Caching in on Sisal: Cache Performance of Sisal vs. Fortran

P. L. Nico

Department of Computer Science
University of California, Davis
Davis, CA 95616
nico@cs.ucdavis.edu

A. Park

Department of Computer Science
University of California, Davis
Davis, CA 95616
park@cs.ucdavis.edu

Abstract

In this paper we investigate the relative cache performance of Sisal *and* Fortran *through trace-driven simulation of representative scientific applications. The range of cache configurations considered corresponds to on-chip caches in current and next-generation microprocessors. We find that in unified caches the performance is equivalent. With split instruction and data caches, performance is still comparable, yet the two languages demonstrate somewhat different tendencies.*

1 Background

With gains in CPU speed far outstripping gains in memory access times, memory performance is rapidly becoming the limiting factor in computer performance. Nearly all current architectures attempt to alleviate this bottleneck by placing small fast cache memories near the processor to take advantage of locality in memory references.[6] Since the difference in memory access time between a cache hit and a cache miss can be in the tens of cycles on a uniprocessor and in the tens to hundreds of cycles on a multiprocessor, cache behavior is of vital interest to those seeking high-performance from their computer systems.

At the speed at which current generation microprocessors run, even the time required to go off-chip has become substantial relative to the clock speed. As a result of this and of advancing VLSI technology, most current-generation microprocessors and virtually all next-generation microprocessors will have on-chip memory caches. Since chip real-estate is a valuable commodity, these caches are relatively small(*e.g.* 8–32Kbytes for the MIPS R4000[3]).

The ready availability of fast, inexpensive microprocessors will make them the building blocks of the next generation of parallel computers[5]. This means, of course, that regardless of how they are interconnected, microprocessor performance in general and memory performance in particular will be of increasing importance in the future.

While FORTRAN is still the workhorse language of the scientific community, the SISAL programming language[4] offers an attractive alternative since it provides the programmer with well-defined, determinate functions while providing the compiler with data dependency information from which it can extract implicit parallelism. Although SISAL's performance competitiveness with FORTRAN has been established on supercomputers such as the CRAY which has no cache[1], questions have been raised as to the relative performance in the cache environment of microprocessor based systems.

We explore some aspects of the relative behavior of FORTRAN and SISAL in the cache environment of current and next-generation microprocessors through simulating their behavior in a variety of caches.

2 The Model

In this study we examine the relative behavior of representative scientific applications in both SISAL and FORTRAN in cache environments similar to those of current and next generation RISC microprocessors.

To properly model potential on-chip cache configurations, we simulate both unified caches and split instruction and data caches. Sizes range from 2Kbytes to 64Kbytes for the instruction caches and 2K to 256K for the data caches. These ranges, though too small for external caches, far exceed the limits of current on-chip cache technology.

Since, by its very nature, a cache must be fast, and an on-chip cache must also be small and simple, most on-chip caches are, and will probably remain, direct

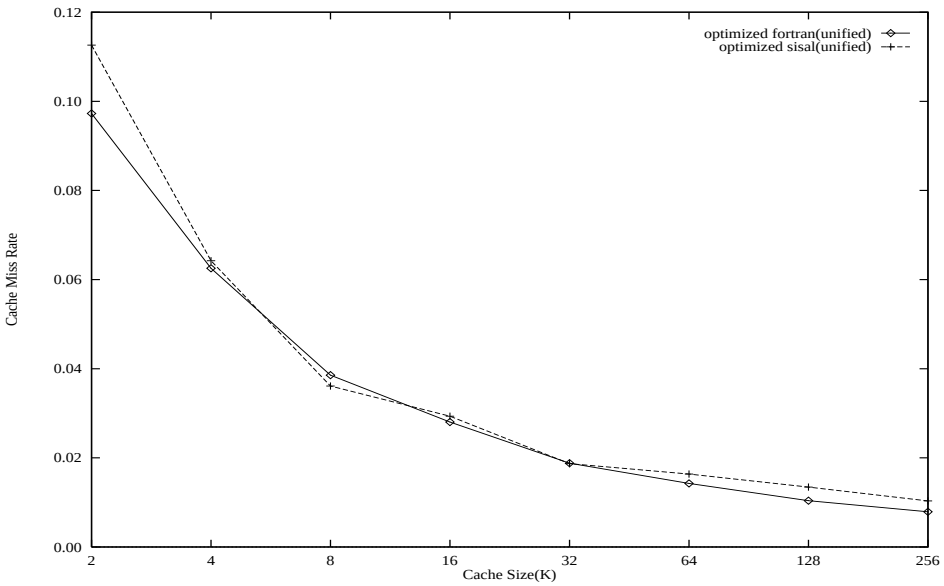

Figure 1: Average miss rate for unified Instruction/Data Caches

mapped.[2] For these reasons, we will concentrate on direct-mapped caches.

