Memory Bandwidth and the Performance of Scientific Applications: A Study of the AMD Opteron™ Processor

By Philip J. Mucci
Innovative Computing Laboratory, University of Tennessee, Knoxville

Abstract

The goal of this paper is to evaluate the memory subsystem performance of the AMD Reference Design Kit (RDK), “Quartet”, a four-CPU system based on the AMD Opteron™ processor. We begin by discussing the unique features of the AMD Opteron processor's memory subsystem and the system architecture of the RDK “Quartet”. Next, we introduce four microbenchmarks used to characterize different aspects of memory subsystem performance. STREAM and CacheBench measure sustained memory bandwidth for a variety of compute operations. LMBench and GUPS evaluate the latency of independent and dependent memory accesses to pseudo-random locations. Each of these four benchmarks is run on three different systems, the four-processor AMD RDK “Quartet”, a four-processor Intel® Itanium® 2 and a two-processor Intel Xeon™ processor. The results are presented in terms of delivered per-CPU performance. Next, we introduce POP and MILC, two large parallel simulations from the Department of Energy Scientific Discovery through Advanced Computing (SciDAC) program. POP or Parallel Ocean Program is a grid-based ocean circulation model from Los Alamos National Laboratory. MILC or MIMD Lattice Computation is a four-dimensional quantum chromodynamics simulation from the University of Utah. Both applications have undergone extensive performance analysis on a wide variety of platforms and their behavior is reasonably well understood. Each application is compiled with the latest release of the Portland Group compiler and runs on the AMD RDK “Quartet” using a shared-memory implementation of MPI (mpich v1.2.5.2). Aggregate hardware performance metrics are collected using a tool from PAPI, the Performance Application Programming Interface. Using this data, we evaluate the applications' performance and explore its relationship to that of the memory subsystem. We conclude with comments on the AMD RDK system’s architecture and its performance on large scientific applications.

Introduction

The memory wall is nearly upon us. No longer do gains in processor clock speed automatically translate to similar increases in application performance. This is primarily due to the growing disparity in performance between the memory subsystem and the CPU. Von Neumann[1] knew that this was a problem, however there were much larger issues than performance to tackle at that time[3]. It wasn't until years later that J. W. Backus[2] popularized the issue and coined the term, the Von Neumann bottleneck. Things have gotten progressively worse. Processor clock rates increase by roughly 60% each year, while memory access time drops only 7%, as shown in Figure 1[4]. Short of any revolutions like the widespread adoption of a processor-in-memory architecture[5][6][7][8][46], memory subsystem performance will continue to be the industry's biggest bottleneck. This is especially a problem in the performance of large scientific simulations run on large clusters and supercomputers, where applications frequently run at the limits of available physical memory.

Here we will explore the memory subsystem performance of the AMD RDK “Quartet”, a four-processor system built around the AMD Opteron processor. The AMD Opteron processor is unique in that it is the first x86 compatible processor to feature a built-in memory controller on each CPU. This design aims to directly address the Von Neumann bottleneck by bringing the DRAM closer to the CPU. We start by introducing four well known memory-centric micro-benchmarks: CacheBench, STREAM, LMBench and GUPS. Each of these benchmarks has played an important role in the evaluation and procurement of next-

generation clusters by the technical computing community. At their most basic, CacheBench[22] and STREAM[12] measure memory bandwidth, and LMBench[25] and GUPS[26] measure memory latency. Here we present and analyze unofficial results for runs of each of these benchmarks on the four-processor AMD RDK “Quartet” as well as on a four-processor Intel Itanium 2 processor (Madison) and a two-processor Pentium® IV (Intel Xeon processor). Following that, we evaluate the performance of the “Quartet” with two heavily analyzed production applications taken from the Department of Energy's Scientific Discovery Through Advanced Computing (SciDAC) effort. The first code is POP[28], an ocean circulation model originally developed at Los Alamos National Laboratory. POP is used as the ocean component in many of the world's leading coupled-climate models. The second code is MILC[33], a four-dimensional lattice gauge theory simulation. These two codes are run in a 'large benchmark' mode included with each distribution. Each simulation is run under a PAPI[40] performance tool to gather data from the AMD Opteron processor's onboard performance monitor. Using this data, we analyze the performance of the codes in the context of memory subsystem performance and overall processor utilization. Finally, we make some concluding remarks about the advantages of running memory intensive scientific applications on the AMD-powered system.

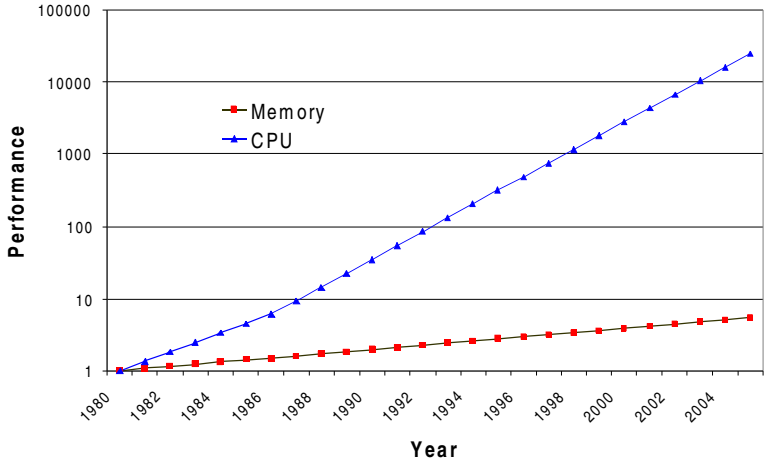

Figure 1. The growing gap in processor and memory speeds[4][41].

