Performance of the IBM *e*server® 325 for Scientific and Technical Applications

Douglas M. Pase, Ph.D.
IBM eServer xSeries Performance Development & Analysis
Research Triangle Park, North Carolina, USA

Abstract

The IBM® eServer™ 325, or e325, is a powerful server designed for high performance technical and commercial computing [1]. The e325 is based on the AMD Opteron™ processor [2]. It incorporates advanced processor and memory features that provide high performance at low cost. In this paper we examine key features of the e325 and show how they affect performance under scientific and technical workloads. We examine memory and processor performance, and the effects of both Symmetric Multiprocessing (SMP) and Non-Uniform Memory Access (NUMA) kernels on performance.

The focus of this paper is floating-point and memory-intensive workloads such as those used in the scientific and technical community. For this reason we examine performance on standard benchmarks. Benchmark suites used for comparison are STREAM [3], Linpack [4,5,6], and SPEC CPU2000 [7].[1]

Introduction

The IBM eServer 325, or e325, is a rack-optimized 1U system based on the AMD Opteron processor. (A "U" is a measurement of height, 1.75 inches, used to separate sets of bolts in a standard electronic equipment rack.) The e325 is optimized for scientific and technical workloads requiring efficient floating-point processing and high bandwidth to memory. The e325 is ideally suited for use in high-performance clusters. It supports one or two Opteron processors and up to 12GB of DDR333 memory. Disk subsystems may be either IDE or hot-swap SCSI.

Scientific workloads may be characterized generally as heavily numeric in nature, often dominated by 64-bit floating-point operations. Many of the codes were originally developed on vector machines like those produced by Cray Research, Fujitsu and NEC. Vector machines have high memory bandwidth and provide large performance gains to applications that apply the same operation to large sections of memory using regular strides. This is very different from cache-based scalar microprocessors, which reward possibly varying operations applied to the same data over and over again. To make things more difficult for the system designer, many key vector applications have been converted to run well on cache-based systems because of their attractive pricing. This means that, for a system to do well on scientific workloads, it must have a fast processor, high bandwidth to memory, a large cache and an affordable price.

[1] Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Actual results may vary. Users of this presentation should verify the applicable data for their specific environment.

Unfortunately, there is often a significant gap between the hardware peak memory performance and the memory performance available to an application. The STREAM benchmark was created to measure available memory bandwidth. STREAM is a simple benchmark of four loops. The first loop, COPY, copies data from one vector into another; the second, SCALE, multiplies a vector by a scalar value; the third loop, ADD, adds two vectors; and the fourth loop, TRIAD, combines a scale with an addition operation. This last loop is also known as a DAXPY operation (Double-precision A times X Plus Y). The arrays used in these four loops are required to be much larger than the largest processor cache, so operands are always retrieved from memory. All operations use 64-bit operands and memory accesses are stride-one (i.e., a[i], a[i+1], a[i+2], ...).

In contrast to STREAM, Linpack focuses primarily on processor Arithmetic Logic Unit (ALU) performance rather than memory performance. Linpack is a collection of subroutines that analyze and solve linear equations and linear least-squares problems. Originally designed for supercomputers in the 1970s and early 1980s, Linpack solves linear systems whose matrices are general, banded, symmetric indefinite, symmetric positive definite, triangular, and tridiagonal square. Linpack is built upon the Basic Linear Algebra Subroutine package, or BLAS. The Linpack benchmark uses the Linpack library to solve a general dense system of linear equations using LU Decomposition [8]. The Linpack library has largely been replaced by Lapack, which extends and improves upon the routines [9]. The routines have been carefully rewritten and tuned to take advantage of processor cache, and relatively few references actually go to memory. For our tests, we use the standard 1000 x 1000 double-precision Linpack DP benchmark. Our benchmark implementation uses the high-performance BLAS created by Kazushige Goto [10].

STREAM and Linpack are both kernel benchmarks; that is, they focus on measuring the performance of a small, relatively simple mathematical kernel. In so doing they give an accurate measure of the performance of a single subsystem of the computer. To avoid the bias inherent in kernel benchmarks, we also measure system performance using a more realistic benchmark, SPEC CPU2000. This benchmark is actually two suites of applications. One suite, SPEC CINT2000, consists of highly cacheable integer applications. The other, SPEC CFP2000, contains memory-intensive, floating-point applications. SPEC CPU2000 allows the suites to be run on a single processor to measure processor speed, or to be run on one or more processors to measure system throughput rates. In our tests we use only the throughput measurements because that represents the environment in which the e325 is most likely to be used.

Processor Architecture

The IBM e325 owes much of its performance to the novel architecture of the Opteron processor. Opteron enjoys two major advantages over its x86 cousins, whether from Intel®, AMD or elsewhere. First, it natively supports the x86 instruction set and the x86-64 extensions. That allows Opteron to execute all x86 applications at full speed. No translation or emulation layer is required to execute x86 programs. The x86-64 extensions provide 64-bit virtual addresses and an extended register set, so 64-bit programs can also be executed on the same machine. Both Linux and Microsoft® Windows® allow both 32-bit and 64-bit applications to be executed at the same time with no difficulty whatsoever. An application can be compiled for 64-bit mode, that is, to use the larger addresses and additional registers, with little more effort than to change a flag on the compile line. The larger address space allows an application to address more than 3GB of memory without resorting to overlays, or interfaces such as PAE36 that are slow to execute and difficult to use. The larger register set allows a compiler to reduce the amount of memory traffic by saving more intermediate values into registers, when appropriate. The x86 and x86-64 registers are illustrated in Figure 1.

Because “64-bit” is such a popular term, it is important to be clear on what it means. For purposes of this document, it means the architecture supports 64-bit virtual addresses. 64-bit integers and 64-bit floating-point operands are supported by x86 and other architectures that are considered 32-bit architectures. Such processors may also support physical addresses larger than 32-bits.

The size of the addresses used by a user application determines whether an architecture, or an application, is 32- or 64-bit.

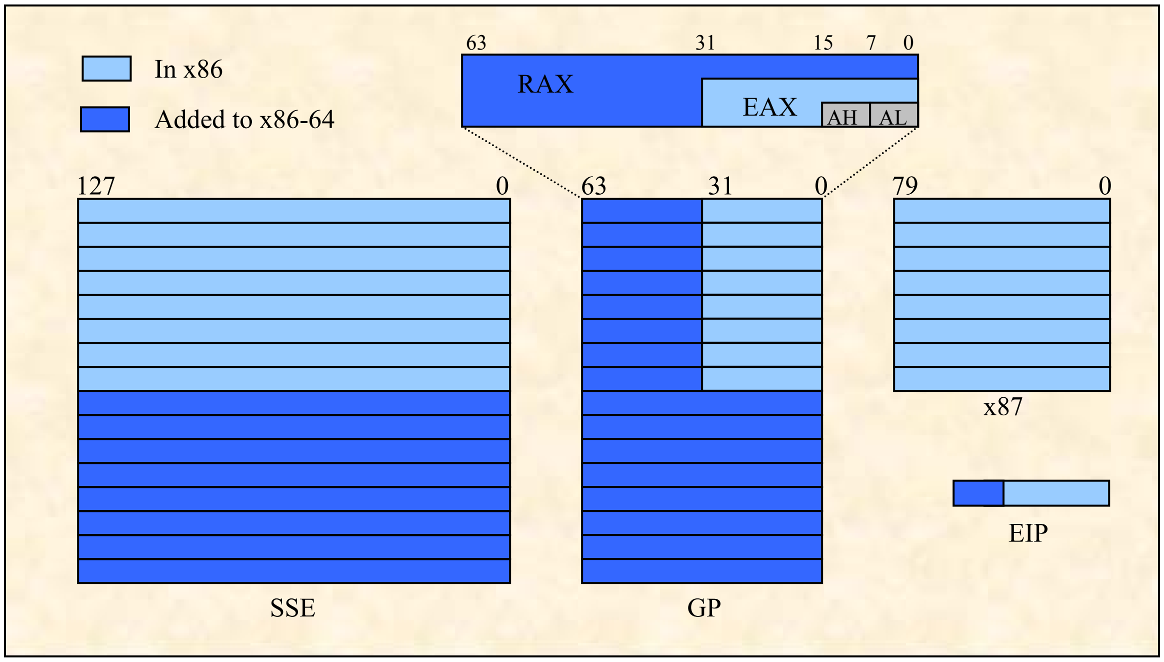

Figure 1. x86 and x86-64 Registers

Opteron's second advantage over its x86 cousins is that the memory controller is integrated into the processor. This is an advantage for two reasons. First, the memory controller is clocked at the same rate as the processor. So, as the processor speed is increased, the memory controller speed is also increased, reducing the latency through the memory controller, allowing faster access to memory. Second, when a processor is added to a system, more paths to memory are also added. As the demand for memory bandwidth increases due to the additional processor, more bandwidth is available to satisfy that demand.

