SV1e Performance of User Codes

Tom Baring and Jeff McAllister
Arctic Region Supercomputing Center, University of Alaska Fairbanks 99775
{baring,mcallist}@arsc.edu

Abstract: The first SV1e processor upgrade at a user site was accomplished at the Arctic Region Supercomputing Center (ARSC) on April 11, 2001. In general, the CPU upgrade, in advance of the "X" memory upgrade, has not improved performance as much as might be expected. We discuss performance data collected on several significant user codes as part of understanding what allows some codes to take advantage of CPU speed increase alone while others require corresponding CPU and memory rate improvements for increased performance. This upgrade underscores an important point: dissociating performance from memory speed is becoming an increasingly important part of using modern computer architectures.

1 Introduction

At the moment, chilkoot has an unusual intermediate hardware configuration. Our SV1's processors are upgraded, but the memory is not. Most sites will upgrade both processors and memory simultaneously. This paper explores the effect of the CPU upgrade on four user codes to understand more precisely why only certain codes show performance improvement. This "upgrade in component steps" provides insight: with CPU speed improvements so much easier to come by than memory speed improvements, upgrades in the future may start looking more and more like this one. With the gap between CPU and memory rates growing greater exponentially, learning why performance improves with faster CPU speeds in some cases and not others is part of developing steps to help users take advantage of hardware advances. In this paper we also consider several metrics which help describe how codes will perform as CPU and memory become more unbalanced.

2 SV1 and SV1ex

The SV1e processor upgrade allows us to see the effect of a CPU speed increase alone, in contrast to other upgrades (i.e. J90-SV1) where CPU speed and memory bandwidth improved simultaneously on a similar architecture. The clock speed on the original SV1 processor is 300MHz. The SV1e processor is 66% faster at 500MHz. However, bandwidth from chip to memory is unchanged. Thus, in proportion to how efficiently a code uses memory, memory speed drives performance -- and this characteristic describes how close performance will remain to SV1 levels until completion of the upgrade path with "X" memory. "Re-balancing" is fortunate in this case, but -- given the industry-wide lag in memory speeds -- future systems are likely to grow more distant from CPU/memory parity.

"Machine Balance" as a metric, is growing more important in discussing performance. It is a measure of a memory subsystem's ability to supply data to a processor at the same rate it can perform work. It is simply the ratio of peak floating point operations per second to measured peak bandwidth [4]. Machine balance isn't necessarily good or bad and doesn't have a direct correlation to performance. However, adding additional memory speed will have diminishing effect as this metric goes toward zero, and adding additional CPU speed will have diminishing effect as this metric increases past one.

ARSC installed a Cray Inc., SV1 in September, 2000 replacing its previous parallel vector processor (PVP), a Cray J90. Even though the machine balance worsened (table 1), the memory and CPU speed increase benefited everyone.

Table 1

Platform	Peak per CPU MFLOPS	Peak per CPU Mem Ref/Sec	Machine Balance
J90	200	200	1.0
SV1	1200	315	3.8
"SV1e"	2000	315	6.4
SV1ex	2000	475	4.2

In contrast, on April 11, 2001, ARSC upgraded the SV1 processors to the “SV1e” version. With the CPU speed and memory bandwidth already out of balance toward too much CPU speed, in theory, increasing CPU speed would not help as much as enhancing memory bandwidth. Our observed data--system-wide and in the four cases studied in greater detail in this paper—concurs. Only codes with certain characteristics limiting their dependence on main memory speed benefited from this upgrade. (Extrapolating from this trend, if balance were to continue to deteriorate, soon only the most memory independent codes would see any improvement from added CPU speed.)

