The effect of adding a scalar D-cache to the Cray-4 vector processor

Steven J. Beaty
Cray Computer Corporation
1110 Bayfield Drive
Colorado Springs, Colorado, 80906
Voice: (719) 540-4129
FAX: (719) 540-4028
beaty@craycos.com
http://www.craycos.com/~beaty

Gearold R. Johnson
National Technological University
700 Centre Avenue
Fort Collins, Colorado 80526
Voice: (303) 495-6404
FAX: (303) 498-0601
gerryj@mail.ntu.edu

Abstract

In the past, vector supercomputers achieved high performance with long arithmetic pipelines coupled with fast scalar processors. Processor speed has increased at a rate greater than memory speed. Indeed, current vector processors have cycle times far faster than the memories they are connected to. When compilers can predict memory access patterns, they vectorize computations and thereby hide the processor/memory disparity. When memory access patterns are not known until run-time, caches can pay large dividends. This paper studies the effects of adding a scalar data cache to a modern vector processor and shows some encouraging results.

1 Introduction

As computer design has matured, processor speed has increased at a rate greater than that of general-purpose memory. This disparity is quite obvious as modern vector processors now have instruction cycle times of 1 nanosecond – substantially shorter than main memory access times. This disparity has led to memory banking, which helps vector accesses, and more levels in the memory hierarchy [HP90] including various levels of instruction and data caches. These caches attempt to bridge the speeds of the processor and memories by relying on the temporal and spatial locality of memory references.

The Cray vector processors have always had a cache for the instruction stream. They have not had any type of data cache as the vast majority of computation is done in vector mode. To support these computations the bandwidth between main memory and the processors is very large. The main memory is designed such that it can supply the vector registers with enough data so results are generated every clock cycle. So long as addresses are calculated rapidly and the vector length is large enough, vectors are efficiently fetched and operated upon.

This work is motivated when it was noticed that some vector operations were being paced by scalar operations. The increasing disparity between the processor and memory speeds exacerbates this situation. Modern compilers cannot always know a priori where in memory a scalar datum will reside. Therefore, it might not be able to have the datum in a fast memory, such as a register, when it is needed for reading or writing. Scalar operands are used for three important areas in computations:

- address calculations,
- control variables such as loop variables, and
- CPU I/O operations.

Therefore, it is important that the speed of the scalar section of a vector processor be matched to the speed of the vector section.

2 Evolution of Cray Memory Hierarchy

Over the years, there have been a number of different organizations of the memory hierarchy in Cray supercomputers. These different organizations have been responses to what were then the state of the art in memory speed and utilization. Within these different schemes, somethings have remained constant. For example, all machines have had eight ‘A’ and eight ‘S’ registers. The ‘A’ registers are so named to denote that addressing expressions are usually (but not necessarily) computed in them. Special address addition and multiplication units are directly attached to this bank. Similarly, the ‘S’ registers are used to compute other scalar expressions and have shifting and logical units directly attached to them. The ‘A’ registers have been sized to hold operands that can address all of common memory (24 bits for the Cray-1, 32 for the Cray-2 and Cray-3, and 35 bits in the Cray-4) while the ‘S’ registers have been sized to hold full-word operands (64 bits in all Cray machines.) The functional units attached to the two different kinds of registers are non-orthogonal. For example, there are no shift units directly attached to the ‘A’ bank, and no

Machine	Clock	LM	CM
Cray-1	80	n.a.	22
Cray-2	250	8	55
Cray-3	500	8	35
Cray-4	1000	n.a.	49

Figure 1: Cray Memory Comparison

integer multiply unit directly attached to the 'S' bank. This is not to say that these operations cannot be performed on the respective operands as one can move the operands to a suitable location and then perform the operation. All Cray machines have also had a bank of vector, 'V', registers. There are eight vector registers, each holding 64 operands. These are used for the majority of the computation that occurs on Crays. There are logical, shift, integer arithmetic, and floating point arithmetic functional units attached to the 'V' bank.