DRAM Technology

While DRAM technology appears stagnant in the context of Moore's Law, it is important to recognize that there has been some progress made in recent years. This is particularly true in the area of memory bandwidth, the speed at which block data can be accessed. Peak memory bandwidth is usually computed by multiplying the speed of the memory bus by the width of the memory bus. With the development of double data rate static DRAM (DDR SDRAM) and Rambus DRAM (RDRAM), peak bandwidths have doubled while delivered bandwidth has not[9][10][11][12][34]. This is because the fundamental problem remains; that memory lies too far from the CPU. System architects have long attempted to hide this problem with additions like multiple levels of non-blocking cache, sophisticated hardware prefetching and out-of-order execution[35][36][38][39][45]. However, these methods have only been moderately successful, largely due to the wide spectrum of applications being run on today's commodity processors and the additional work that the compiler must do to exploit these resources. Is this performance gap caused by the lack of memory bandwidth or too much memory latency? As a general rule, unit-stride compute kernels suffer most from bandwidth constraints, while kernels with non-unit stride suffer from latency concerns[12][13][14][37]. Furthermore, applications that exhibit some degree of spatial and temporal locality perform better than those that do not because

good locality matches the usage model for which the caches were designed[8]. For example, the dense matrix multiplication routine (GEMM) found in most systems' linear algebra libraries frequently runs at 80 to 90 percent of the peak speed of the processor after exhaustive optimization[14]. The dominating factor for GEMM is memory bandwidth; moving blocks of contiguous data to/from various levels of cache. Algorithms that do not lend themselves to prefetching, blocking, tiling or other intelligent cache-line re-use strategies continue to perform poorly[13]. Consider the sparse case where the matrices are in compressed row storage (CRS) format. Performance is dominated by dependent indirect array references with little or no locality. Thus, the code and data structures require major restructuring to obtain even mediocre performance in the 20 to 30 percent of peak range [13].

It is for reasons like these that chip architects have begun to explore ways to move the memory closer to the CPU. In traditional design, the memory is connected to the processor via a memory controller. This controller sits on the CPU's data bus, commonly referred to as the front side bus. This bus runs at a fraction of the processor clock speed. The memory controller is also attached to the memory bus, which runs at low speed. At the time of this writing, systems with an 800 MHz front side bus have been announced for processors in the 3+ GHz range. The latest cycle time for DDR SDRAM[15] is 200 MHz (commonly called DDR400), a factor of four slower than the controller and a factor of fifteen slower than the processor. It is important to note that the front side bus is additionally responsible for handling all I/O traffic. All I/O transactions contend with memory accesses for bus bandwidth. On multiprocessor machines, the problem of contention is aggravated by multiple CPU's contending for access to the memory controller. As the controller is a shared resource, it must be negotiated for before it can be used. This arbitration process drives up the access time. Every additional CPU proportionally increases the load on the controller and the memory bus. Designs that do not provide adequate bandwidth are very susceptible to poor scaling of applications. Because the memory subsystem is so much slower than the CPU, additional CPUs working on a problem can easily starve each other of available memory bandwidth. Even with the industry’s latest memory controllers, the aggregate bandwidth available to all processors can be consumed by a single CPU[16][17][18]. SMP systems of this design lack true scalability. While this might not be an issue for transaction processing and other server related tasks, it is at the fulcrum of the performance of scientific applications. The only solution to this problem is to provide all available DRAM bandwidth on a dedicated channel by giving each processor a memory controller of its own. Since the controller is no longer a shared resource, it can be directly incorporated into the CPU's die and run at full processor speed.

The AMD Opteron processor is the first general-purpose, high-performance commodity processor with an on-board memory controller. This idea is not revolutionary in itself. Previously, on-board memory controllers were found in specialized graphic coprocessors like those from Sony and NVidia or in expensive CPU's like HP's Alpha EV7/8, Sun's UltraSPARC III and IBM's Power 4. This feature combined with 8-way glueless SMP capabilities make the AMD Opteron processor an ideal building block for technical compute servers. Memory bandwidth scales linearly with each additional processor as each has its own dedicated pathway to memory. Since the memory controller is not shared, there is no arbitration process. This represents a significant decrease in latency of accesses. Perhaps the most important difference about the on-board memory controller is that the AMD Opteron processor does not share bandwidth between DRAM and any I/O devices on the system. The channel between memory and the CPU is entirely dedicated to memory traffic. I/O is performed by a completely separate on-chip controller that operates over a standardized, high bandwidth, point-to-point fabric called HyperTransport™ technology.

The AMD Opteron processor has additional features that further improve the performance of the memory subsystem. A complete discussion[19][20][21] of these is

beyond the scope of this paper, but we include the most important features here. First, the AMD Opteron processor is an aggressive, out-of-order, superscalar design, meaning that multiple instructions complete as soon as their dependencies are satisfied. The processor has a built-in hardware prefetching algorithm. Subsequent misses to consecutive cache lines cause the memory controller to bring in as many as four cache lines ahead to the level 2 cache. The processor has deep load/store queues. These queues allow the processor to issue memory operations without having to wait until they complete. The AMD Opteron processor has an eight-entry queue for data references to the level 1 cache and a unified 32-entry queue to level two. The processor supports data forwarding, also known as critical word first. In normal applications, a miss is rarely triggered by the first word in a cache line. Since the basic transfer unit is a cache line, processors without data forwarding have to wait until the entire line is transferred before computation can resume. On the AMD Opteron processor, the word that missed the cache is delivered first, thus satisfying the dependency so that the load or store instruction completes. The remainder of the cache line is then transferred asynchronously as the computation continues.

The AMD Reference Platform, "Quartet"

Each processor in the AMD Quartet reference system is an AMD Opteron processor model 848 (2.2GHz) with 8GB of DDR333 SDRAM. Peak per processor memory bandwidth is 5.3 GB/sec[1]. As each processor has its own memory controller, peak aggregate memory bandwidth is 21.2 GB/sec. Each processor is connected to two adjacent neighbors by way of a HyperTransport technology link running at 3.2 GB/sec in each direction (providing up to 19.2 GB/sec peak bandwidth per processor). Two other systems are used in our comparisons, a Dell Precision Workstation 530 and an Intel Tiger 4. Characteristics of the systems tested are summarized in Table 1.

Microbenchmarks

Over the years, numerous microbenchmarks have been developed to analyze the performance of the memory subsystem. Here we present results from four popular microbenchmarks: CacheBench, Stream, LMbench and GUPS. The numbers reported are the average of four runs on each of the three systems. Each system is fully subscribed; every processor is busy running a copy of the same serial benchmark. It is important to note that the results here are both unofficial and un-tuned. Commercial versions of the benchmarks have been exhaustively tuned to generate the best possible numbers. Here, no tuning is done beyond enabling aggressive optimization in the compiler.