To see this more clearly, let's consider a traditional architecture that uses a shared Front-Side Bus (FSB), as illustrated in Figure 2. It contains two processors connected by a shared FSB. Each processor communicates with the FSB through its Bus Interface Unit (BIU). The memory controller is an additional component connected to the FSB. The memory controller may have one or several channels to memory. In this diagram there are two channels. The memory controller also routes commands and status between the I/O bus and processors, and data between the I/O bus and memory.

There are several items worthy of note in this diagram. The most obvious is that there is a single resource shared between the two processors, that is, the FSB. The speed of the FSB places an upper bound on the rate at which a processor can send data to or receive data from memory. In fact, FSB bandwidth is often tuned to match the bandwidth of available memory technology at the time. It is expected that a single processor will **not** saturate the FSB over time because the processor has a cache where data most likely to be referenced are stored. Cache reduces the FSB pressure, so there is capacity to allow more than one processor to operate on the FSB.

Nothing in this abstract architecture inherently limits the number of memory channels, or the size of cache, or the speed of the FSB. In the real world, the limit is its cost. For a bus to be faster, it must be wider or it must be clocked at a faster rate. To make a cache larger requires more transistors (i.e., more silicon). All of this adds cost to the processor or system. Common practice today is to match the FSB bandwidth to the memory bus bandwidth. It is a cost-effective solution that, nevertheless, impacts performance on data-intensive applications.

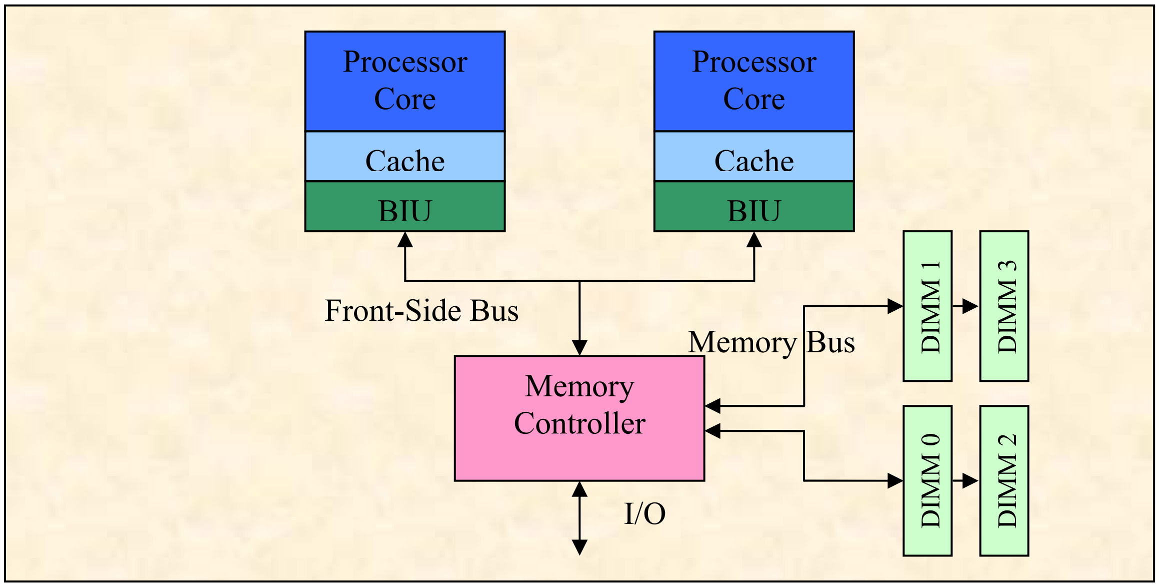

Figure 2. Shared Front-Side Bus Architecture

A typical Intel Xeon™ processor-based system available today has a Front-Side Bus that is eight bytes wide and clocked at 533MHz. Its memory controller has two channels, each 8 bytes wide, to DDR266 memory. This gives the FSB and the memory bus 4.3 GB/s of bandwidth each. Emerging Xeon processor-based systems have an 800MHz FSB and use DDR400 memory, for a bandwidth of 6.4 GB/s.

Now consider the Opteron architecture as shown in Figure 3. As in the previous diagram there is a processor core and cache. But in place of a bus interface unit and an external memory controller, there is an integrated memory controller (MCT), an interface to the processor core (SRQ), three Coherent HyperTransport (cHT) units and a cross bar switch to handle routing of data, commands and addresses between them. The memory controller supports up to two 8-byte channels to memory. The processor is able to support up to 400MHz (DDR-I 400) registered Error Correcting Code (ECC) memory. At 333MHz (DDR-I 333), the memory bandwidth is 5.3 GB/s across both channels; at 400MHz it is 6.4 GB/s. The cHT units may be used to connect to I/O devices or to other processors. The protocol used for routing memory traffic is somewhat more elaborate than what is used for I/O, but the I/O protocol is a proper subset so cHT links may be used for either purpose.

Note once again that every device within the processor package is clocked using a synchronized clock. As the processor clock is increased from one generation or processor speed bin to the next, the memory controller clock is automatically increased at the same rate. This has the advantage of decreasing the latency of a memory request from the processor core to the memory, which speeds access.

The cHT links are point-to-point connections. Links are never shared. A processor can be directly connected to only as many processors as it has available links. In theory it can be indirectly connected to an arbitrary number of processors in a ring, or tree or other topology, but in practice the number of processors in a system is limited.

Each link consists of two unidirectional paths, each 2 bytes wide, with a small number of control lines for sending commands and status. Coherent HyperTransport links today operate at 800MHz, with speeds increasing to 1GHz and beyond in the future. Data transmissions are "double pumped"; that is, data are transmitted on both the rising edge and on the falling edge of the clock.