One aspect that has varied in these machines is the presence of the 'T' ("temporary") register bank. This was included in the original Cray-1, and consisted of 64 1-word (64-bit) registers. It was omitted for the Cray-2 and Cray-3. The Cray-4 reintroduces it. It is used by the compilers to hold a variety of temporary data.

Another aspect that has varied in direct relation to the presence of the 'T' registers is the presence of 'local memory.' The Cray-2 and Cray-3 machines had a memory that was small (16KW) but fast to access, 'nearer' to the processor than the large common memory. This memory is managed by the compiler and used for often-used variables and process-specific information. This provides the compilers with an intermediate storage area, farther than registers but closer than common memory. See Figure 1 for a comparison of the memory hierarchies; the clock speed in in megahertz and the memory speed is in machine cycles.

As should be obvious from this evolution, there is no one optimal memory hierarchy for Crays or anyone elses general-purpose machine. This is because one cannot know a priori what type of jobs will be run on the processor. One tries to find a good mixture of available faster and slower memories so that most jobs fit within the constraints of the hierarchy.

Figure 2 shows some of the current state-of-the-art processors and their cache schemes. For example, the Mips R8000 has a split-cache scheme where integer references are directed to an 16KB on-chip cache and floating-point references are sent to a off-chip cache whose size is up to 16MB. The NEC SX-3 is a vector processor that has a cache that contains only references to scalar data. The single-chip implementations have between 16KB and 32KB in total cache; the multi-chip designs have more. Most of the newer designs also have provisions for some type of second-level cache. In [KSF+94], the addition of a data cache for all memory references, including vector references, was studied. Part of the thrust of that work was to study using lower-cost and more-dense DRAMs, instead of faster SRAMs, and adding a cache to speed the access and increase the bandwidth of the memory.

3 Method

A little background on the approach and methods used by this work is in order. We were able to do very precise studies as a number of excellent software tools were available to us. These included a simulator, a highly-optimizing compiler, and a program suite that is thought to be exemplary of the type of programs users wish to run on these types of computers.

For this study, the performance of the Cray-4 had to be simulated as there were not enough actual machines running at the time this study was undertaken. However, there is an internal software simulator that exactly matches the operation of the hardware on a cycle-by-cycle basis. This simulator is known to exactly match a logic simulator's result and so was judged to be completely satisfactory for this work. It calculates all of the timing found on the actual machine including all the possible memory contentions. The software simulator was run on Cray-2's and Cray-3's for this study.

The compiler used to generate code was Cray Computer Corporation's optimizing FORTRAN compiler. It is a state-of-the-art vectorizing compiler. It does all the traditional optimizations [ASU86] along with extensive machine-specific ones. One of the major aims of the compiler is to vectorize all computations so that they are performed using the pipelined vector units. This tends to remove scalar references from the program.

For this study, we used the 24 Livermore Loops [McM86]. These are often used to quote performance figures in the supercomputer industry. It is important to note that the timing figures given include all the overhead of the master program for the loops. This includes a number of calls to I/O routines and other setup procedures. Most of the actual loops themselves are completely vectorized by the compiler and contain no references to scalar memory values.

We varied a number of the parameters for the D-cache. One 64-bit word was the block size. Cache sizes of 1KW and 4KW were examined. We varied the associativity of the cache between direct-mapped and 8-way.

Company	Processor	D-cache Size	I-cache Size	D-cache Mapping
Digital	21064 [Sit92]	8KB	8KB	direct
Motorola	601 [Die94]	32KB combined		8-way
Motorola	603 [Die94]	8KB	8KB	2-way
Mips	R4000 [MWV92]	16KB	16KB	direct
Mips	R8000 [Hsu94]	16KB	16KB	direct
NEC	SX-3 [SWL+92]	64KB scalar-only		