[1] These figures are in units of 10^9 not 2^{30}.

	Intel Xeon processor	***Intel Itanium 2 "Madison"***	***AMD Opteron processor***
Clockrate (# CPU's)	2.4GHz (2)	1.3GHz (4)	2.2GHz (4)
Motherboard	Dell Precision 530	Intel Tiger 4	AMD Quartet
Level 1 Cache	16KB/16KB/4-way S.A.	16KB/16KB/4-way S.A.	128KB/2-way S.A.
Level 2 Cache	512KB/8-way S.A.	96KB/6-way S.A.	1MB/16-way S.A.
Level 3 Cache	-	3MB/12-way S.A.	-
Line Size	64B/64B	64B/64B/128B	64B/64B
Page Size	4KB	16KB	4KB
RAM	PC800 RDRAM	DDR266	DDR333
RAM BW	3.2GB[1]/sec (shared)	4.2GB[1]/sec (shared)	5.3GB[1]/sec (per CPU)
Total Mem.	2GB	32GB	32GB
OS	Redhat 7.3	SUSE 8.1	SUSE 9.1
Kernel	2.4.18-18.7	2.4.21.144	2.4.21-193
Compiler	GNU gcc 2.96	Intel ecc 7.1	PGI 5.1 Beta 2
Flags	-O3 -mpentiumpro	-O3 -ip -g	-tp k8-64 -fastsse

Table 1. Benchmark System Characteristics

CacheBench

CacheBench is designed to measure the peak usable memory bandwidth of all levels of the memory hierarchy. It is part of the LLCBench package[22][23] along with BLASBench and MPIBench. The goal of the LLCBench package is to parameterize important low-level performance characteristics of an architecture and its various subsystems. CacheBench works by initializing a large contiguous vector of doubles and then performing various operations on successively larger portions of that vector. It generates two sets of data for the three basic operations on memory: read, write and read-modify-write, the latter of which is implemented as an increment operation. The first set of data is for the native implementation of a doubly nested loop. The second set of data is for a tuned version where the inner loop has been unrolled eight times. The goal of the tuned version is to see how the compiler handles code that has already been optimized. In addition to these six operations, CacheBench also reports the performance of the memcpy() and memset() subroutines found in the C library. The memcpy() operation has long been known to be the limiting factor in I/O performance[42][43]. As a result, these routines are heavily optimized to take advantage of the processor's instruction set and microarchitecture. Thus these numbers can be used to place an upper bound on maximum memory bandwidth. Here we report measured bandwidth for the tuned and untuned versions of the read/modify/write loop as well as memcpy().

CacheBench Results

In Figure 2 we present the results from the un-tuned version of the Read-Modify-Write microbenchmark. Note that all levels of cache are visible on all the machines except for the Intel Itanium 2 processor ("Madison"). All accesses to double precision data bypass the level 1 cache on the Intel Itanium processor architecture. At vector lengths that fall outside of cache, the AMD Opteron processor is more than twice as fast as the Intel Itanium 2 "Madison" processor and nearly four times faster than the Intel Xeon processor. Note that the performance does not decrease when running out in main memory, as one might expect considering the increasing number of TLB misses. This is because all three processors have hardware prefetching. Since the benchmark operates with unit stride, the prefetching engine triggers TLB misses on cache lines well ahead of the current position. Thus by the time the processor issues a load instruction, the address of that load has already been computed and placed in the TLB.

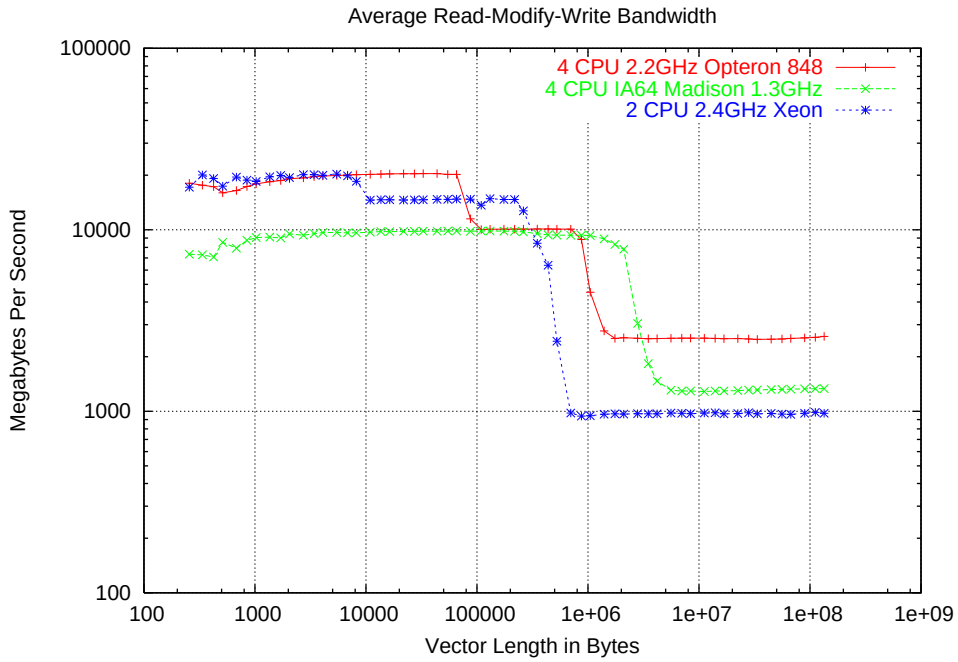

Figure 2. Per CPU Read-Modify-Write bandwidth from CacheBench

In Figure 3, the results for the hand-tuned version of the code are shown. We find that the AMD Opteron processor system again beats the other systems delivering more than twice the bandwidth of the Intel Xeon based system and over four times that of the Itanium 2 based system. It is interesting to note the differences between the previous two graphs. On the IA64 system, the performance of the un-tuned code is much higher than the tuned version for the same level of compiler optimization. Here the Intel compiler is not able to effectively optimize simple and portable optimized code. Loop unrolling is an established and effective optimization for RISC processors yet the Intel compiler fails dramatically. This is a serious problem as many production applications have already undergone similar rudimentary performance tuning.