2.1 The Programs

The applications used here were taken from those used by David Cann for the performance comparison of SISAL and FORTAN on the CRAY Y-MP [1]. Missing are FFT and the Lawrence Livermore Loops. The FFT program was simply a call to the CRAY library which was clearly not portable for our purposes, and the Livermore Loops did not produce execution traces of sufficient length to properly exercise the caches.

The other applications and their subject areas can be seen in Table 2.1. We chose these applications as a reasonably representative scientific workload with which to exercise the compilers and caches. No attempt was made to re-tune them for the memory architecture of the microprocessor, and no special optimization instructions were given to the compilers.

Application	Subject Material
AMR	Hydrodynamics
BMK11A	Particle Transport
KIN16	Gel Electrophoresis
RICARD	Gel Chromatography
SIMPLE	Hydrodynamics
WEATHER	Weather Modeling

Table 1: The Applications

2.2 The Simulation

The cache performance data were generated by tracing the execution of each program and simulating the behavior of different cache configurations for that sequence of references. The reference streams were captured using the pixie profiling system on a Silicon Graphics MIPS R3000 based computer. All of the execution traces are from sequential execution.

The SISAL and FORTRAN programs were compiled using OSC version 12.9 and the MIPS distribution f77 compiler respectively. In each case, the standard "-O" level of optimization was used. The pixie profiler modifies the executable image so that, while running, it outputs its memory reference stream on an unused file descriptor. After being instrumented, each program was run and its reference stream fed into our cache simulator and analyzed.

Each reference stream was made up of 10 million references after skipping the initial 2 million references to account for transient start-up behavior. Since the largest caches simulated are only 256K, this is sufficiently long to demonstrate steady-state behavior.

The results of these simulations are presented below.

3 Results

The results presented here are those for direct mapped caches with line sizes of 16 bytes. Though

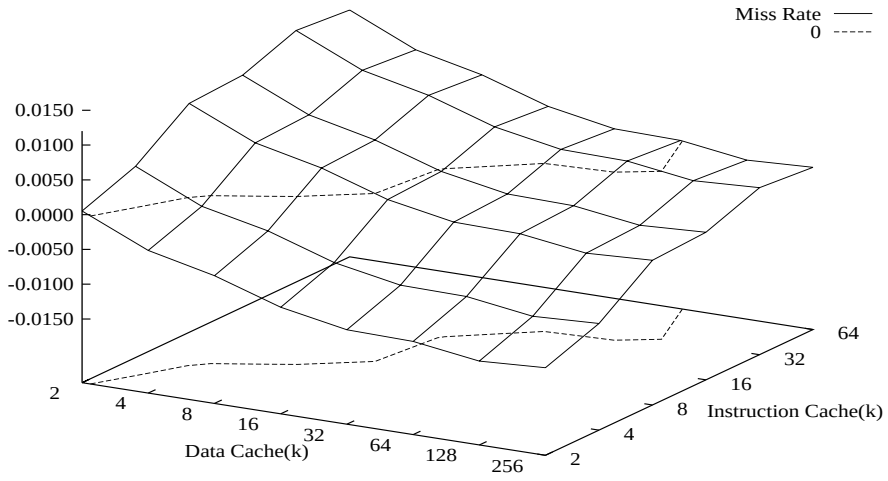

Figure 2: Average overall miss rate difference for split caches (FORTRAN−SISAL)

caches with different associativity and line sizes were simulated and gave different values, the relative behavior remained consistent.

First, Figure 1 gives the average miss rates for unified caches. These results were disappointing not because they were bad, but because nothing interesting happened. As can be seen from the figure, the SISAL and FORTRAN miss rates track each other closely with no clear advantage. Over the sampled space, the average difference is well less that 1%. Still, this shows the two languages to be competitive in the unified cache.

