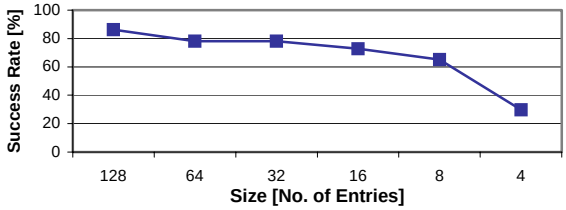

Figure 7. The success rate for different sizes of History Table.

5.2 *Correlating Cache Performance*

In this section, we present results for the energy and performance measurements of 6 cache configurations. The first configuration is the base configuration where the level 1 data cache (DL1) is directly accessed. The performances of the remaining techniques are presented relative to this base processor. Besides 4 different configurations of the correlating cache architecture, we also simulate a configuration called "without prefetching" technique (***WOP***). WOP has a small cache is placed between the CPU and the L1 data cache.

In all of these techniques, the large CB (LCB) is a 32-entry direct-mapped cache. We also perform simulations with a small CB that has 8-entries. When applicable, CHT and DCE have 16 entries and are direct-mapped. Using the CACTI 3.1 [2] tool, we found the properties of different CB sizes. In all cases, the CB can be accessed within a single cycle. When a CB is used (for WOP and all correlating cache configurations), the L1 data cache accesses are completed in 3 cycles.

The results for the energy consumption are presented in Figure 8. For each application, Figure 8 reports the reduction in the energy consumption of different caches compared to the base configuration. On average, we see that the correlating cache with large CB and small level 1(LCB x SDL1) data cache (2K direct-mapped) results in the least energy consumed - the energy consumed by the caches in the base processor is reduced by 25.9% on average. On the other hand, the correlating cache using the large CB (32-entry direct-mapped) combined with the LDL1 reduces the energy consumption by 25.7%, and WOP reduces the energy consumption by 22.2% on average. In general, we see that the LCB

x SDL1 configuration has one of the best energy performances. However, for the ssl-m application, it increases the total energy consumption by 27%. The reason for this is the large data size accessed by this application. The main reason for the reduction in the energy consumption with respect to the base processor is that most accesses are satisfied by the smaller CB (SCB) structure. In addition, the correlation architecture has a better energy efficiency compared to the WOP because more accesses are satisfied by the CB. In fact, the average miss rate for the CB in WOP is 32.6%. Correlation prefetching reduces this rate to 17.1%. The number of level 1 data cache accesses is not reduced in WOP. DL1 is accessed after a miss, in correlating cache it is accessed for prefetch. In fact, the number of DL1 accesses is slightly increased in the correlating cache architecture due to unused prefetch (on average 4%). Nevertheless, the energy consumed for CB misses is significantly reduced because of the reduction in the number of misses. Overall, the energy consumed by the CB is reduced by 13.6% for the correlating cache architecture compared to the WOP. Another factor in calculating the energy consumption of the correlating cache is the energy consumed for prefetch signals. On average, 35% of the CHT generated prefetch signals results in an actual prefetch operation. The remainder of the requests hit in the CB. These hits increase the energy consumed by the tag array. However, since the tag array consumes less energy than the data array, the overall penalty remains low. Comparing the energy consumed by all the correlating cache structures, the total extra energy consumed for the LCB x LDL1 configuration for prefetch is 11.4% of the total energy consumed by the CB.

Figure 9 presents the performance implications of the simulated architectures. Since the execution cores in Network Processors do not have the exact ARM processor architecture we are simulating, we present the average data access latency instead of execution cycles. Average data access latency is the average number of cycles for the memory subsystem to satisfy a request. We see that the LCB x LDL1 configuration has the best performance. It reduces the average access latency of the base processor by 36.1% on average. Overall, assuming that 15% of the processor energy is consumed for data accesses, the LCB x LDL1 configuration reduces the energy-delay product of the simulated architecture by 14.5%.