In Figure 4, we first note that the level 1 cache plateau is now visible on the Intel Itanium 2 processor. This is because memcpy() does not use the floating point hardware to transfer data, thus the level 1 cache is used. Note thes ubstantial difference in the performance of memcpy() when running out of cache. The AMD Opteron processor-based system delivers more than twice the bandwidth of the Intel Itanium 2 and the Intel Xeon

processor-based systems. The excellent performance of memcpy() means a significant performance advantage in all operations that transfer data through the kernel. This directly translates to better filesystem and networking performance, especially on platforms with I/O subsystems capable of delivering gigabytes per second of bandwidth like RAID arrays and 10G Ethernet. As Linux has not yet been optimized to provide true zero-copy operation, memcpy() will continue to be the dominating factor in kernel I/O.

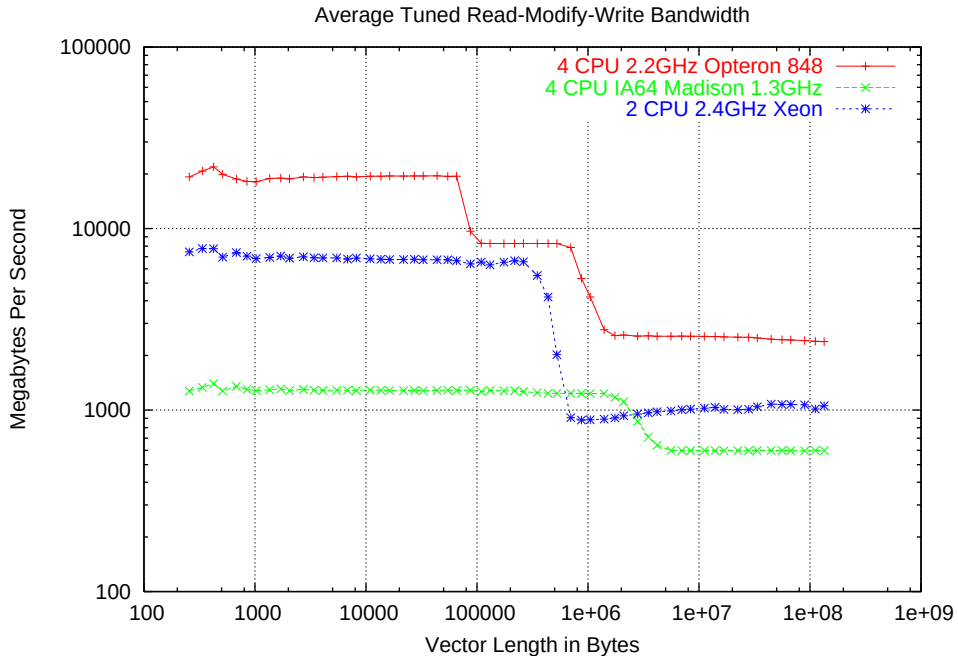

Figure 3. Per CPU Tuned Read-Modify-Write bandwidth from CacheBench

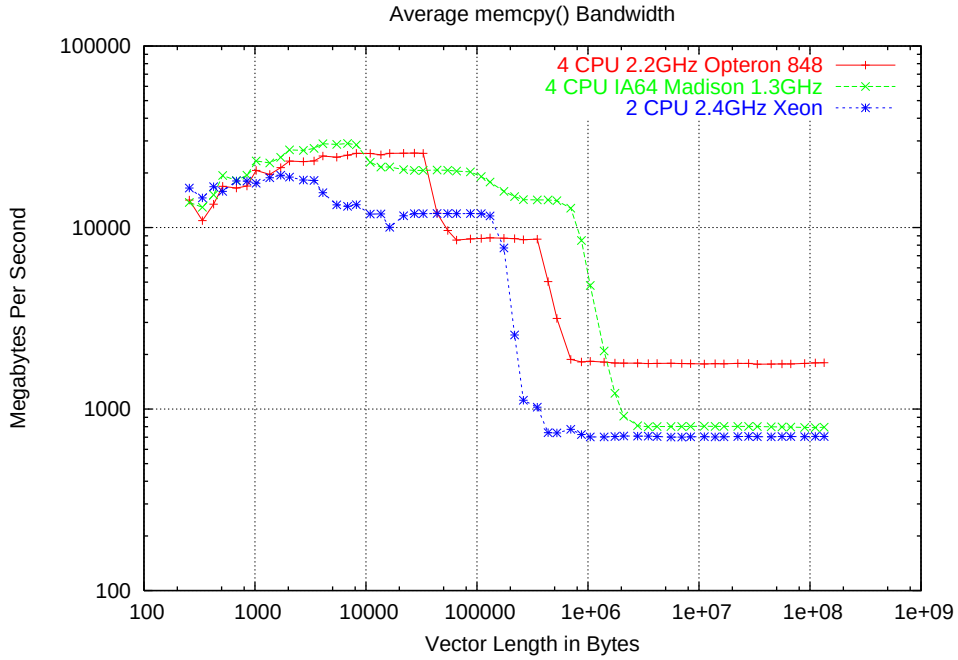

Figure 4. Per CPU memcpy() bandwidth from CacheBench

Stream

Stream is a popular benchmark designed to measure sustainable memory bandwidth for vector compute kernels[12]. It measures the bandwidth of the following operations:

Assignment	A[i]=B[i]
Scale	A[i] = constant * B[i]
Add	A[i] = B[i] + C[i]
AXPY	A[i] = B[i]+ scale * C[i]

Like CacheBench, this benchmark moves through memory with unit stride and thus benefits from any hardware prefetching done by the processor. Stream has demonstrated significant increases in performance as faster front side buses and double data rate memory technology have become prevalent. In Figure 5 we show the average results for the original unmodified sources for the C version of Stream with heavy compiler optimization enabled. Each of the above operations was repeated ten times on an array of 10,000,000 double precision numbers. Total memory required was approximately 230MB.