The case where the instruction and data caches are split is somewhat more intersting. In these figures, Fig. 2 through Fig. 4, we show the difference in the miss rates found by subtracting the SISAL value from the FORTRAN value rather than the miss rates themselves. . We find that not only does this reduce the clutter of the graphs, but it also shows what we are really looking for, the *relative* performance. Where the difference is above zero, SISAL is performing better, while when below zero, FORTRAN is better. The contour line shows the crossover point.

Figure 2 shows the overall miss rate differences for the whole range of caches averaged over the six applications. Here the differences begin to develop. For small instruction caches, FORTRAN consistently outperforms SISAL except when there is only 2k of data cache. When the instruction cache size in increased, however, SISAL comes into its own. For a current microprocessor, instruction and data caches of 8k are not unreasonable to consider. This puts it just barely above zero (0.2%). Note that in all cases the difference is not very great. The greatest differences at the top and bottom corners of the graph are 1.1% and −1.2% respectively.

Figures 3 and 4 decompose this graph into instruction and data miss rates respectively. Now it becomes evident why Figure 2 has the form it does. It can be seen from Figure 3 that SISAL requires a greater amount of instruction cache than FORTRAN before its miss rates come down. SISAL only catches up to FORTRAN when the instruction cache reaches 32k.

In the data cache, this tendency is reversed. Here, in Figure 4, it appears that SISAL makes more efficient use of the data cache until the data cache becomes as large as 128k and FORTRAN catches up. In each case, the two level off together when most of the working set is resident in the cache.

4 Analysis

Though we have not determined the cause of these differences, we have examined some possibilities.

Since we see that SISAL and FORTRAN behave so differently in their instruction and data references we considered the possibility that the two could have very different memory demands, as opposed to simply better or worse locality. To see the memory characteristics of the applications, each was run and the instruction and data references counted. Since some of these applications can run for a very long time, they

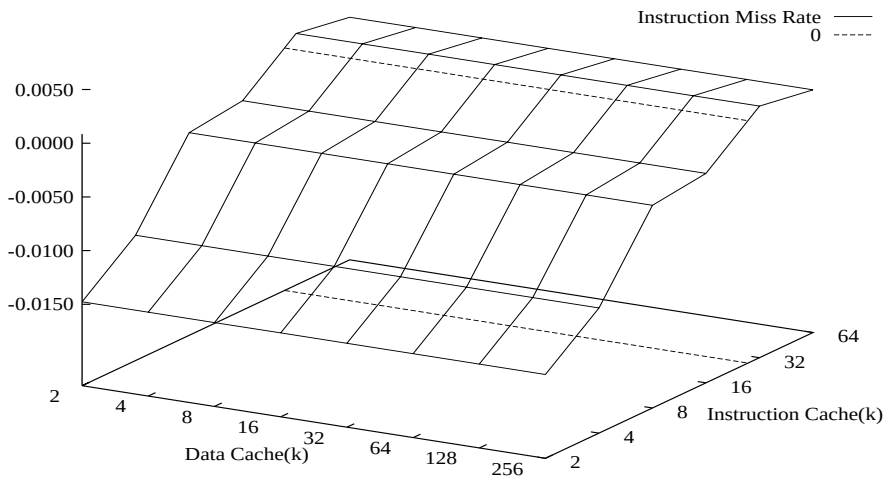

Figure 3: Average Instruction miss rate difference for split caches (FORTRAN−SISAL)

were limited to 10 minutes of processor time. Table 4 shows the memory characteristics for each application. There is variability among the applications, but no clear tendency emerges. It appears that the effect above is indeed based on locality rather than demand.

It is also reasonable to consider whether the behavior above is an artifact of the direct mapped caches with 16 byte lines. Since the compilers did not have any information about the cache architecture, it is possible that one could lay out the instructions or data in such a way that this particular configuration was disadvantageous. In order to counter this possibility we also ran simulations with line sizes of 4 bytes so there could be no prefetching effects, as well as two-way associative caches to limit destructive interference. In each case, the magnitude of the difference between the two languages changed, but the characteristic shape remained the same.

5 Conclusions and Future Work

In this paper, we have attempted to characterize the relative cache performance of FORTRAN and SISAL in environment of current and next-generation microprocessors. We have seen that in all cases simulated the performances have been comparable. Neither language dominates the other.