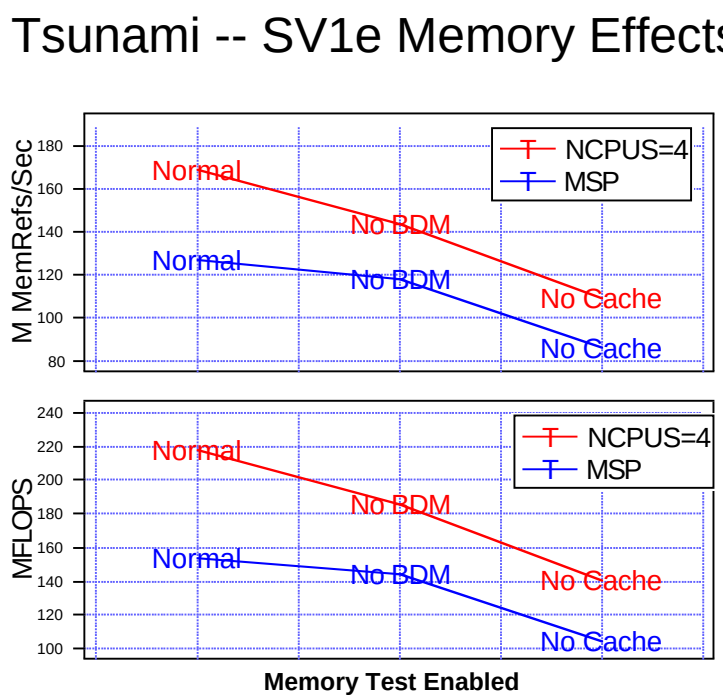
Fortunately, ARSC will upgrade the memory subsystem to “X” memory as soon as it becomes available (at which point chilkoot will be an SV1ex). The “X” memory is projected to improve peak per CPU memory bandwidth by a factor of about 1.5 and to approximately double overall system memory bandwidth [2]. This upgrade will mean that data becomes available for processor operations sooner, and calculations can proceed at rates closer to the processor's clock speed.

When looking at balance, the "missing" side drives performance. As John McCalpin wrote in 1995, in the past, "floating-point operations were considered quite expensive, often costing 10 times as much as an uncached memory reference. Today the situation is dramatically reversed" [4]. Now, it can generally be said that memory bandwidth drives performance. To prove this to ourselves, we switched off two hardware facilities, cache and bidirectional memory (BDM), for test runs of one of our user codes, Tsunami. The user-level commands to do this are:

```
/etc/cpu –m ecdoff  ./a.out   # Run with cache off
/etc/cpu –m bdmoff  ./a.out  # Run with BDM off
```

We draw some obvious conclusions from graphs (figure 1) of results from these tests.

Figure 1



First, for Tsunami, BDM and Cache contribute positively to performance. An "injury" to the memory subsystem (turning BDM off) causes a reduction in memory references. Second, the reduction in memory bandwidth causes a proportionate drop in MFLOPS. This supports our contention that the memory upgrade will have a positive effect on this code's performance, and that performance is, to some degree or another, driven by memory speed.

Fortunately, memory speed is only one of the many factors that go into performance. Efficient reuse of cached data can compensate. On the SV1e, cache memory resides on the same chip as the CPU, so the access rate can increase with clock speed. This is illustrated in the graph above. This code is somewhat cache-friendly, deriving at least part of its performance from cache reuse. While not among the best performers of the codes tested, this code achieves at least some independence from memory speed.

3 Achieving Peak Performance on the SV1

Achieving peak performance requires awareness of the underlying architecture at some level or another. Achieving acceptable performance gains with hardware upgrades (any vendor) now also requires an awareness of and efficiency with memory. It is no longer sufficient to merely consider CPU efficiency when developing algorithms.

Programming for increasingly unbalanced platforms goes hand in hand with existing performance considerations. For example, cache memory--in addition to being faster--is more responsive to CPU speed increases. Also, describing codes in terms of how close they are to peak according to the following performance metrics plays a part in evaluating which codes will respond to CPU speed increases and which must rely on memory speed for improvement.