Stream Results

The AMD Opteron processor-based system again outperforms the other systems by a factor of between two and three, depending on the operation. Relative performance of each of the four operators appears to be about the same for the three systems. The exception is on the Itanium 2-based system where the Triad case is slightly faster than the Add case. This is due to the fused multiply-add instruction in the Intel Itanium processor architecture. This instruction executes in the same amount of time as an add instruction and thus makes better use of the available memory bandwidth.

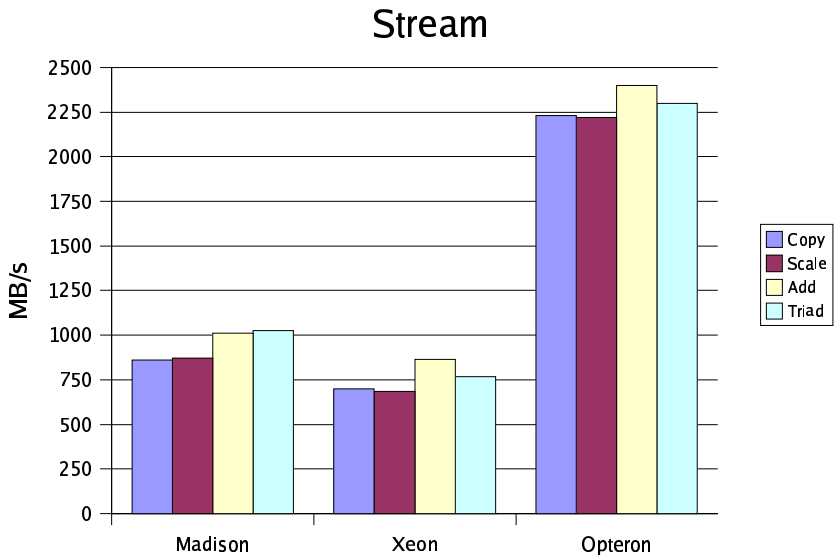

Figure 5. Per CPU bandwidth from Stream (C version)

```
for all sizes of p {
      p = head;
      while (p->p_next) {
            p = p->next;
      } }
```

Figure 6. Linked list traversal at the core of the memory read latency benchmark

LMbench and lat_mem_rd: Memory Read Latency

LMbench is a fairly comprehensive low-level benchmark suite that evaluates everything about a system, from processor to operating system to library performance[24] [25]. Here we use only one executable from the package; the memory read latency benchmark or **lat_mem_rd**. This benchmark is very similar to that of the ScienceMark benchmark popular with the Windows community. At the core of the benchmark is the performance of a linked list traversal, as shown in the pseudo-code in Figure 6. Notice the inner loop; the processor must wait for the value to be returned from the memory subsystem before the computation can continue. In practice, the code is highly unrolled to increase utilization of the pipeline, but the steps of execution are the same. The outer loop progressively increases the size of the data set, from 512 Bytes to 100 MB. As it does so, the various latencies of cache can be easily seen in the data. The only interesting input parameter to the benchmark is the stride between successive accesses. Since we wish to measure the latency of the memory subsystem and not the cache, we choose a value that will not trigger the prefetching hardware found on the processors. Here we set the stride to be two full cache lines plus 8 bytes. As all the systems have a 64 byte line size for the level 1 cache, the stride used was two cache lines plus two words or 136 bytes.

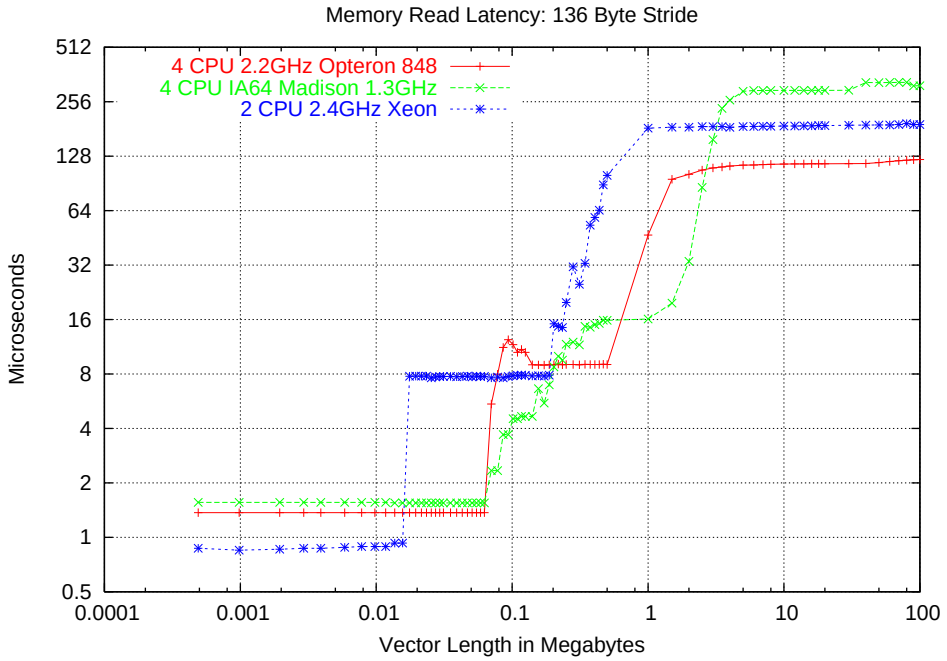

Figure 7. Per CPU memory access latency from lat_mem_rd in microseconds[2]

[2] Lower is better.

Memory Read Latency Results

In Figure 7 all three cache levels are visible for each of the platforms. As expected, the lowest latency is found for accesses to the level 1 cache of the processor with the highest clock rate; the 2.4GHz Intel Xeon processor. However, here we are interested in the performance of applications that exhaust the cache resources. So as we examine the curve out past the boundary of the caches, we notice that the average latency of the AMD Opteron processor-based system is one third that of the Itanium 2-based system and half that of the Intel Xeon-based system. The Intel Xeon processor's performance is undoubtedly a result of the higher clock rate of the RDRAM subsystem. Note the relationships of the peak numbers quoted in Table 1 versus that in this graph. Neither the peak bandwidth, processor clock rate, nor the clock rate of the memory subsystem predicts the performance seen here. Memory latency is inherently very difficult to quantify due to the complexity of the hardware and processes involved in accessing it. In this case, it is the AMD Opteron processor's on-chip memory controller that is responsible for the observed reduction in memory latency.

