Exploring Cache Optimization for Bioinformatics Applications

Shannon Dybvig, Megan Bailey, and Timothy Urness
Department of Mathematics and Computer Science
Drake University
2507 University Avenue
Des Moines, IA 50311
shannon.dybvig@drake.edu
megan.bailey@drake.edu
timothy.urness@drake.edu

Abstract

This paper explores the benefits of splitting cache and creating an optimal split in cache for bioinformatics applications. Split-cache architectures are most commonly divided into data-cache and instruction-cache. The split between instruction cache and data cache is usually optimized for average computer usage, although some processors are optimized for specific types of programs. Previous split-cache research with non-bioinformatics systems has allowed for optimization of cache by program type, but no optimization standard exists for bioinformatics software. Bioinformatics is currently one of the fastest-growing fields within the computer science realm because of its long-term medical and genetic implications. Given the amount of processing and analyzing that bioinformatics applications entail, efficiency is absolutely critical. This inspired us to explore the effects that different split-cache architectures have on the efficiency of bioinformatics software. We ran the benchmarks with several different configurations of primary cache and compared the results.

1. Introduction

Bioinformatics algorithms entail analyzing large amounts of data. Data can come in the form of deriving migration paths of animals, modeling evolution, or attempting to determine protein structure and behavior. The most common and important uses of bioinformatics deal with genetic material. When analyzing a sequence of DNA, billions of calculations or searches will sometimes need to be done. These algorithms can include looking for a specific sequence, attempting to identify one mutation out of thousands of base pairs, or attempting to identify an entire gene expression area. With the current setup, this takes hours, if not days for some analyses. Given the medical importance of this data, a more efficient system needs to be developed.

The largest obstacle to efficient bioinformatics processing is the "memory wall" problem. While new processors can handle such a large amount of data with efficiency and ease, accessing memory itself is still a slow process. To improve the situation, cache architecture needs to be updated. While much is being done with cache research, little is being focused on bioinformatics specifically. Although most modern processors have their primary cache divided for data and instruction cache, none are specifically tailored to the needs of bioinformatics applications. We wanted to find the optimal split between data and instruction cache for bioinformatics.

In order to establish what cache split is best, we used sim-cache, a cache architecture simulator from the SimpleScalar [1]. SimpleScalar creates a virtual replica of the hardware we wish to test. Through this program, it is possible to control sizes of cache, how many levels exist, if each level is split, and what associativity is used. We used ClustalW, a benchmark from the BioBench bioinformatics benchmark suite, to experiment with a small assortment of cache divisions.

2. Related Work

Some research on split-cache architecture involves dynamically allocating memory between the two parts of the cache and splitting cache based on data type. Splitting cache based on data type (that is, array versus scalar, as they have separate associatively needs) is vastly more efficient than leaving data cache unified. However, without an optimal split between instruction cache and data cache, the increased efficiency of data type splitting is less significant. In general, dynamic allocation is less efficient because of time wasted assessing which part of the cache needs more space. The majority of advanced processors split the L1 Cache; some also employ a split L2 Cache.

One of the problems with cache design is that there is no obvious way to improve it. Increased size allows for more data to be held, yet can greatly increase miss penalties [2]. More complex theories have algorithms that attempt to "read ahead" and choose what

data will be reused, but this method can greatly increase the cost of hardware. It is also not easy to implement and can have large miss penalty increases [3].

Another proposed structure is the use of a victim cache. When grouped with direct mapping cache, the victim serves as a means to keep useful data around a little longer in a reserve. After having two chances to be reused, it is then disposed of [4]. Another proposed cache design for set-associative cache takes advantage of the most recently used (MRU) data. All tags are read at the same time from MRU, but the data from MRU is instantly sent out [5].

3. Experiment

The three split configurations we used were a data cache (DL1) 128 blocks of size 64 with an instruction cache (IL1) of 256 blocks of size 64, a DL1 of 512 blocks of size 32 with an IL1 of 128 blocks of size 64, and a DL1 of 256 blocks of size 64 with an IL1 of 128 blocks of size 64. Each of these three splits were tested with 1-way, 2-way, and 4-way associativity. Secondary cache was unified. In all cache, the least recently used (LRU) method of clearing data was used.