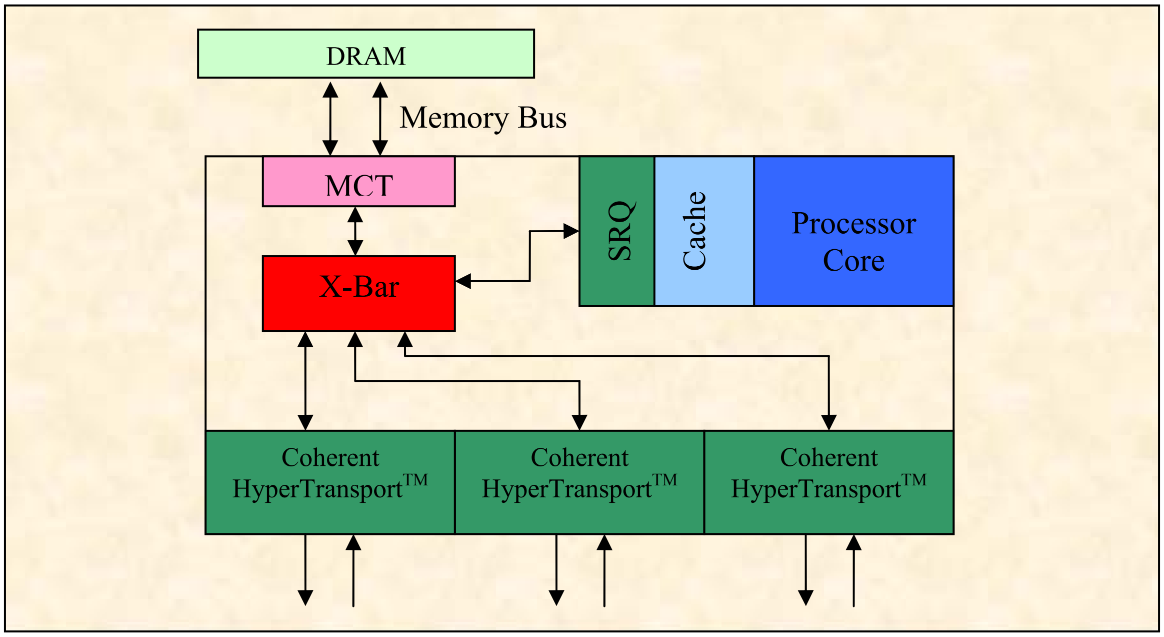

Figure 3. Opteron Processor Block Diagram

It is this combination of integrated memory controller and point-to-point links that is one of Opteron's strengths. Because of this, adding more processors to a system automatically adds memory bandwidth. Thus there is less risk of oversubscribing a shared resource like the Front-Side Bus of the previous architecture.

System Architecture

The e325 is a 1U dual-processor Opteron system. It is rack-optimized and designed to be used as a computational node in a scientific cluster. It supports up to 12GB of main physical memory in six DIMM slots, two IDE or two hot-swap SCSI drives, two Gigabit Ethernet (GbE) ports, and two 100MHz PCI-X slots. Processor speeds currently range from 1.4GHz to 2.2GHz, and the system supports memory speeds of DDR266 and DDR333. Figure 4 is a block diagram of the system.

The system must be loaded with one, two or four DIMMs on processor A, and may have any of zero, one or two DIMMs on processor B. All memory must be of the same speed, that is, DDR266 or DDR333. Pairs of DIMMs – DIMMs 0 and 1, 2 and 3, or 4 and 5 – must also have the same size. So, one operational configuration would be to load four 512MB DIMMs into slots 0 through 3, and two 1GB DIMMs into slots 4 and 5. This configuration gives 2GB per processor for a total of 4GB in the system. This particular configuration is desirable for two reasons: memory is balanced across the processors, and there is more than 1GB on processor A. (The former should be intuitive; the latter will be explained later.)

Memory Performance by Channels

It was mentioned in the previous section that the e325 supports two channels to memory per processor. As might be expected, two channels give better performance than one. The extent to which two-channel performance is less than twice that of a single channel is an indication of conflicts within the system, most likely involving the memory controller. Ideally, it is desirable to see very few conflicts when both channels are operating at full load.

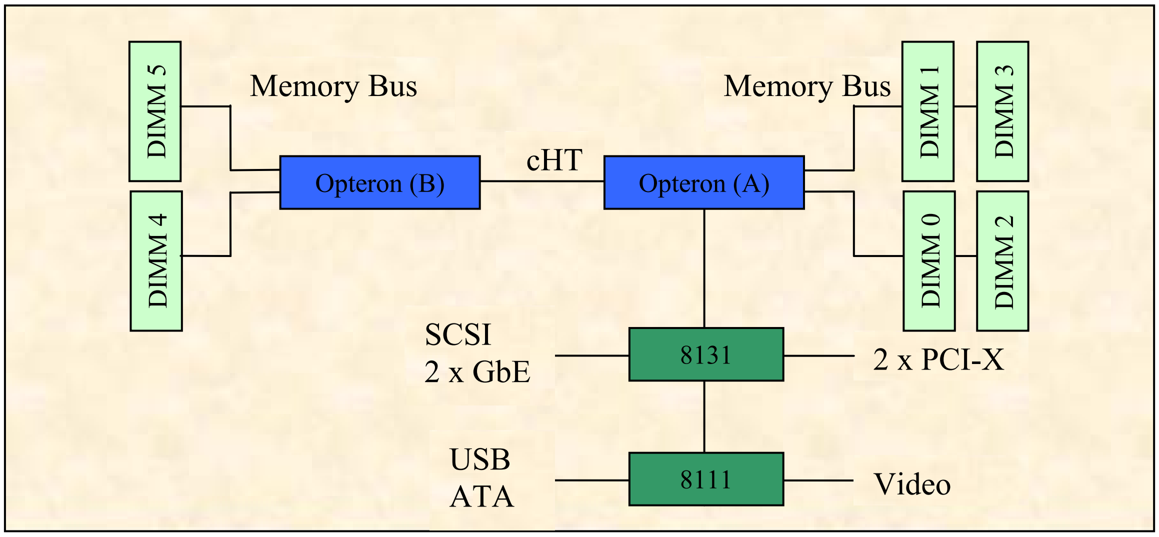

Figure 4. IBM eServer 325 System Architecture

To determine the performance loss due to internal conflicts, we performed the following experiment. STREAM was run on a 2.0GHz system populated with DDR333 memory. Several experiments were run using only a single thread. The first used a single processor with a single channel populated. The next experiment was similar, but populated both channels with memory. A third experiment used two processors, one with two channels populated, the other with only one channel populated. Additional experiments used two threads, with one channel populated on each processor, then both channels, then one channel on one processor, and two channels on the other. The experiments were repeated using DDR266 memory. The results are shown in Figure 5. Single-thread results are labeled “1T”; dual-thread results are labeled “2T.”

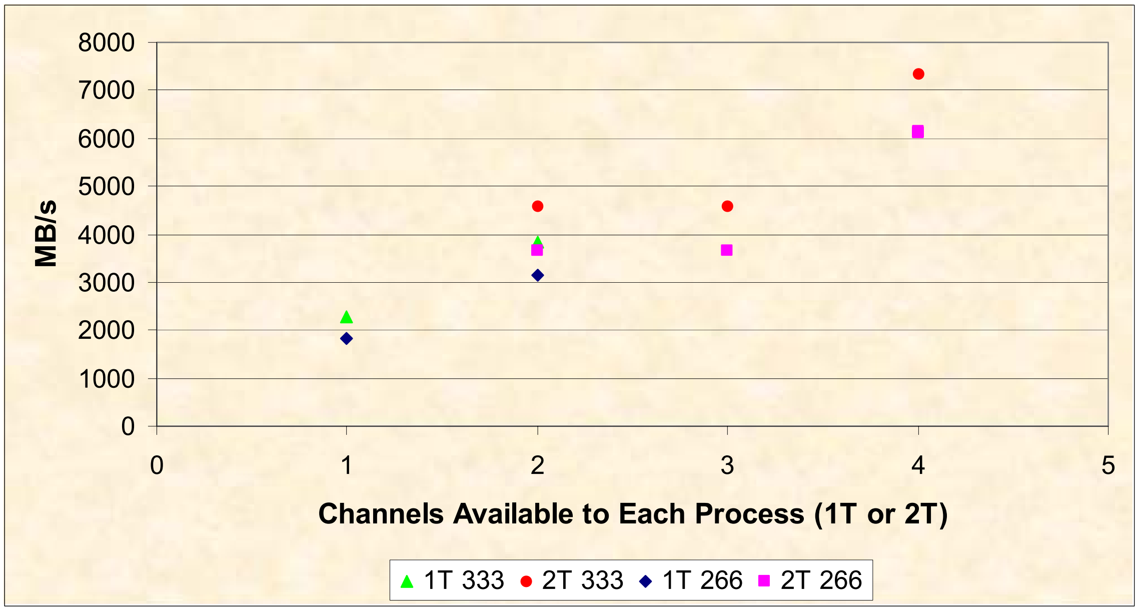

Figure 5. Memory Performance of STREAM Triad for a 2.0GHz e325 by Memory Channels

The best results were obtained using two threads to drive all four channels. The performance was about 7.3 GB/s. Each of the four channels is capable of up to 2.67 GB/s, so this represents about 69% of the hardware peak of 10.6 GB/s. When each thread had only a single memory channel, the system performed at a little less than 4.6 GB/s, or 86% of its 5.3 GB/s hardware peak.