GUPS: Giga-Updates Per Second

GUPS, or Giga-Updates Per Second, was popularized as a paper and pencil benchmark used by the National Security Agency[4][26]. At its core, GUPS measures the time to perform a fixed number of updates to random locations in main memory. There are two versions of this benchmark: one that pre-computes the indices to be updated and one that computes them on the fly. For this paper, we have chosen the former because we found it to be more representative of the original workload[26]. Thus this benchmark contains two large tables, one containing the original data and the other containing the indices of the entries to be updated. The time to compute these indices is not included in the performance measurement. In the runs used for this paper, the main table has 16 million 64-bit entries for a total of 128MB. The index table has 64 million 32-bit entries as we update the entire table four times.

```
for (i = 0; i < iters; i++) {
     index = indices[i];
     data = field[index];
     data = data + iters;
     field[index] = data; }
```

Figure 8. Loop at the core of the GUPS Benchmark

GUPS Results

In Figure 9, we observe that the AMD Opteron processor-based system is 20 percent faster than the Itanium 2-based system and 10 percent faster than the Intel Xeon-based system. This benchmark is bound by the ability of the memory subsystem to provide reasonable performance of a dependent transaction. As with the memory latency benchmark, no amount of memory bandwidth will improve the performance of this benchmark because the costs are dominated by memory bus arbitration and DRAM access time. Here the AMD Opteron processor's performance is again due to the dedicated path each CPU has to main memory.

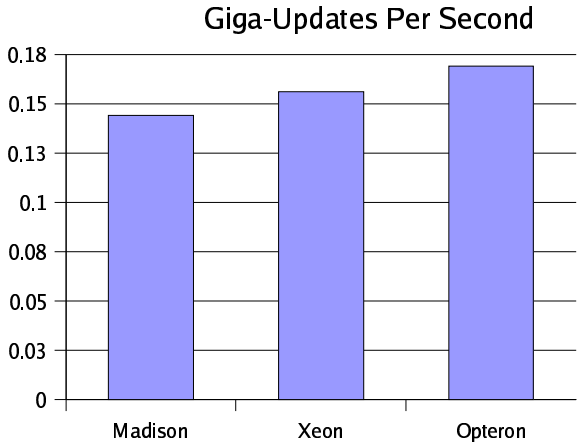

Figure 9. Per CPU Giga-Updates Per Second

POP 1.4.3: Parallel Ocean Program

The Parallel Ocean Program is an explicit finite-difference model developed at Los Alamos National Laboratory to perform high-resolution ocean circulation studies. POP is written entirely in Fortran 90 and makes extensive use of *modules*, *array syntax*, and *where* constructs. The model uses two-dimensional domain decomposition; communications (global reductions and broadcasts and nearest neighbor) are isolated within a few machine dependent procedures. Solves for the three-dimensional baroclinic pass require nearest neighbor communication. Solves for the two-dimensional barotropic pass use a conjugate gradient solver. The solver requires that global reductions, broadcasts and nearest neighbor communications be performed at the end of every iteration.

As a community code, POP has been extensively analyzed and optimized over the years. Jones and Levesque[27][28] demonstrated that the performance of POP could be tripled (to over 100 MFLOPS/s) on the Origin 2000 with some hand optimization. These changes included tuning for unit-stride, cache padding and loop fusion and fission. Many loops were split in order to avoid the costly masking operations prevalent throughout the compute loops. However, these optimizations never made it into POP 1.4, although the lessons learned provided the motivation for POP 2.0. Here we use version 1.4.3 of the code as it exists in the SciDAC application repository. Recent experiments of POP on the IBM Power 3 platform demonstrated low cache miss rates but high stall times, indicating poor scheduling of the compute loops[44]. Figure 10 illustrates some statistics collected from POPrun using the same x1 data set as the Power 3 experiments. We ran the code for 20 timesteps on all four processors of the AMD RDK “Quartet” using MPICH 1.2.5.2 and the shared memory device ch_shmem. Measurements from the AMD Opteron processor's hardware counters were taken using the papiex tool from the PAPI CVS tree[40] .

POP Results

% Mem Ops.	34.1	**% L1 Hitrate**	95.45
% Load Ins.	67.8	**% L2 Hitrate**	98.65
% Store Ins.	32.2	**% FPU Idle Cyc**	26.75
% Branch Ins.	9.3	**IPC**	0.6
% Packed SSE(2)	39.34	**FLOPS/Mem Op.**	0.8
% Peak	8.51	**% Cyc. Dispatch Stalled**	54.08
MFLOPS/S	374.0	**% Cyc. Dispatch Stalled Mem.**	15.25
		% Stalls are Mem. Stalls	28.69

Figure 10. Per CPU Performance Data of POP 1.4.3 on the AMD RDK "Quartet"

Here we find that the application does very well. The application gets nearly 10 percent of peak and delivers slightly more than a third of a gigaflop per second per CPU. The computational intensity of this application is not great: the POP is doing 0.8 floating point operations for every memory operation. As a result, opportunities for cache line re-use are limited although hit rates in both caches are quite good. The large percentage of cycles in which the dispatch stage is stalled is consistent with measurements taken on other platforms. Most importantly, note that dispatch was stalled due to a full load-store buffer for only 15 percent of all cycles. This low percentage is a direct result of having a dedicated, high-speed and lower latency path to main memory. It should be noted that a pre-release of the Portland Group v5.2 compiler was also tried on POP. The result was an approximate 40 percent decrease in runtime corresponding to a 40 percent decrease in the number of floating point instructions. The MFLOPS per second did not change significantly between the two versions.