We have also seen that each language does have its areas of strength. SISAL behaves better in the data cache while requiring more instruction cache. FORTRAN behaves in a complementary fashion, running in a smaller instruction cache, but requiring more data space. This information could be useful either in selecting a language for a known cache configuration or in allocating the precious amount of cache real-estate if the application language is known at the time the cache is configured.

There are still many unanswered questions. We have only simulated one version of each of the compilers on one architecture and have only begun to investigate why the two languages behave as they do. In the future we would like to see how they behave on a wider variety of platforms and to further explain their behavior.

References

[1] D. Cann. Retire fortran? a debate rekindled. *Communications of the ACM*, 35(8):81–90, August 1992.

[2] M. D. Hill. A case for direct-mapped caches. *IEEE Computer*, 21(12):25–40, December 1988.

[3] G. Kane and J. Heinrich. *MIPS RISC Architecture*. Prentice-Hall, Englewood Cliffs, New Jersey, 1992.

[4] J. McGraw, S. Skedzielewski, S. Allan, R. Oldehoeft, J. Glauert, C. Kirkheim, W. Noyce, and

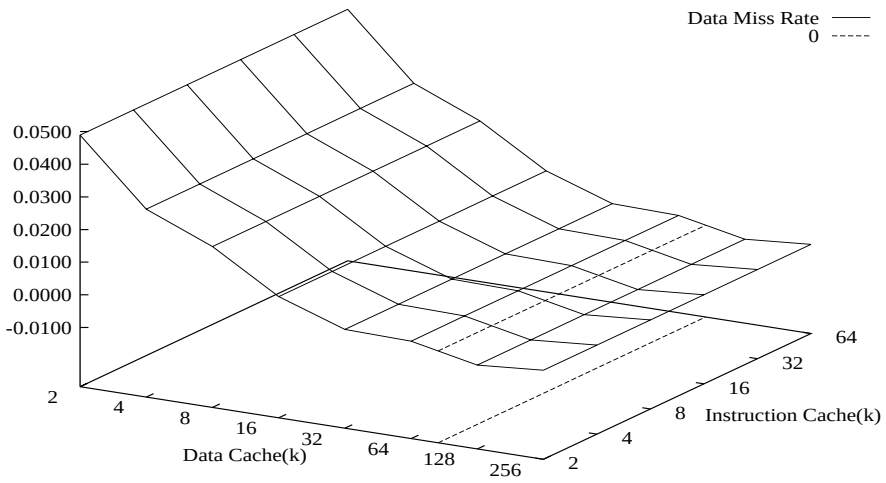

Figure 4: Average Data miss rate difference for split caches (FORTRAN−SISAL)

Program		Fetches	Loads	Stores	mem/inst
amr	SISAL	1,043,831,565	268,213,741	111,393,127	0.363
	FORTRAN	1,136,562,760	256,871,897	80,011,158	0.296
bmk11a	SISAL	854,104,288	212,210,754	127,706,750	0.398
	FORTRAN	941,454,917	244,412,172	141,005,308	0.409
kin16	SISAL	1,309,306,131	480,184,102	69,513,119	0.420
	FORTRAN	965,836,314	408,995,150	52,259,190	0.478
ricard	SISAL	805,401,526	298,707,707	143,946,230	0.550
	FORTRAN	803,445,241	238,072,848	147,489,178	0.480
simple	SISAL	844,093,836	272,734,276	90,036,300	0.430
	FORTRAN	867,464,552	278,596,699	51,846,183	0.381
weather[a]	SISAL	385,490,039	99,840,172	38,605,406	0.359
	FORTRAN	391,735,035	92,457,819	42,783,829	0.345

[a] Ran to completion

Table 2: Memory characteristics of programs (10 minutes running time)

R. Thomas. SISAL: *Streams and iteration in a single assignment language: Reference manual version 1.2.* Lawrence Livermore National Laboratory, Livermore, CA, manual M-146, rev. 1 edition, March 1985.

[5] J. Rattner. Supercomputing's destiny: the microprocessor. *Supercomputing Review*, June 1989.

[6] A. J. Smith. Cache memories. *Computing Surveys*, 14(3):473–530, September 1982.