Figure 2: Current D-cache Schemes

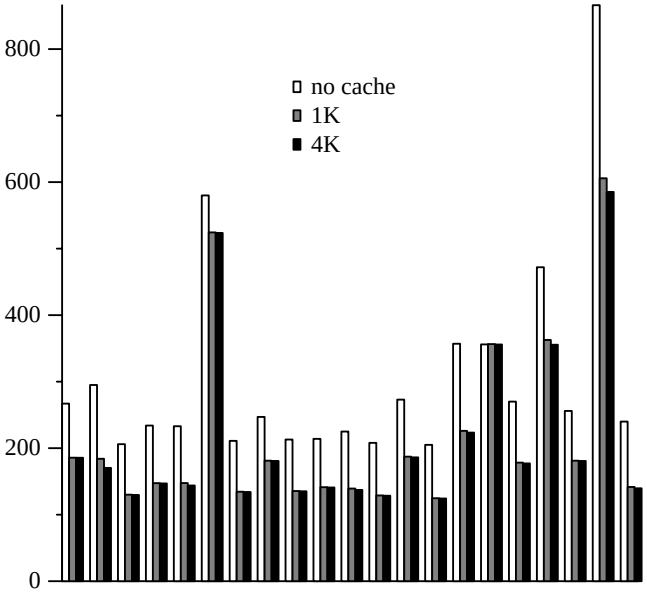

Figure 3: Comparison of a direct mapped D-cache

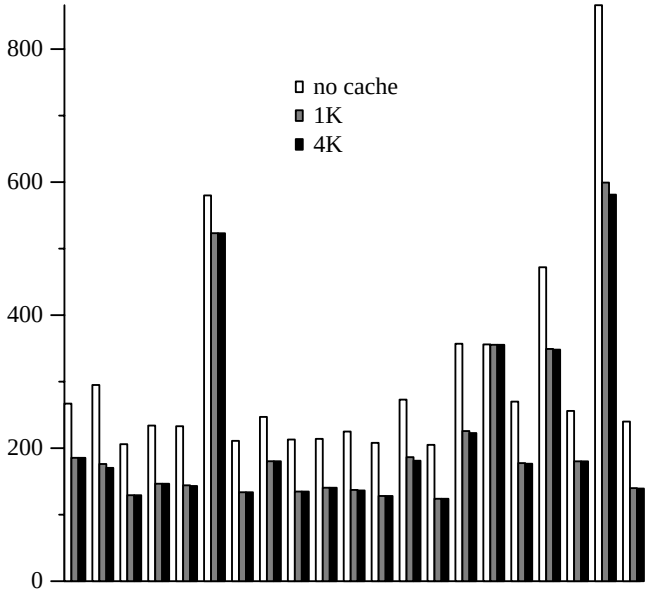

Figure 4: Comparison of 8-way set associative D-cache

4 Results

Figure 3 shows the total instruction count for a direct mapped cache under three conditions:

1. with no scalar D-cache at all,
2. with a 1KW scalar D-cache, and
3. with a 4KW scalar D-cache.

The Y-axis values are in millions of machine cycles. This shows the relative speedups experienced with the different sizes of caches. Figure 4 shows the total cycle count for an 8-way set associative cache under the same conditions as Figure 3.

Figure 5 shows the number of cache hits and misses for a direct mapped cache of sizes 1K and 4K words. As the table shows, quadrupling the size of the cache usually halves the number of cache misses. For some, such as Liv02, it does substantially better than this. For others, such as Liv01, the difference is minimal. The caches were always completely filled at the end of execution for all the programs.

When a cache was employed, it reduced the overall execution time for the programs by an average of 20% – 30%. We consider this a good result. Most of the time saved is in code that is ancillary to the actual computational sections. In the Livermore loops, most of the sections that perform the actual mathematics are heavily vectorized. The code that surrounds this, i.e.: the code that performs the setup, timing, and output, is where the time is saved. In code that has less of this type of non-vectorizable code, the savings would obviously be less.

5 Conclusions

The influence that a scalar-only data cache has on a modern vector processor was studied. A highly-optimizing

compiler, a cycle-by-cycle simulator, and an industry standard benchmark were used to produce a set of data. These data demonstrated that the addition of the D-cache had a good effect on the overall run time for the programs. The speedup was on the order of 20% – 30% with most of the speedup coming from the timing and I/O parts of the program, parts that have a lot of scalar code.

6 Bibliography

References

[ASU86] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers—Principles, Techniques, and Tools*. Addison-Wesley, Reading, MA, USA, 1986.