MILC 6: MIMD Lattice Collaboration

MILC performs four dimensional lattice gauge theory simulations intended for the understanding of the behavior and properties of nuclear matter. The forces that binds nuclear matter together is so strong that their exploration cannot be performed using traditional calculation approaches. The theory of strongly interacting particles is known as Quantum Chromodynamics or QCD. QCD calculations are based on lattice gauge theory where time and space are represented as a four dimensional lattice. Answers are derived from this lattice by evaluating a multidimensional integral over this lattice. Due to the size of the lattice (the 4-D lattice commonly has 32 points on a side), the only way of solving the integral in a reasonable amount of time is with a Monte Carlo method and the conjugate gradient technique. MILC is widely used by the nuclear physics community; the San Diego Supercomputing Center reports that MILC consumes well over a million CPU hours a year. For our purposes, we run the su3_rmd benchmark that uses the conjugate gradient method on the Kogut-Susskind quarks. Gottlieb et. al. report [29][30][31][32] that the calculations for these quarks require significantly more memory bandwidth and thus this data set is chosen for the work here. Experiments indicate that performance is largely limited by memory bandwidth, cache and TLB capacity for production lattice sizes. For our runs on MILC, we use the 14 mode case (nx,ny,nz) and nt is 56. We run the code again using MPICH 1.2.5.2 with the ch_shmem shared memory device. It should be noted that a more recent version of MILC (6.2) exists that has been heavily hand-optimized for the x86 architecture. This version was not used in our experiments, instead we use the stock MILC 6 release from the SciDAC repository.

MILC 6 Results

% Mem Ops.	52.4	**% L1 Hitrate**	96.94
% Load Ins.	98.8	**% L2 Hitrate**	97.27
% Store Ins.	1.2	**% FPU Idle Cyc**	7.35
% Branch Ins.	3.0	**IPC**	0.52
% Packed SSE(2)	80.4	**FLOPS/Mem Op.**	1.2
% Peak	16.4	**% Cyc. Dispatch Stalled**	78.52
MFLOPS/S	723.00	**% Cyc. Dispatch Stalled Mem.**	17.80
		% Stalls are Mem. Stalls	22.00

Figure 11. Per CPU Performance Data of MILC 6 on the AMD RDK "Quartet"

Figure 11 illustrates the performance characteristics of MILC 6 running on the AMD RDK "Quartet" system. It delivers over two-thirds of a gigaflop per second per CPU. The computational intensity is slightly higher than POP with 1.2 floating point operations per memory reference. Notice the figures for the percentages of load/store instructions. This measurement is actually taken from the level 1 miss counter, so the percentages are only indicative of those loads/stores that miss the level 1 cache. More than likely, the percentage of stores is higher, but those stores are hitting in level 1 and not being recorded. Here, like POP, the percentage of cycles where the dispatch unit is stalled waiting on memory is a relatively low 17%. This code performs nearly five times faster than POP, yet the dispatch stage stalls much more often. This is due to the fact that the processor is much more highly utilized, and as such, the dispatch unit stalls feeding the processor new instructions.

Conclusion

From our experiments, we can see that the AMD Opteron processor delivers as anticipated. By incorporating an on-board memory controller, the AMD Opteron processor delivers increased memory bandwidth and lower memory latency. In CacheBench and STREAM, the AMD Opteron processor-based system ("Quartet") delivers significantly more bandwidth than either the 2-way Intel Xeon-based system or the 4-way Itanium 2-based system when running out of cache. For data sizes in cache, the AMD Opteron processor performs very well, proportional to its clock rate. Regarding memory latency, the AMD RDK "Quartet" also shines, delivering measured latencies up to 50 percent lower than the other two platforms on the memory latency benchmark. This advantage is confirmed by the AMD system's performance on GUPS, where it performs nearly 20 percent faster than the competition [Ten percent relative to Xeon.] In examining real application performance, both POP 1.4 and MILC 6 deliver very good performance as a percentage of peak. This is in contrast to previous measurements [27][29] on other platforms where the percentage of peak obtained was in the single digits. Stall time measurements indicate a relatively low percentage of cycles being lost on waiting for memory, a critical metric when examining applications with large data sets. The AMD Opteron processor's memory subsystem is built to directly address the needs of such applications. This subsystem, combined with a full 64-bit address space, relatively low power consumption, and a high clock rate make the AMD Opteron processor and systems like the AMD RDK "Quartet", based on the AMD Opteron processor an ideal choice for a high-end, multiprocessor, technical compute server.

References

[1] J. von Neumann. “On the principles of large scale computing machines.” In Taub, A. H., editor, *John* von Neumann *Collected Works*, The Macmillan Co., New York, Volume V, 1-32.

[2] J. Backus. “Can Programming be Liberated from the von Neumann Style?” *Communications of the ACM*, no. 8, vol. 21, August 1978.

[3] M. Scheffer. *Liberation from the Von-Neumann Bottleneck?* Eindhoven University of Technology, February 2001.

[4] D. Nelson. *Modelling and Simulation: The Good, the Bad and the Hopeful.* DOE Computational Science Graduate Fellowship Conference, http://www.krellinst.org/csgf/conf/2003/presentations/nelson.ppt, July 1998.

[5] A. Saulsbury, et al. “Missing the memory wall: The case for processor/memory integration.” In *Proc. of the 23rd Intl. Symp. of Computer Architecture*, May 1996, pp. 90-101.

[6] L. Gwennap. “New processor paradigm: V-IRAM.” *Microprocessor Report*, vol. 12, no. 3, March 1998, pp. 17-19.

[7] W. F. Lin et. al. ”Reducing DRAM Latencies with an Integrated Memory Hierarchy Design.” In *Proc. of the 7th Intl. Symp. on High Performance Computer Architecture*, January 2001, pp. 301-312.

[8] C. Yelick et. al. “Memory-Intensive Benchmarks: IRAM vs. Cache-based Machines.” *Proc. of the Intl. Parallel and Distributed Processing Symp.,* April 2002.

[9] B. Jacob et. al. “Concurrency, Latency or System Overhead: Which Has the Largest Impact on Uniprocessor DRAM-System Performance?” P*roc. of the 28th Intl. Symp. on Computer Architecture*, June 30-July 4, 2001.

[10] V. Cuppu et. al. “A Performance Comparison of Contemporary DRAM Architectures.” In *Proc. of the 26th Intl. Symp. on Computer Architecture*, May 2-4, 1999.

[11] D. Burger et. al. “Memory bandwidth limitations of future microprocessors.” In *Proc. of the 23rd Intl. Symp. on Computer Architecture*, May 1996, pp. 78-89.
W. Wulf. et. al. “Hitting the Memory Wall: Implications of the Obvious.” *Computer Architecture News*, March 1995, pp. 20-24.

