Compiler–Directed Cache Line Size Adaptivity *

Dan Nicolaescu[1], Xiaomei Ji[1], Alexander Veidenbaum[1], Alexandru Nicolau[1], and Rajesh Gupta[1]

Department of Information and Computer Science
444 Computer Science, Building 302
University of California Irvine
Irvine, CA 92697–3425
{dann,xji,alexv,nicolau,rgupta}@ics.uci.edu

Abstract. The performance of a computer system is highly dependent on the performance of the cache memory system. The traditional cache memory system has an organization with a line size that is fixed at design time. Miss rates for different applications can be improved if the line size could be adjusted dynamically at run time. We propose a system where the compiler can set the cache line size for different portions of the program and we show that the miss rate is greatly reduced as a result of this dynamic resizing.

1 Introduction

The area available for on–chip caches is limited and the size and associativity of a cache for a given processor cannot be significantly increased without causing an increase in the cycle time. Currently in a given technology implementation processor designers decide the size of cache lines by considering different tradeoffs between speed and latency. But this tradeoff also influences the miss rate of the cache system. We show that this decision has a great impact on the miss rate of the memory system.

We propose a simple system that allows the cache line size to vary at run time. To achieve this we augment the ISA with a single extra instruction that sets the line size. A compiler can insert this instruction in the code at points it determines suitable by either static code analysis or profile–directed feedback.

While the hardware modifications are modest, the following questions need to be answered to determine the feasibility of the approach:

1. When should the cache line size be changed,
2. How often is it necessary to reconfigure,
3. What is the optimal reconfiguration policy?

On one hand it would not be feasible to change the cache line size every few instructions as the overhead associated with such reconfiguration would make

* This work was supported in part by the DARPA ITO under Grant DABT63-98-C-0045.

the approach prohibitively expensive. On the other hand if we reconfigure too infrequently, e.g. once per function call, we might miss some optimization opportunities because a function may contain a number of loops, each of them with a distinct cache behavior.

It has been shown that the majority of dynamic instructions in a program are executed in innermost loops. An inner loop is also likely to have reasonably stable spatial/temporal locality characteristics. This suggests that an inner loop may be a good place to change the cache line size and maintain the setting for the duration of such a loop. In this paper we study the performance of changing the cache line size at loop level and show that such an approach is feasible.

We currently use a profile–based mechanism for the control of adaptation by the compiler. Future work will study the opportunity to use compile–time analysis for making adaptivity decisions.

2 System Organization

The system being studied consists of a 3–level memory hierarchy. The cache line size is reconfigurable at run time using a special instruction. The set of sizes is: 8, 16, 32, 64, 128 and 256 bytes. A fully associative write buffer is also used.

The L1 cache is direct mapped and the hit latency is assumed to be 1 cycle. The L1 bus transfer takes 2 cycles. L2 is a 2–way set–associative with the access latency of 15 cycles. The main memory access latency is 100 cycles.

3 Experimental Infrastructure

3.1 Simulator

The framework provided by the ABSS [2] simulation system is used in this study. ABSS is a simulator that runs on SUN Sparc systems and is derived from the MINT simulator [3].

The ABSS simulator consists of 5 parts: augmentor, thread management, cycle-counting libraries, user-defined simulator of the memory system and the application program.

The augmentor program (called *doctor*) parses the original application assembly code, and adds instrumentation code that sends information about the loads and stores executed by the program to the simulator.

Our custom memory architecture simulator simulates a 3–level memory hierarchy, the cache line for the L1 cache is changeable at run time via commands embedded in the simulated program.

3.2 Compilation

We have used version 2.95 of the GCC compiler collection to conduct all the experiments. The compiler back–end was modified to emit special code sequences before entering a loop, or on the code path for exiting a loop. Given that the

compiler back–end is common to the C and Fortran77 compiler we were able to use this instrumentation for compiling all the SPEC95 benchmarks.

The code sequences were used for adjusting the cache line size, and for collecting statistics and identifying the loop (source file name and line number), and signaling to the cache simulator that a loop is being entered or exited.

All the benchmarks where compiled using the -O2 optimization flag, the target instruction set was SPARC V8plus.

4 Experiments

We have run the simulations for a memory system using different cache line sizes.

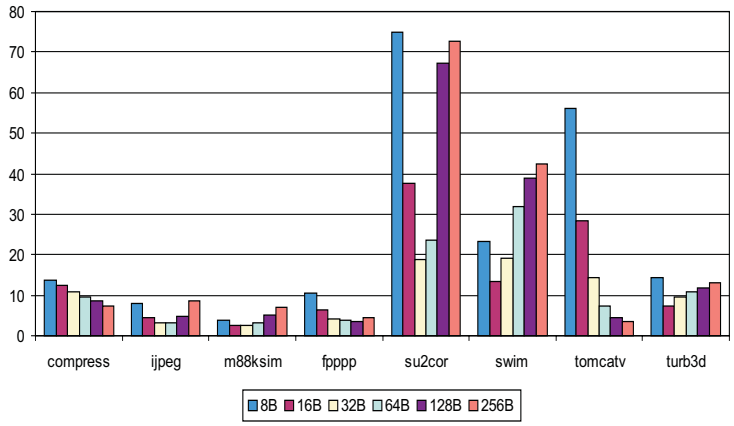

Fig. 1. L1 miss rate for the loop containing the most memory references.