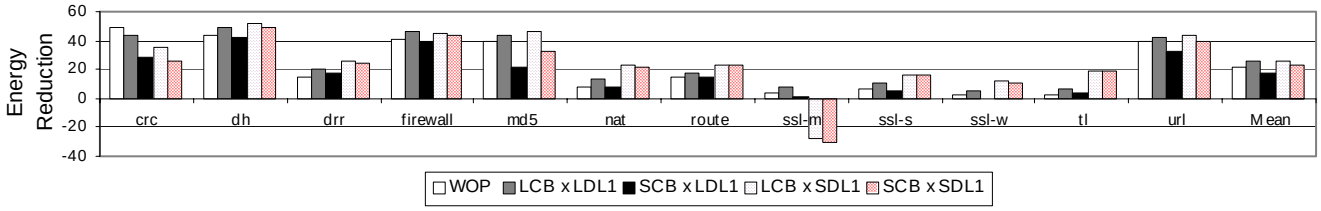

Figure 8. The energy reduction for the simulated techniques relative to the base processor: WOP (CB w/o any prefetching), NBP (CB with next-block prefetching), LCB x LDL1 (large CB and large DL1 combination), SCB x LDL1 (small CB and large DL1 combination), LCB x SDL1 (large CB and small DL1 combination), SCB x SDL1 (small CB and small DL1 combination). Small CB is 8-entry direct-mapped, large CB is 32-entry direct-mapped, large DL1 is 256 entry 2-way associative (8 KB), and small DL1 is 64 entry direct-mapped.

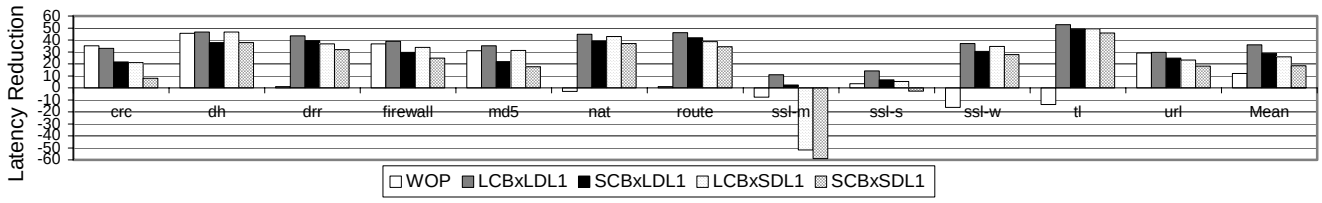

Figure 9. The reduction in average data access latency. The labels are identical to those of Figure 8.

6. Related Work

McKee et al. [3] proposed a special stream buffer unit (SBU) to store the stream accesses. Benitez and Davidson [4] presented a compiler framework to detect streaming data. Our proposed architecture does not require any compiler support for its tasks. In addition, earlier techniques are not applicable to networking applications as the displacement of accesses in most networking applications is not fixed. New techniques have been proposed to reduce the power consumption of high-performance processors [5,6] which concentrates on restructuring the cache. Our correlating buffer resembles the filter cache[5]. However, it improves the performance of the processor instead of degrading it.

7. Summary and Conclusions

In this paper, we introduced a correlating cache architecture that reduces the energy consumption of the local caches as well as overall energy consumption. In the heart of the architecture lies the observation that source addresses accessed by consecutive load operations usually exhibit a constant offset. For a set of representative applications, this architecture is able to reduce the average data access time by as much as 52.7% and 36.1% on average, while reducing the energy consumption of the caches by as much as 49.2% and 25.7% on average.

References

1. Memik, G., W.H. Mangione-Smith, and W. Hu. *NetBench: A Benchmarking Suite for Network Processors*. in *International Conference on Computer-Aided Design (ICCAD)*. Nov. 2001. San Jose / CA.

2. Wilton, S. and N. Jouppi, *An enhanced access and cycle time model for on-chip caches*. July 1995, Digital Western Research Laboratory, 93/5.

3. McKee, S.A., et al., *Smarter Memory: Improving Bandwith for Streamed References*, in *IEEE Computer*. July 1998. p. 54-63.

4. Benitez, M.E. and J.W. Davidson. *Code Generation for Streaming: An Access/Execute Mechanism.* in *Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 1991. Los Alamitos / CA.

5. Kin, J., M. Gupta, and W.H. Mangione-Smith. *The Filter Cache: an energy efficient memory structure*. in *Intl. Symposium on Microarchitecture*. Dec. 1997. Research Triangle Park / NC.

6. Powell, M.D., et al. *Gated-Vdd: A circuit technique to reduce leakage in cache memories.* in *Intl. Symposium on Low Power Electronics and Design*. July 2000.