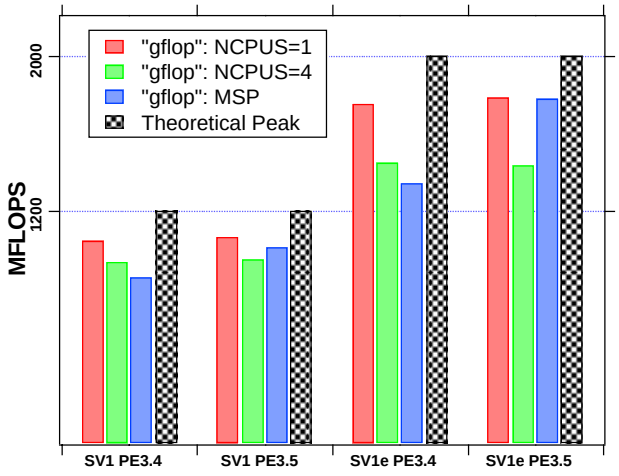
Performance Metrics for SV1 Codes:

1. high vector/scalar ratio (MFLOPS/MIPS)
2. high computational intensity (MFLOPS/M memory references per sec)
3. high cache/memory refs ratio (M cache hits per sec/M memory references per sec)

In general, codes which display these characteristics (especially the last two) perform well on any system. A major part of high CPU throughput is optimizing communication with memory, and this also turns out to be the best way to reduce the effects of machine imbalance.

SV1 codes with all of these characteristics can achieve very high performance and respond to CPU speed increases. ARSC held a contest in issue 213 of its HPC Users' Newsletter (http://www.arsc.edu/pubs/HPCnews), seeking codes which would achieve near peak performance on the SV1. The winner, dubbed "gflop," achieves about 90% of peak and its timings, under both PrgEnv 3.4 and 3.5, and both processors are shown in figure 2. The "MFLOPS" values shown are per CPU.

Figure 2



Although scientific applications on ARSC PVP systems return a smaller percentage of peak, "gflop" demonstrates the limits of actual, albeit "toy," codes, and could demonstrate the performance of a portion of a real code. It shows the potential speedup available from the SV1-"SV1e" upgrade. This code scores well on the three metrics mentioned above, and is not dependent on memory speed for increased performance. This supports our thesis, and indicates that if future optimizations on user codes strove to

push the same metrics in the same directions their performance would be closer to the peak CPU rate -- and they would also benefit more from increases in CPU rate.

4 Individual User Codes

Throughout the programming environment and processor upgrades, we have run tests using four user applications. All performance numbers were obtained using “hpm”, the Cray hardware performance monitor, and “ja,” the job accounting utility. The codes are arranged in order of observed performance improvement. Some codes displayed a wide range between tasking levels and we do not have SV1 performance data for all tasking combinations. Codes which did not see much improvement with the "SV1e" processor upgrade are expected to improve when the "X" memory is installed.

GAMESS

GAMESS is a well-known quantum chemistry package, and is run on one dedicated chilkoot CPU. This means it is locked into a processor, and although it can’t be swapped out, it does share the processor, cache, and memory subsystem. In going from the SV1 to the SV1e, GAMESS obtained the best performance boost we saw in a user code, essentially the clock speedup of 66%. The cache hit rate, memory reference rate, and MFLOPS rate all increased by approximately this amount. Total CPU time for equally sized jobs dropped a commensurate 38%. Note, however, that the code/data combination is dominated by scalar, not vector, operations, and obtains about 80 MFLOPS on the SV1e processor, or 4% of peak.

FLAPW

This is a well-vectorized quantum physics code which, in our runs on 4 CPUs, logged increases of 23% to 40% in its cache hit, memory reference, and MFLOPS rates, and a commensurate 18% drop in CPU seconds. It obtains 150-300+ MFLOPS in our test data, and is the highest performer in the group.

TSUNAMI

This is a finite difference code, written with traditional Fortran 77 style DO loops, and achieves reasonable vector performance. It solves the shallow-water wave equations for nested grids of increasing resolution. On one SV1e processor, it sustains 250 MFLOPS, an improvement of 6% over the SV1.

POLAIR

This is a coupled ocean/land/atmosphere/ice model, and is unique in having been written using Fortran 90 constructs, such as WHERE statements and array syntax to the elimination of DO loops. It is extremely memory bandwidth limited due to its algorithm for computing finite differences of adjacent grid points. Rather than implement this as a 4-point stencil in nested DO loops, which would be cache and memory friendly, it duplicates the array four times, shifting it N, S, E, and W, processing its boundary

conditions, and then evaluates the stencil by comparing the same grid points on each of the four shifted grids plus the original.