After installing SimpleScalar and configuring gcc to compile in SimpleScalar's assembly language, ClustalW was compiled and put through some test-runs. We established that running full-length simulations of the benchmark was impractical and opted to limit the number of instructions performed each time the benchmark was run. Once we finished a run testing each of the nine possible configurations, we wrote a bash script to automate 720 more runs of the program (leaving us with results for 80 simulations of each configuration). At the end of the experiment we wrote a Perl script to parse the results and average the elapsed time of each simulation as well as the hits, misses, and miss rates of both the IL1 and DL1.

4. Results

The time the simulations took tended to decrease as we increased the associativity. However, the average quickest simulation was when the DL1 was configured to have 256 blocks with 2-way associativity, with 512-blocks with 4-way associativity just barely slower.

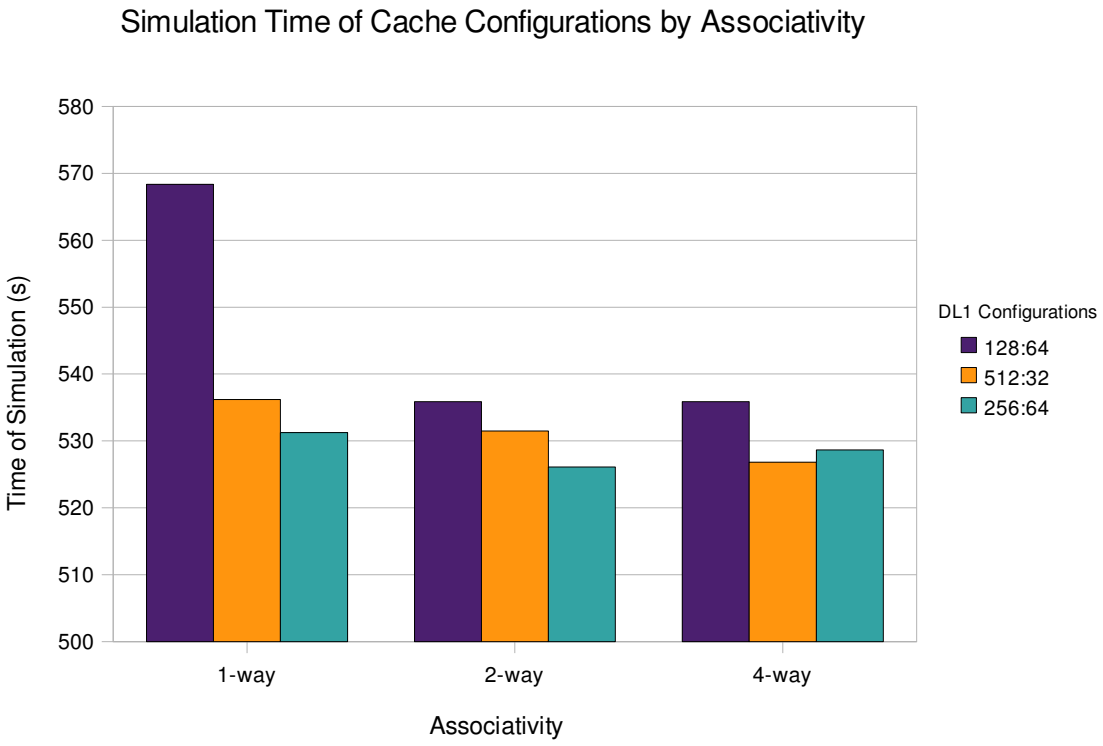

Figure 1: The time a simulation took changed based on associativity, but the configuration of the split caused the associativity to have greater or lesser effect.

Perhaps more interesting is the number of misses (in both DL1 and IL1) by configuration. One-way associative caches had substantially more misses, so we threw those data points out. Configuring DL1 to have 128 blocks of size 64 also had considerably more misses than the other configurations, so we removed it as well. Although it was only the third-fastest simulation on average, DL1 256:64 with 4-way associativity very clearly has the fewest number of misses, suggesting that in a non-simulated environment it would be the optimal configuration.