Similar, but slightly better, results are obtained using DDR266 memory. This is the expected result since the slower memory gives the memory controller more time to clear its conflicts.

Memory Performance by Memory Frequency

All DIMMs are 8 bytes wide. Double Data Rate, or DDR, memory drives its address and command lines at its native frequency. However, it drives the data lines at twice its native frequency, hence the name Double Data Rate. The data rate is indicated by its name. For example, DDR333 has a native clock speed of 166MHz and a data rate of 333MHz.

It is interesting to consider how the memory performance changes as the memory speed changes. A simple calculation might lead to the expectation that DDR333 would be 25% faster than DDR266; 333MHz is, after all, 25% faster than 266MHz. In fact, DDR333 is 25% faster than DDR266 only when the channels are lightly loaded, that is, when only one channel per processor is in use. This can be seen in Figure 6.

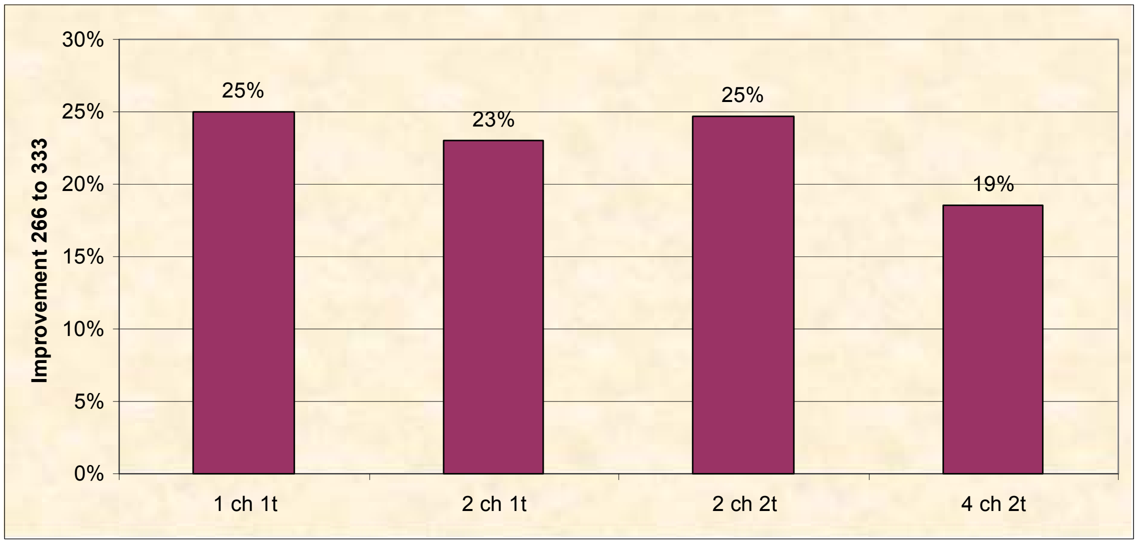

Figure 6. Memory Performance Improvement of a 2.0GHz e325 on STREAM Triad

When each thread uses only one channel, memory commands are processed and passed through the memory controller at its most rapid rate. There are plenty of resources within the controller to handle DDR333 memory speeds. When two threads each use a single channel, the cHT links become involved, but the loss is very slight. When a single thread uses two channels on the same processor, again, the efficiency is only slightly below that of DDR266 memory. The performance loss is compounded as all channels become active. The total performance is greatest, but the efficiency is lower for DDR333 memory than for DDR266. Memory performance improves only by 19% instead of the full 25%.

This result has significance for understanding how Opteron will behave with DDR400 memory. The normal mode of operation is to use both channels on both processors. It gives much better total performance than using fewer channels. But the efficiency declines as the memory gets faster. There is a slight decline when the HyperTransport is used (the “2 channel, 2 thread” case) and a marginally greater decline when a single controller is saturated (the “2 channel, 1 thread” case). The decline increases sharply when both are combined, as in the “4 channel, 2 thread” case, indicating a possible compounding effect.

This suggests less than perfect scaling going from DDR333 to DDR400 on 2.0GHz Opteron processors. However, the scaling should improve as the processor clock is increased above 2.0GHz, as the next section will discuss.

Memory Performance by Processor Frequency

In this section, we quantify the contribution of processor frequency to memory performance. In a previous section, we mentioned that the Opteron processor has an integrated memory controller, and that the controller is clocked in sync with the processor. As the processor clock speed increases, the clock rate of the memory controller also increases. As the memory controller clock speed increases, the path through the memory controller takes less time and memory performance increases. Memory latency becomes shorter, and memory throughput increases as Figure 7 shows.

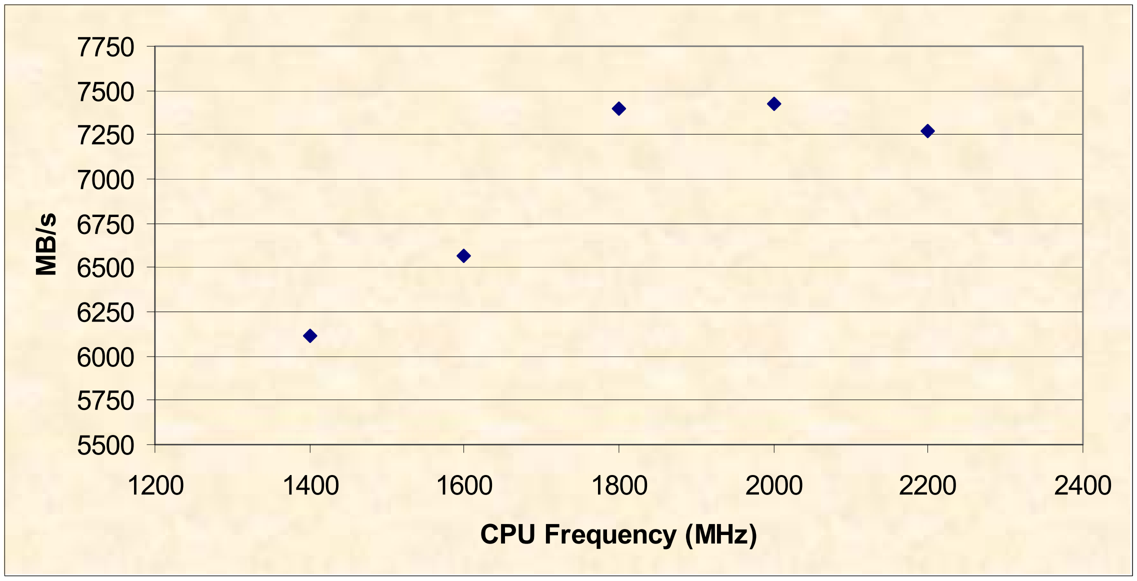

Figure 7. STREAM Triad Performance by Processor Frequency

Figure 7 shows the memory performance for processor frequencies ranging from 1.4GHz through 2.2GHz. Throughput increases in roughly a straight line from 1.4GHz to 2.0GHz. Over this range, the frequency increases by 600MHz, and the throughput increases by just over 1,200MB/s. That gives approximately 2MB/s throughput improvement for each 1MHz processor clock improvement.

Notice, however, there is a dip from the 2.0GHz processor to the 2.2GHz processor. The explanation for the dip is subtle. The memory interface has a synchronous design. In other words, to reduce the design complexity and to increase performance, the clock driving the memory must be synchronous with the processor clock. Since the memory clock (e.g., 166MHz for DDR333) is much slower than the processor clock (e.g., 2.2GHz), one or both of them must be changed until the slower clock is a multiple of the faster. AMD decided to slow the memory frequency until it evenly divides the processor clock.