Interestingly, POLAIR appears to use the vector registers and processors, but given the huge load on the memory subsystem, only achieves 85 MFLOPS per CPU when run on 4 CPUS. As the most memory-dependent code tested, it is the least responsive to CPU speed increase. Different tasking levels showed widely different speedups, ranging from -20% to 11%, though the average is close to 0%.

5 Summary And Conclusions

The following table summarizes the metrics we've considered and shows how the four codes score, using average of performance increases at 1, 2, 4, and 8 processors (where available):

Table 2

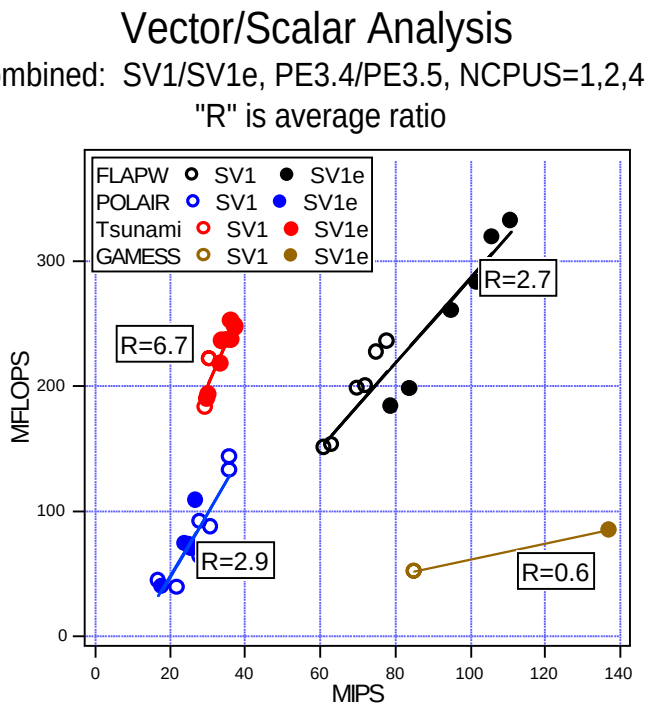
	GAMESS	FLAPW	TSUNAMI	POLAIR
Vectorization	low	high	high	high
Comp. intensity	low	high	medium	low
Cache/Mem ref ratio	high	high	low	low
MFLOPS increase	66%	29%	6%	0%

Within this group of codes, use of cache instead of memory (high cache/mem ratio) is the most effective indicator of how well a code will respond to CPU speed increases. The largest gains went to the most cache-friendly codes while the least went to the most memory-intensive. As independence from main memory speed is the goal, this conclusion makes sense.

A plot of MFLOPS as a function of MIPs, for four of the codes tested (figure 4), shows the range of vector use.

In this and subsequent plots, auto-tasked runs made on 1, 2, 4, and 8 processors are all included (when available), as are runs compiled under both programming environments 3.4 and 3.5, as are runs on both the SV1 and SV1e processors. The auto-tasked runs account (in general) for the spread of points along the linear least squares fitted lines, and the SV1e runs (in general) out-perform the SV1 runs. The values plotted are per processor averages, even for the multiple-CPU runs.