The results shown in Figure 1 are miss rate for the loop that contains the most memory references for each benchmark is shown. It can be observed that no individual cache line size gets the minimum miss rate for all benchmarks, and that there is no rule for all benchmarks that could determine the optimal cache line size.

Figure 2 shows that even for the same benchmark, different loops have better miss rates for different cache line sizes. Based on these two fact we can conjecture that adapting the line size on loop boundaries improves the miss rate.

We used profiling to determine the best cache line size for each loop, we run the benchmarks using the training input set, determined for each loop what is the cache line size that generates the minimum miss rate and used that data to run the benchmarks using a compiler generated instruction to change the cache line size to the one that was determined to generate the minimum number of misses. This approach is practical since we have determined that the data

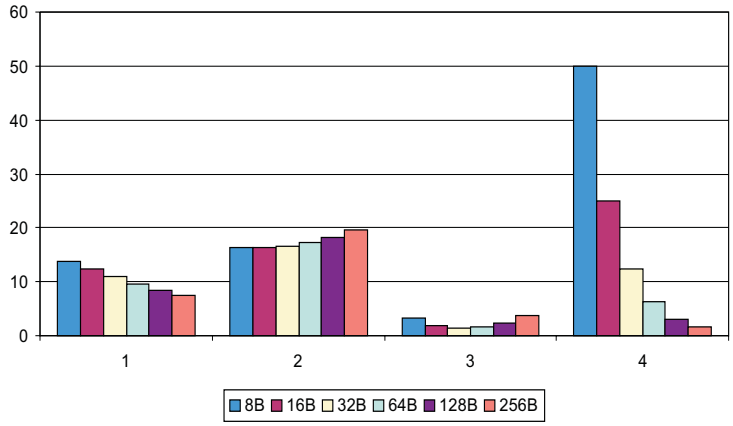

Fig. 2. L1 miss rate for the loops that generate most memory references for the compress benchmark

obtained using the small, manageable training input sets extrapolates well to the actual application. The results are shown in Figure 3. It can be seen that the miss rate always improves, sometimes by a wide margin.

Another interesting observation can be made from Figure 3. The "worst case" data is obtained by using the profile data for setting the line size in such way to maximize the miss rate. It can be observed that this worst case is in all cases very close to the miss rate for at least one of fixed line sizes. So it is likely that for about any fixed line size there will exist an application that will have a very high miss rate, therefore cache line size adaptability is a worthwhile feature for a general purpose processor that has to run well a variety of applications.

5 Conclusions and Future Work

We have shown that adapting the cache line size on a per loop basis improves the cache miss rate. We have used a profile base approach, future work will determine the cache line size at compile time using analytical approaches and we are also working on using hardware based approaches to dynamically change the cache line size.

References

[1] Teresa L. Johnson and Wen mei Hwu. Run-time adaptive cache hierarchy management via reference analysis. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, 1997.

[2] D. Sunada, D. Glasco, and M. Flynn. ABSS v2.0: SPARC simulator. Technical Report CSL-TR-98-755, Stanford University, 1998.

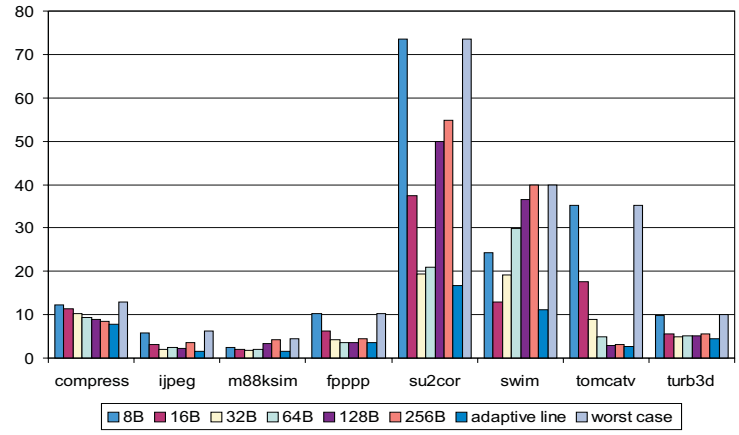

Fig. 3. L1 miss rate for different cache line sizes, and for an adaptive cache

[3] Jack E. Veenstra and Robert J. Fowler. Mint: A front end for efficient simulation of shared-memory multiprocessors. In *Intl. Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 201–207, 1994.

[4] Alexander V. Veidenbaum, Weiyu Tang, Rajesh Gupta, Alexandru Nicolau, and Xiaomei Ji. Adapting cache line size to application behavior. In *Proceedings ICS'99*, June 1999.

[5] Peter Van Vleet, Eric Anderson, Lindsay Brown, Jean-Loup Baer, and Anna Karlin. Pursuing the performance potential of dynamic cache line sizes. In *Proceedings of 1999 International Conference on Computer Design*, November 1999.