Dividing 166MHz into 2,000MHz yields an even 12, but 166MHz does not evenly divide 2,200MHz. To find a divisor of 2,200MHz, the memory clock must be slowed to 157MHz, which is 6% slower than its native speed of 166MHz. But our measured score is only 1% slower, not 6% slower. Where is the difference? The 2.2GHz processor clock is 10% faster than the 2.0GHz clock. If memory were able to operate at full speed, this would contribute about 5% to the memory performance. But about 6% is lost because the memory cannot operate at full speed, hence, the 1% performance loss we measured.

Notice that the fact that processor clocks are not even multiples of 166MHz causes a ratchet effect in memory performance. At 1.4GHz, the data rate of DDR333 must be slowed to 311MHz. At 1.6GHz, it must be slowed to 320MHz. At 1.8GHz, it must be slowed to 327MHz, and at 2.0GHz, it operates at a full 333MHz speed. But at 2.2GHz it drops down to 314MHz and begins the climb again.

While this ratchet effect occurs with DDR266 and DDR333 memory, it does not occur with DDR400. The native speed of DDR400 is 200MHz, and Opteron clock frequencies are always integer multiples of 200MHz. Thus, there is no loss due to synchronizing memory to processor.

Memory Performance by DIMMs per Channel

The next analysis concerns the amount of parallelism available within a memory channel. The Opteron processor uses registered ECC memory. Registered memory places a latch, or a register, between DRAM and the memory bus. This allows communication to take place on the memory bus while the DRAM inside the DIMM is executing some other operation, such as a read or a write. Some DIMMs are also divided into banks, usually four banks to a DIMM, and operations on one bank can occur independently of operations on another bank.

It is worth noting that a BIOS setting may also play a role here. Under *Advanced Settings* in the BIOS, there is an entry called *DRAM Interleave*, which should always be set to *AUTO*. This field allows the user to select whether cache lines are stored within a single bank (i.e., disabled) or each line is striped across all banks. The advantage to striping is that every fetch and store of a cache line draws upon the full parallelism available within the DIMM. We have not yet found any disadvantage to striping cache lines in this way.

In this comparison we run STREAM Triad on a single CPU using two DIMMs (one per channel), then again with four DIMMs (two per channel), to see how performance improves. The results, shown in Figure 8, are broken out by memory speed and size. It was expected that the differences would be small, less than 2%, which turned out to be the case. What was not expected was that the advantage decreased with memory size, and became slightly negative (0.25%) for 1GB DDR333 DIMMs. However, this last result is so small that it may be an anomaly.

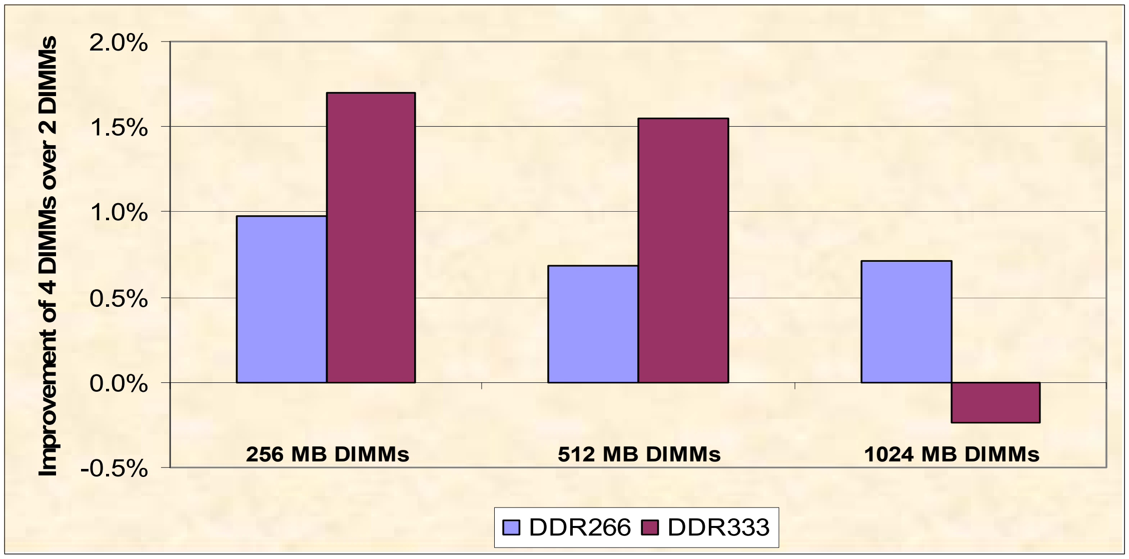

Figure 8. Parallelism between DIMMs on a 2.0GHz e325 (STREAM Triad)

As part of this same analysis, we examined whether placement of DIMMs had any impact on performance. We found no difference whatsoever. We placed two DIMMs in slots 0 and 1, and ran the results. We then placed the same two DIMMs in slots 2 and 3, and compared the results to the former runs. We tried this for all DIMM sizes and speeds. We found no difference in any of our experiments.

Operating System Performance

We mentioned in an earlier section that the e325 has a NUMA design. To take advantage of this design, the operating system must be aware of the design and respond appropriately. In general there are three challenges associated with NUMA-aware operating systems. First, a thread should have much stronger affinity to a processor than is required for symmetric multiprocessing (SMP) systems. Second, physical memory should be allocated close to the thread. Third, I/O buffering must be done to minimize data traffic through the system. This last requirement is not a problem with the e325 because all I/O devices are attached to the processor that owns low memory, which is where I/O buffers are traditionally placed.

In an SMP system, when a thread migrates from one processor to another, at most, the contents of the processor cache may have to be reloaded from memory. If a thread is active, that may be an unnecessary cost that should be incurred infrequently or not at all. If the thread has been asleep for a while, perhaps while waiting for I/O, the cache contents have been flushed anyway and any choice of processor is as good as the next.

When scheduling threads to processors in a NUMA system, the cost of accessing memory must be accounted for as well as the cost of reloading cache. In theory this can be a difficult problem, balancing processor load against the distance to memory. In practice it is often the case that a thread is simply locked to a processor and never migrated. This solves the processor and memory affinity issues together. And while inefficiencies can still result, it is often better than any of the alternatives.

A NUMA system must allocate physical memory close to the processor where the thread is running. By definition, in a NUMA system some physical memory is near and some is far, and there are performance advantages to using near memory. The operating system must decide where to allocate physical memory. To do this the operating system must have a mechanism for recognizing how physical memory is mapped to processors and for allocating memory appropriately when requested.

The mechanism for mapping physical memory to processors is found in the BIOS. It is called the *ACPI Static Resource Allocation Table*, or SRAT. It contains entries that describe the mapping of physical memory address ranges to processors. It must be enabled or the operating system does not know what memory is local to a processor and what memory is remote.

A related concern that arises in NUMA operating systems is that memory allocation for the kernel and for user space have different needs. User threads need to be assigned to a processor and rarely, if ever, migrated to another processor. Kernel threads may be invoked at any time from either processor, and thus no assignment of data to memory can always be local. Unfortunately, we have observed that some NUMA kernels boot only on a single processor. By itself this is not a problem, except that they use the same local memory allocation scheme as is used for allocating physical memory to user threads. This has the effect of allocating all kernel physical memory on the boot processor, or CPU A. For this reason it is desirable to have at least 2GB of memory on CPU A so that there is room for user threads in addition to the kernel.

Given the complexity of NUMA kernels and the many opportunities to get important details wrong, we wanted to understand which kernels performed well, which performed poorly, and how much difference there was between them. We ran our tests on eleven different Linux kernels, nine from SUSE LINUX and two from Red Hat. The SUSE LINUX set includes kernels from the SLES 8 standard release, SLES 8 Service Packs 2 and 3, and 9.0 Professional. The Red Hat kernels include AS 3.0 Gold and the AS 3.0 NUMA update. The results are presented in Figure 9 and Tables 1 and 2.