Figure 3



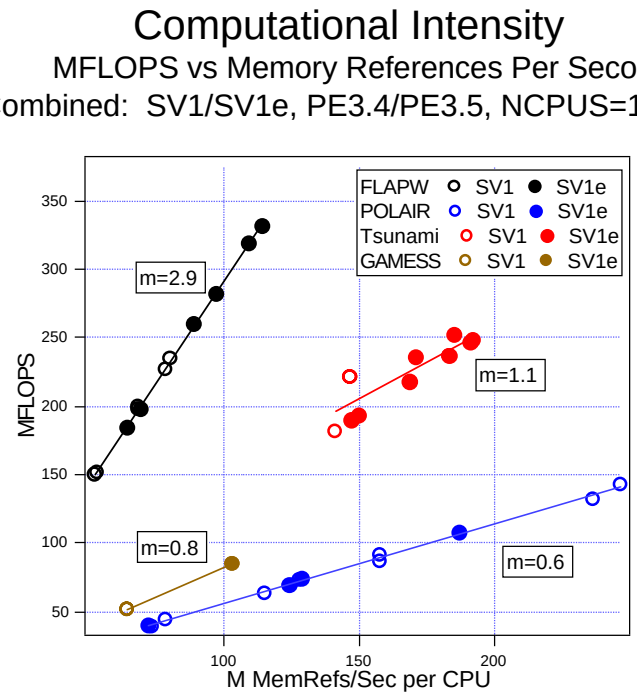
The value, “R” shown for each code is the ratio of MFLOPS to MIPS, and thus represents the number of operations per instruction averaged over entire runs of the code. Each ratio, “R,” is actually computed as the average of the ratios for all the runs.

Degree of vectorization, though it is crucial to SV1 performance, does not contribute much negatively or positively to CPU-based performance improvement. Vectorization is a magnification of otherwise existing patterns -- the same operations, loads, and stores done on many memory elements. While it magnifies MFLOPS rates, other effects are more important in determining how much a code can benefit from increased CPU speed alone.

GAMESS, a scalar code, showed the best performance increase. Tsunami, a reasonable vector code, showed the second worst performance increase. POLAIR achieves vector-like ratios of operations per instruction, scalar-like ratios of operations per second, and the worst performance increase. Degree of vectorization is clearly a poor metric for assessing a code’s independence from memory bandwidth.

Another useful comparison is operations performed per memory reference. Cray’s guide to “Optimizing Application Code on UNICOS Systems” [3] defines computational intensity as the memory reference rate divided by the floating point operations rate. We’ve plotted the inverse ratio (figure 4) to maintain consistent axes with other graphs, and because we believe that in general, MFLOPS is the dependent variable, relative to memory reference rates.

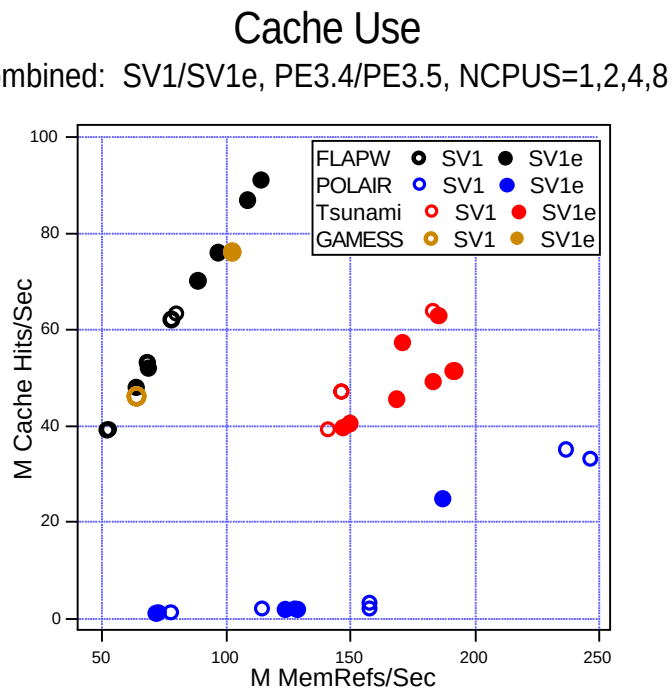
Figure 4



In this plot, the slopes of the lines are labeled as "m" and show that, for instance, FLAPW performs almost three operations for every memory reference. Accepting the relative scatter of point in the case of Tsunami, the linear trends of these lines indicate that the computational intensity in the codes is inherent in the algorithms, their implementation, and their treatment under the compilers. The upgrades to programming environment and processor have not altered the codes' computational intensities.