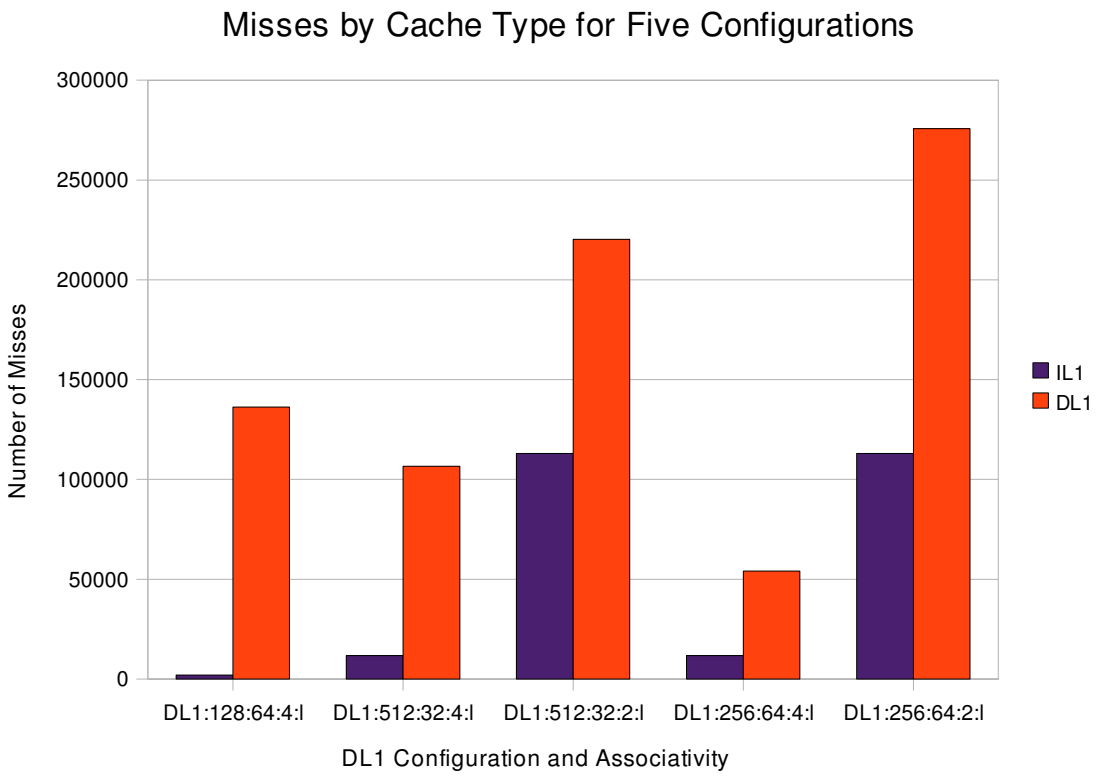

Figure 2: A lower instruction cache miss rate is going to have a more substantial effect on the efficiency of a program as it can carry out other instructions while waiting on data, while data cannot do anything while waiting for new instructions.

5. Conclusion

Based on the simulations of DL1:256:64 with IL1:128:64 with 4-way associativity, it seems that the best configuration is to have the data cache be twice as big as the instruction cache and have a high-degree associativity. However, there were some relatively constricting parameters on this set of simulations. In future research it would be of interest to explore different sizes of primary cache to see if it changes which configuration is optimal. Another issue to address is whether this split is best for all bioninformatics applications, or if this configuration is specific to ClustalW, and if a fully-associative cache would render different results (although associativity is less of a concern). Eventually, we would like to explore setting up a split in the data cache

between scalar and array data. Much of bioinformatics information is scalar data, though DNA structures can also be stored as array type. Once we have established an optimal split between instruction and data cache we intend to look into splitting the data cache, as scalar and array data have different associativity needs.

References

[1] SimpleScalar/PISA. http://www.simplescalar.com/

[2] Alan Jay Smith. Cache Memories. *Computing Surveys*. Vol. 14, No. 3, September 1982.

[3] Yong Chen, Surendra Byna, and Xian-He Sun. *Data Access History Cache and Associated Data Prefetching Mechanisms.* ACM 2007.

[4] Afrin Naz, Mehran Rezaei, Krishna Kavi, and Philip Sweany. *Improving Data Cache Performanc with Integrated Use of Split Caches, Victim Cache, and Steam Buffers.* The University of Northern Texas,

[5] L.K. John, T. Li, A. Subramanian. Annex Cache: A Cache Assist to Implement Selective Caching. *Microprocessors and Microsystems.* Vol. 23, 537-551, 1999.