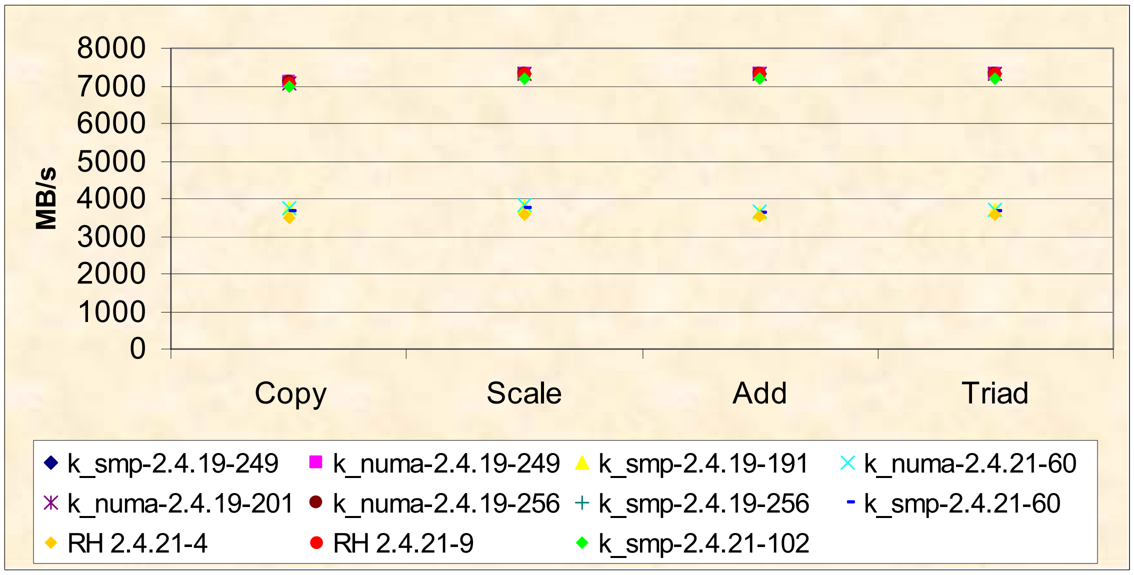

Figure 9. Linux Kernel Performance on a 2.0GHz e325

Figure 9 demonstrates that the kernels we tested fall into two very clearly differentiated groups. There is very little performance difference between kernels within each group, but there is a noticeable difference in performance between groups. The higher-performing group is aware of and takes advantage of the system's NUMA characteristics. The lower-performing group does not, and, in fact, includes several that are true SMP kernels.

It is difficult to see from Figure 9 which kernels performed well and which did not, simply because the data is grouped so tightly. For that reason we have included Tables 1 and 2. Table 1 shows the kernels we tested that behaved like NUMA-enabled kernels; Table 2 shows those with SMP behavior. Note that some kernels labeled "SMP" perform well, and some that are labeled "NUMA" perform poorly. Names can be misleading. In these tables, *Max* is the best result obtained out of 10 trials. *Spread* is the relative gain of the best result when compared to the worst, that is, (*max* – *min*) / *min*.

		Copy		Scale		Add		Triad	
		Max	Spread	Max	Spread	Max	Spread	Max	Spread
Red Hat	2.4.21-9.ELsmp	7077	2.4%	7310	2.6%	7332	2.7%	7331	2.6%
SUSE LINUX	k_numa-2.4.19-201	7080	0.1%	7314	0.1%	7324	0.1%	7329	0.2%
	k_numa-2.4.19-249	7090	0.1%	7313	0.1%	7319	0.1%	7320	0.1%
	k_numa-2.4.19-256	7090	0.2%	7313	0.2%	7321	0.2%	7319	0.1%
	k_smp-2.4.19-249	7080	0.1%	7316	0.1%	7324	0.1%	7328	0.2%
	k_smp-2.4.19-256	7084	0.1%	7311	0.2%	7320	0.1%	7317	0.1%
	k_smp-2.4.21-102	6975	0.1%	7196	0.1%	7194	0.1%	7198	0.1%

Table 1. NUMA-Enabled Kernels

It is difficult to differentiate performance between NUMA kernels using only this test. The most important information in this table, however, is what kernels are included in the set. A person who prefers SUSE LINUX might choose k_smp-2.4.19-249 from Service Pack 2. (Of the SUSE LINUX kernels, this is the one that we prefer.) A person who prefers Red Hat might choose 2.4.21-9.ELsmp. Both perform well and memory performance doesn't appear to be a distinction between them.

Table 2 shows how SMP kernels behave on this system. A somewhat greater performance differentiation exists between SMP kernels than between NUMA-aware kernels, but not a lot. There is a clear difference between the SMP kernels and the NUMA kernels. So, if performance is a consideration, a NUMA kernel should be chosen over an SMP kernel for use on this system.

		Copy		Scale		Add		Triad	
		Max	Spread	Max	Spread	Max	Spread	Max	Spread
Red Hat	2.4.21-4.ELsmp	3508	3.9%	3564	2.1%	3532	0.3%	3593	1.8%
SUSE LINUX	k_numa-2.4.21-60	3728	9.0%	3814	6.4%	3646	2.2%	3694	2.2%
	k_smp-2.4.19-191	3725	9.4%	3798	6.8%	3611	4.8%	3690	3.0%
	k_smp-2.4.21-60	3679	8.4%	3759	6.4%	3620	2.3%	3677	2.9%

Table 2. SMP-Enabled Kernels

The previous experiments ran STREAM 10 times on each kernel, with all channels populated and two threads active. The value selected for comparison was the best run from each set. The spread between the best and worst results is also given. This gives a sense of overall performance, but it does not give a good indication of how variable a kernel is at scheduling tasks. The data appeared to be consistent, but this was not tested rigorously.

To create a rigorous test, a single kernel, k_smp-2.4.19-249, was chosen. The system was set up so that two memory channels were populated on CPU A, and only one channel was populated on CPU B. STREAM was then run using only a single thread. The expectation was that if the kernel assigned the thread to CPU A and left it there, the performance would match the two-channel value. If it assigned the thread to CPU B and left it there, the performance would match the single-channel value. If the thread migrated between the two CPUs, that would indicate poor scheduling. The performance would fall somewhere between the one- and two-channel values if the memory were local to CPU A, and it would be worse than the single-channel results if memory were local to CPU B. This experiment would amplify any variability due to memory allocation or task scheduling.

The results of this experiment are shown in Figure 10. Out of 10 runs, the kernel scheduled half on CPU B (experiments 1 through 5) and half on CPU A (experiments 6 through 10). Those scheduled on CPU B show very little variation from the single-channel, single-processor result (left-most bar), whereas those scheduled on CPU A show very little variation from the dual-channel, single-processor result (right-most bar). This suggests that, at least in this kernel, scheduling and memory allocation are handled well.

Processor Performance

So far we have focused extensively on memory performance. Memory throughput determines how effectively data can be fed from memory to the processors. The fastest processor can operate only as quickly as it receives data to process. Scientific and technical applications are especially sensitive to memory performance. Next we examine the e325's processor performance. Specifically we examine processor frequency scaling on several processor-intensive workloads.