FLAPW, one of the codes showing the best performance increases, scores highest according to this metric. It performs more computations per memory access and therefore is more independent of memory speed. POLAIR will remain subservient to memory bandwidth improvements of the future and thus, is under evaluation for deep changes to its basic algorithm.

Figure 5

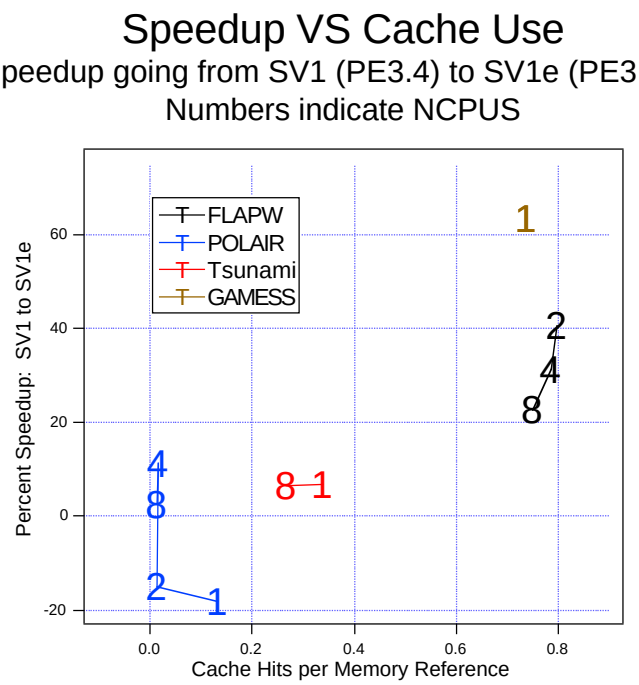


The cache use graph shows the proportion of memory references which are satisfied from cache.

It places the most scalar of the codes, GAMESS, on top of the best performer, FLAPW. In one sense, both of these codes are doing exactly what we want: making fewer memory references, and reusing them from cache more often. These two codes received the greatest improvement from the SV1e processors in all categories: cache hits, MIPS, memory references, MFLOPS, and CPU time. This is due to the codes' ability to reuse data from cache. CPU to cache bandwidth increased by 66% with the 66% clock speed increase, thus increasing effective memory bandwidth for these cache-friendly codes, despite the fact that the rest of the memory subsystem was not upgraded.

This relationship is further highlighted by the following graph (figure 6), which relates CPU-based speedup to the ratio of cache hits per memory reference. This ratio turns out to be especially telling, as it underscores where a code is satisfying most of its memory needs. If most memory references result in a cache miss and retrieval from main memory, the code cannot escape the relative slowness of that memory. On the other hand, the more a code is able to use cache (or, even better, registers) the more independent it can be from memory speed.

Figure 6



In a world where CPU speeds are generally improving much faster than memory bandwidth, we're glad that the SV1ex provides, not only a substantial upgrade, but machine balance only slightly larger than its predecessor, the SV1. Our experience with the "SV1e" processors underscores just how significant, and necessary, the faster "X" memory is.

Getting this "upgrade in component parts" has been a valuable study in machine balance. The insight into ways to decouple performance from memory speed will improve our ability to advise users towards the most productive way to optimize codes not just for present performance, but so codes can continue to do more as hardware is upgraded.

References

[1]: Maynard Brandt, Jeff Brooks, Margaret Cahir, Tom Hewitt, Enrique Lopez-Pineda, Dick Sandness.. "The Benchmarker's Guide for CRAY SV1 Systems", Cray Inc., July 20, 2000

[2]: Personal communication with Cray Inc. SV1 Project personnel. May, 2001

[3]: "Optimizing Application Code on UNICOS Systems", Cray online Software Publications, 004-2192-003

[4]: John D. McCalpin. "Memory Bandwidth and Machine Balance in Current High Performance Computers", IEEE Technical Committee on Computer Architecture newsletter, December 1995.

Acknowledgements:
Thanks to the users and creators of the codes used in this article. Thanks also to John Metzner (Cray Inc.) and Guy Robinson (ARSC).