[Die94] Keith Diefendorff. "History of the PowerPC architecture". *Communications of the ACM*, 37(6):28–33, June 1994.

[HP90] John L. Hennessy and David A. Patterson. *Computer Architecture—A Quantitative Approach*. Morgan Kaufmann Publishers, Los Altos, CA 94022, USA, 1990.

[Hsu94] Peter Yan-Tek Hsu. "Design of the TFP microprocessor". *IEEE Micro*, 14(2), April 1994.

[KSF+94] L. I. Kontothanassis, R. A. Sugumar, G. J. Faanes, J. E. Smith, and M. L. Scott. "Cache performance in vector supercomputers". In *Proceedings Supercomputing '94*, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, November 1994. IEEE Computer Society Press.

[McM86] F.H. McMahon. "The Livermore FORTRAN kernels: A computer test of numerical performance range". Technical report, Lawrence Livermore National Laboratory, December 1986.

[MWV92] S. Mirapuri, M. Woodacre, and N. Vasseghi. "The Mips R4000 processor". *IEEE Micro*, 12(4):10–22, April 1992.

[Sit92] Richard L. Sites. "Alpha AXP architecture". *Digital Technical Journal*, 4(4), 1992.

[SWL+92] M. L. Simmons, H. J. Wasserman, O. M. Lubeck, C. Eoyang, R. Mendez, H. Hirada, and M. Ishiguro. "A performance comparison of four supercomputers". *Comm. of the ACM*, 35(8):116, August 1992.

Program	Size	Hits	Misses	%
Liv01	1KW	1,322,331	1,234,647	48.29
Liv01	4KW	1,327,240	1,229,738	48.09
Liv02	1KW	3,236,353	1,263,850	28.08
Liv02	4KW	4,466,249	33,954	0.75
Liv03	1KW	2,092,484	31,983	1.51
Liv03	4KW	2,108,293	16,174	0.76
Liv04	1KW	2,312,890	34,441	1.47
Liv04	4KW	2,329,505	17,826	0.76
Liv05	1KW	2,397,759	797,372	24.96
Liv05	4KW	3,158,741	36,390	1.14
Liv06	1KW	6,563,853	49,862	0.75
Liv06	4KW	6,588,990	24,725	0.37
Liv07	1KW	2,106,262	31,861	1.49
Liv07	4KW	2,121,218	16,905	0.79
Liv08	1KW	2,267,478	32,813	1.43
Liv08	4KW	2,282,308	17,983	0.78
Liv09	1KW	2,143,885	31,630	1.45
Liv09	4KW	2,158,197	17,318	0.80
Liv10	1KW	2,105,415	32,780	1.53
Liv10	4KW	2,120,787	17,408	0.81
Liv11	1KW	2,316,582	396,557	14.62
Liv11	4KW	2,678,438	34,701	1.28
Liv12	1KW	2,084,367	31,436	1.49
Liv12	4KW	2,098,875	16,928	0.80
Liv14	1KW	3,900,313	369,082	8.64
Liv14	4KW	3,993,904	275,491	6.45
Liv16	1KW	2,092,466	32,361	1.52
Liv16	4KW	2,106,897	17,930	0.84
Liv17	1KW	3,510,736	181,843	4.92
Liv17	4KW	3,655,794	36,785	1.00
Liv18	1KW	2,121,094	45,901	2.12
Liv18	4KW	2,148,204	18,791	0.87
Liv19	1KW	4,031,598	179,885	4.27
Liv19	4KW	4,189,937	21,546	0.51
Liv20	1KW	3,588,070	3,137,605	46.65
Liv20	4KW	4,984,789	1,740,886	25.88
Liv22	1KW	2,315,210	37,281	1.58
Liv22	4KW	2,330,077	22,414	0.95
Liv23	1KW	13,276,091	14,059,896	51.43
Liv23	4KW	22,592,297	4,743,690	17.35
Liv24	1KW	2,778,092	97,527	3.39
Liv24	4KW	2,859,043	16,576	0.58

Figure 5: Percentage of 1-Way D-cache Read Misses