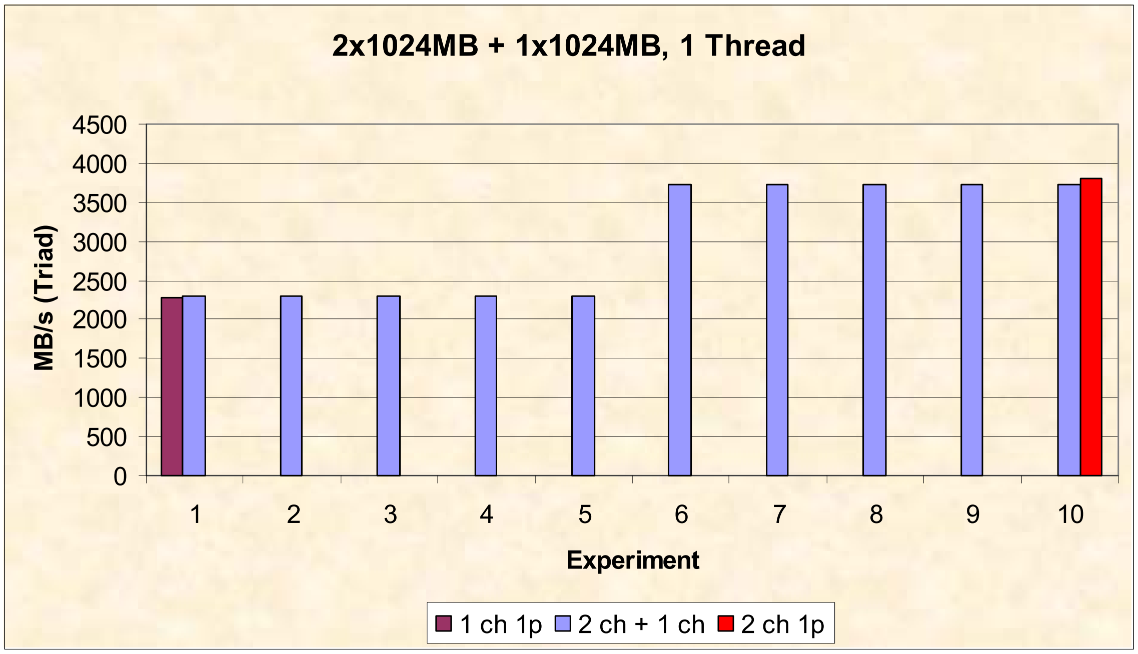

Figure 10. Quality of NUMA Schedule for SUSE LINUX Kernel k_smp-2.4.19-249

When we describe a workload as processor-intensive, we mean that, to a large degree, the processing speed of the Arithmetic Logic Unit (ALU) is the limiting factor to performance. The ALU is the core intelligence of the processor. It is the portion of the processor that does all of the integer and floating-point calculations. It computes where data is stored in memory, and sometimes, where a program should next begin executing.

Notice that a program may use 100% of the available CPU time and still not be ALU-intensive. STREAM is a good example of that. STREAM spends most of its time waiting for data to arrive from memory, so the ALU is mostly idle. A faster processor does not give a faster STREAM result unless it also gives faster access to memory. Network- and disk-bound benchmarks may also use 100% of the CPU without being ALU-bound. In general, I/O-bound benchmarks tend to be limited by memory performance (among other things) just as STREAM is. This happens because I/O devices and device drivers spend much of their time copying data from one location in memory to another.

The workloads we chose are each relevant to the scientific and technical community. We chose Linpack, SPEC CINT2000 Rate, and SPEC CFP2000 Rate. Aside from their general familiarity and relevance as benchmarks, they were selected because each exercises a different aspect of the system. Linpack exercises the floating-point capabilities of the processor almost exclusively. It retrieves most of its data from registers or from processor cache, and thus does not place heavy demands on the memory subsystem. SPEC CINT2000 Rate is a suite of desk top applications combined to form a composite benchmark. Like Linpack, it retrieves most of its data from cache. But, instead of floating-point processing, it makes heavy use of the processor integer units. SPEC CFP2000 Rate is also a suite of applications combined to form a single benchmark. However, it is important because it uses a realistic mix of integer, floating-point and memory operations in its execution.

We ran Linpack 10 times on a single processor using a single-threaded version of the library, and selected the best results out of the 10 runs. The problem size was the standard 1000 x 1000 elements, but the calculation was repeated enough times within a timing interval so that we would not have invalid measurements resulting from the granularity of our clock. We then ran the problem again on both processors using a parallel version of the code. The results are shown in Figure 11.

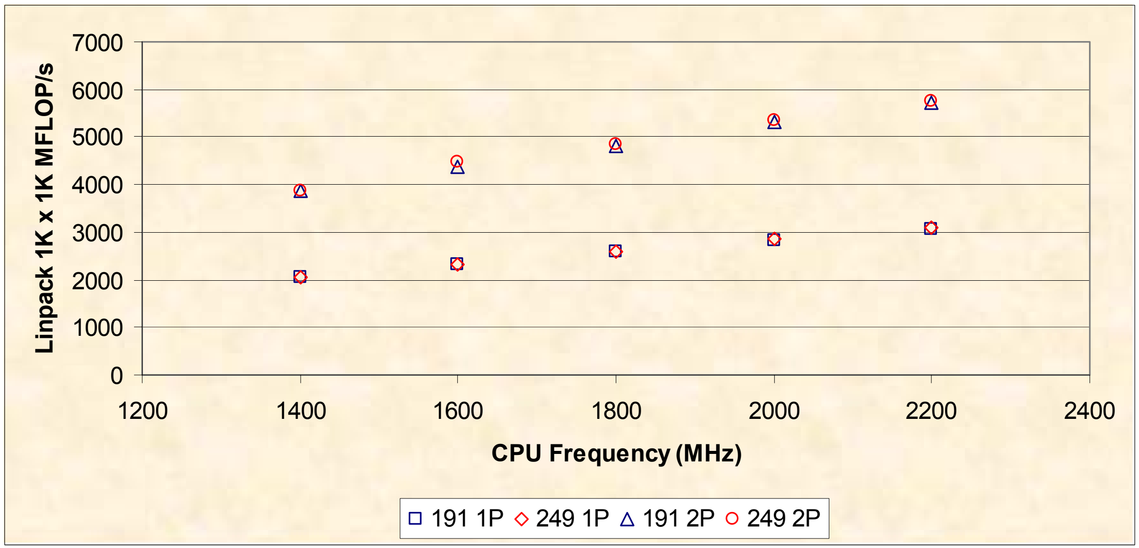

Figure 11. Linpack Performance

These experiments were run using two different SUSE LINUX kernels. The first to be used was k_smp-2.4.19-191, which does a poor job of maintaining memory locality on this system. The experiments were repeated using k_smp-2.4.19-249, which maintains memory locality well. The results obtained under k_smp-2.4.19-191 are effectively identical to those obtained under k_smp-2.4.19-249. This confirms that Linpack is not strongly affected by memory performance. It also points out that SMP kernels, such as 191, may be suitable on NUMA systems when the applications are highly cache-optimized.

Now notice, in Figure 11, that the one- and two-processor Linpack results scale with near perfect linearity. This is expected because the components used most heavily by this benchmark are all governed by the same clock. Speeding up the clock makes each of the components work faster by an amount that is proportional to the speed increase of the clock; therefore, the benchmark scaling is linear. (Although this result may be intuitive, it is still important to confirm that there are no hidden bottlenecks.)

A similar effect occurs with SPEC CINT2000 Rate (see Figure 12). Again, this occurs because, for the most part, application data may be stored and retrieved from processor cache. The greatest challenge of using a NUMA system effectively is to avoid fetching data from a remote processor. Both Linpack and SPEC CINT2000 Rate do this by maintaining most of their data in cache.

On the other hand, SPEC CFP2000 Rate does not fit most of its data working set into cache. Our experience has been that its performance is affected by cache size, processor speed and memory speed. Effective frequency scaling for this benchmark depends on a combination of processor and memory subsystem designs. If either is deficient – there is a bottleneck in the ALU or the memory speed becomes insufficient – the performance will level out even as the processor frequency increases. We use this benchmark for this reason, that is, to check that the memory subsystem remains adequate even as the processor speed increases. If the balance is there, the benchmark will scale linearly. If it is not, the performance may rise for a while, but it will ultimately level out into a plateau.