[12] J. McCalpin. *STREAM: Sustainable Memory Bandwidth in High Performance Computers.* http://www.cs.virginia.edu/stream.

[13] S. Toledo. “Improving the memory-system performance of sparse-matrix vector multiplication.” *IBM Journal of Research and Development*, vol. 41, no. 6, 1997.

[14] R. Whaley et. al. *Automatically Tuned Linear Algebra Software.* Technical Report UT-CS-97-366, http://www.netlib.org/lapack/lawnspdf/lawn131.pdf, 1997.

[15] *Samsung DDR400*.
http://www.samsung.com/Products/Semiconductor/DRAM/ddrsdram/DDR400_333.htm

[16] S. Woo. *High Performance PC Memory Systems.* Rambus Developers Forum, http://www.rambus.co.jp/events/Tech2-4_Rambus_Steven.pdf, July 2003.

[17] Infineon Technologies North America Corp. et. al. *Intel Dual-Channel DDR Memory Architecture White Paper.* http://www.kingston.com/newtech/MKF_520DDRwhitepaper.pdf, September 2003.

[18] B. Jacob et. al. *DRAM Memory System: Lecture 16*. University of Maryland, http://www.ece.umd.edu/courses/enee759h.S2003/lectures/Lecture16.pdf

[19] D. O'flaherty. et. al. *AMD Opteron Processor Benchmarking for Clustered Systems*. http://www.sun.com/amd/39497A_HPC_WhitePaper_2xCli.pdf, July 2003.

[20] D. Rintala. *Benefits of the AMD Opteron Processor for LS-DYNA.* European LS-DYNA Users Conference, http://www.dynamore.de/download/eu03/papers/K-I/LS-DYNA_ULM_K-I-43.pdf, May 2003.

[21] R. Brunner. *Building Blocks for 64-bit Opteron Processor-Based Clusters.* ClusterWorld, http://www.amd.com/us-en/assets/content_type/DownloadableAssets/RichBrunnerClusterWorldpresFINAL.pdf, June 2003.

[22] P. Mucci et. al. *The CacheBench Report.* Technical Report UT-CS-98-25, http://www.cs.utk.edu/~mucci/latest/pubs/mucci98cachebench.pdf, July 1998.

[23] P. Mucci. *LLCbench Home Page.* http://icl.cs.utk.edu/projects/llcbench

[24] L. McVoy et. al. "lmbench: Portable tools for performance analysis." In *Proc. of the USENIX 1996 Technical Conference*, January 1996, pp. 279-294.

[25] L. McVoy. *Lmbench – Tools for Performance Analysis.* http://www.bitmover.com/lmbench

[26] B. Gaeke. *GUPS Sample Implementation*. http://www.dgate.org/~brg/files/dis/gups

[27] P. W. Jones et. al. "Practical performance portability in the Parallel Ocean Program (POP)." *Concurrency and Computation: Practice and Experience*, 2003.

[28] P*OP Home Page*. http://climate.lanl.gov/Models/POP

[29] S. Gottlieb. *Benchmarking and Tuning the MILC code on Clusters and Supercomputers.* Indiana University, http://www.physics.indiana.edu/~sg/lattice01.

[30] D. Holmgren et. al. *Performance of MILC Lattice QCD Code on Commodity Clusters.* Science on Cluster Computers, Fermi National Accelerator Laboratory, http://qcdhome.fnal.gov/badHonnef.pdf, August 2002.

[31] C. Mendes. *Experiments of Instrumenting MILC with SvPablo.* University of Illinois, http://bugle.cs.uiuc.edu/~cmendes/milc.

[32] M. Grannis. "Quantum Chromodynamics with MILC." *NPACI & SDSC Envision*, vol. 16, no. 1, http://www.npaci.edu/envision/v16.1/milc.html, January – March, 2000.

[33] *MILC Home Page.* http://physics.indiana.edu/~sg/milc.html

[34] B. Dipert. "The Slammin' Jammin' DRAM Scramble." *EDN*, January 2000, pp. 68-82.

[35] J. L. Hennessey et. al. *Computer Architecture: A Quantitative Approach,* 2nd Ed. Morgan

Kaufmann Publishers Inc., 1996.

[36] P. Machanick. “Approaches to Addressing the Memory Wall.” Technical Report, School of IT and Electrical Engineering, University of Queensland, December 2002

[37] L. A. Barroso, et al. “Memory system characterization of commercial workloads.” In *Proc. of the 25th Intl. Symp. of Computer Architecture*, June 1998, pp. 3-14.

[38] D. Burger. *Hardware Techniques to Improve the Performance of the Processor/Memory Interface.* PhD. Dissertation, University of Wisconsin, 1998.

[39] S. McKee et. al. “Increasing Memory Bandwidth for Vector Computations.” In *Proc. Of Europar'95*, August 1995.

[40] P. Mucci et. al. “The PAPI Cross-Platform Interface to Hardware Performance Counters.” In *Proc. of the Department of Defense Users' Group Conference*, http://www.cs.utk.edu/~mucci/latest/pubs/dodugc99-papi.pdf, June 2001.

[41] D. A. Patterson et. al. *Computer Organization and Design,* 2nd Ed. Morgan Kaufmann Publishers Inc., 1997.

[42] V. Karamcheti et. al. “Software Overhead in Messaging Layers: Where Does the Time Go?” In *Proc. Of ASPLOS-VI*, October 1994.

[43] J. Ousterhout “Why Aren't Operating Systems Getting Faster as Fast as Hardware?” In *Proc. Of USENIX Summer Conference*, June 1990.

[44] P. Mucci. “DynaProf: A Live Tutorial on the IBM Power 3 at NERSC.” *Supercomputing 2003,* http://www.cs.utk.edu/~mucci/latest/pubs/PERC-SC2003-Session-III.pdf, November 2003.

[45] N. Mahapatra et. al. “The Processor-Memory bottleneck: Problems and Solutions.” *ACM Crossroads*, Spring 1999.

[46] D. Patterson et. al. “A Case for Intelligent RAM: IRAM.” *IEEE Micro*, April 1997.