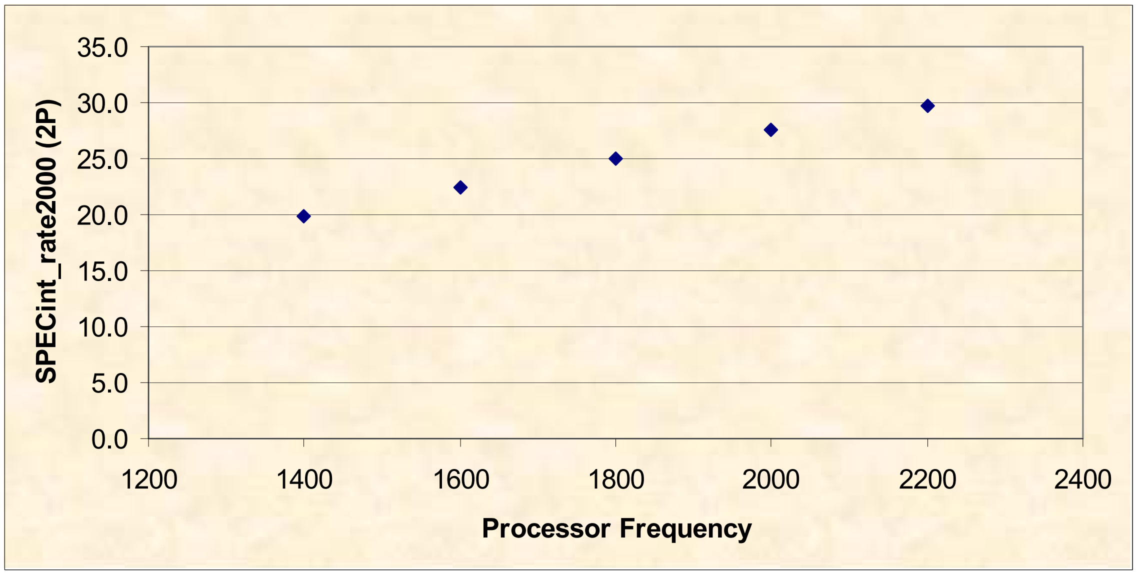

Figure 12. SPEC CINT2000 Rate Performance. All results were measured but have not been reviewed by SPEC, and so are considered estimates.

Figure 13 shows that the benchmark performance scales quite linearly with increasing processor frequency; that is, processor and memory subsystems are well-matched for this workload, and the memory subsystem does not become a bottleneck.

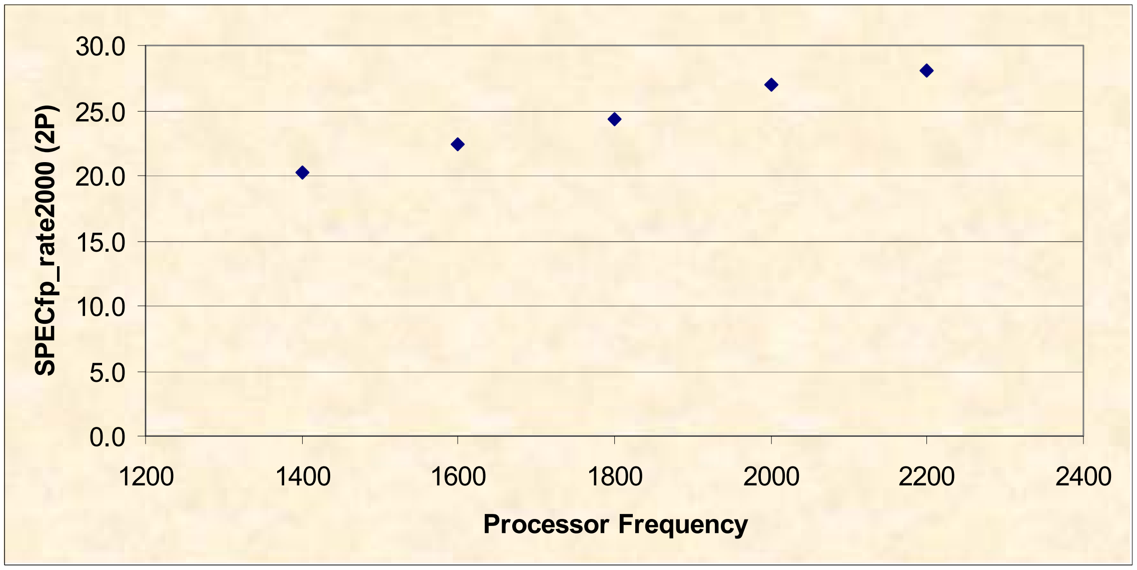

Figure 13. SPEC CFP2000 Rate Performance. All results were measured but have not been reviewed by SPEC, and so are considered estimates.

Conclusions

The AMD Opteron processor-based IBM eSeries 325 is a powerful 1U system. It supports high memory bandwidth and throughput because of its integrated memory controller and NUMA architecture. The memory bandwidth scales with the number of processors; that is, a second processor adds more bandwidth to the system, reducing memory bottlenecks.

Based on our experiments, we make the following conclusions:

1. The memory controller achieves approximately 70% of the hardware peak performance when all four channels are driven at full capacity using DDR333 memory. It achieves greater than 85% of peak when only one channel per processor is driven to capacity.

2. Memory frequency scaling is nearly perfect from DDR266 to DDR333 when only one channel per processor is driven, but is reduced to 19% out of a possible 25% when all four channels are driven. This indicates that the memory controller might have some difficulty with DDR400 memory when the processor speed is 2.0GHz. But 2.4GHz or 2.6GHz processors would have faster memory controllers, and that would reduce conflicts that arise from using DDR400 memory on a 2.0GHz processor.

3. The integrated memory controller is very effective at increasing memory performance as processor clock rates are increased. Memory throughput increases at about 2MB/s for every 1MHz that the processor clock increases. However, there is also a ratchet effect that reduces performance when the memory native clock speed does not evenly divide the processor clock speed, as is the case with DDR266 and DDR333 memory. This ratchet effect would not occur with DDR400 memory.

4. Parallelism within a memory channel can improve performance by up to 1.5% when 256MB or 512MB DIMMs are used within the same channel. This would represent memory configurations with four 256MB or four 512MB DIMMs on CPU A. But there may be a small loss of performance, around 0.25%, when four 1GB DIMMs are used. CPU B is not affected.

5. The selection of operating system kernel has a major impact on memory performance, based on its ability to keep memory references local. SMP kernels do run on the e325 but have just slightly better than half the performance of tuned NUMA kernels. SUSE LINUX and Red Hat Linux each produce at least one kernel that performs well. The SUSE LINUX kernel scheduler was also quite stable.

6. The processor performance increased linearly with processor clock on ALU-intensive workloads. Scaling was nearly perfect for both Linpack and SPEC CINT2000 Rate, which fetch most of their data from processor cache. Because of the high bandwidth to memory, the scaling was also nearly perfect for SPEC CFP2000 Rate, which has a strong memory component as well as a strong floating-point component to its performance.

References

1. IBM eServer 325, ftp.software.ibm.com/pc/pccbbs/pc_servers_pdf/e325spec.pdf.
2. AMD Opteron Processor Data Sheet, http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/23932.pdf.
3. STREAM: Sustainable Memory Bandwidth in High Performance Computers, www.cs.virginia.edu/stream/.
4. J. J. Dongarra, "The Linpack benchmark: An explanation," in A. J. van der Steen, editor, Evaluating Supercomputers, pages 1-21. Chapman and Hall, London, 1990.
5. J.J Dongarra, Cleve B. Moler, G. W. Stewart, *Linpack User's Guide*, Society for Industrial & Applied Mathematics, June 1979.
6. Linpack, www.netlib.org/linpack/index.html.
7. SPEC CPU2000, www.spec.org/cpu2000/.
8. W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, "LU Decomposition and Its Applications," §2.3 in *Numerical Recipes in FORTRAN: The Art of Scientific Computing*, 2nd ed. Cambridge, England: Cambridge University Press, pp. 34-42, 1992.
9. E. Anderson, L. S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenhaum, S. Hammarling, A. McKenney, Sorensen D., Zhaojun Bai, Christian H. Bischof, *Lapack User's Guide*, 3rd edition, Society for Industrial & Applied Mathematics, 2000.
10. Kazushige Goto, "High-Performance BLAS," www.cs.utexas.edu/users/flame/goto/.