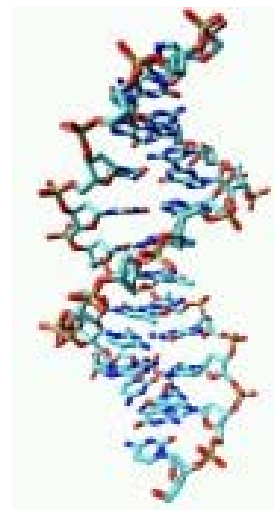
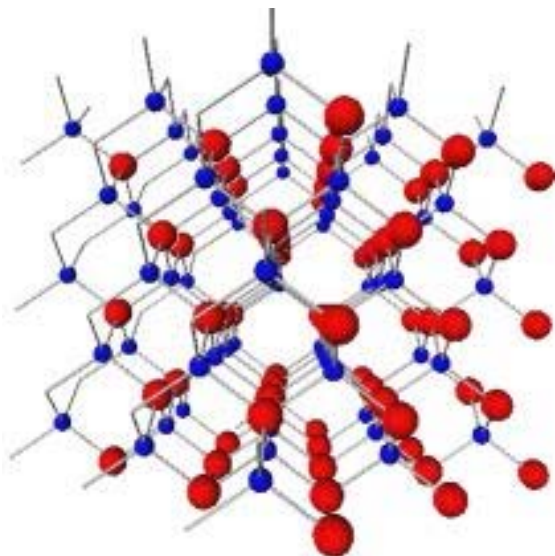


# Scalable Informatics

Performance comparison of AMD Multi-Core systems and single-core systems on chemistry and informatics applications

Joseph Landman, Ph.D.  
Scalable Informatics LLC  
<http://www.scalableinformatics.com>  
landman@scalableinformatics.com



AMD, the AMD Arrow logo, AMD Opteron and combinations thereof, are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

## **Performance comparison of AMD Multi-Core systems and single-core systems on chemistry and informatics applications.**

This white paper will review recent performance benchmarking and analysis of AMD Multi-Core systems, comparing identical runs across such systems. The goal of this paper is to examine and analyze performance of these systems, and compare with existing single-core AMD Opteron™ processor based systems. Specifically, the performance effect on computational chemistry and bioinformatics codes running on a Multi-core system sharing a memory controller on a single chip, will be addressed.

Performance will be measured using applications in common use by researchers in computational chemistry and informatics fields. Due to the breadth of these fields, this report cannot possibly cover all possible programs or computational simulation/analysis cases. Specific test cases and programs were selected from the most commonly used tool sets. Publicly available benchmark data was used. It is assumed the reader has an understanding of the principles associated with code compilation and code optimization.

### **Improving Microprocessor Instructions per Clock Cycle**

One critical limitation on performance on microprocessors is the number of instructions per clock cycle (IPC) that can be executed. Microprocessor developers have sought to provide mechanisms to increase the effective IPC as this is regarded as a significant performance bottleneck. Efforts in this direction have included approaches such as MMX<sup>®</sup>, 3DNow!™, SSE, and SSE2. The fundamental idea is to create a large register within the microprocessor that enables operation on multiple values at once. So if the compiler places four single precision floating point numbers into a single 128 bit register, and executes an operation in a single clock cycle, then for those operations the microprocessor appears significantly faster than several individual operations.

Unfortunately SSE, SSE2, and related are large and complex portions of the processors. Moreover, designing the silicon gets more complex with more functional units, and deeper pipelines. Higher complexity silicon often results in lower manufacturing yields, which increases per-unit costs. The costs associated with design and redesign of the processors are large, though the net performance benefits for end user code can tend to be low, as the new features and functionality are not transparently usable. Moreover, the benefits will remain low if compiler developers are slow to adopt these new technologies.

Adding functional units is not the only way to increase the IPC. Building more or deeper pipelines could enable more instructions in flight, and thus a larger effective IPC. Increasing the clock speed could also increase IPC. However, other considerations work to offset the advantages of these approaches.

- Building wider pipelines requires more silicon spent in management of the functional units, which leaves less silicon for the functional units themselves.
- Building deeper pipelines also requires more silicon, and typically requires that the pipeline stages do less work per stage. This must be offset by an increase in clock rates.
- Increasing clock speed has implications for higher power consumption.

CPU designs that decouple a memory controller into a discrete component or chipset may suffer from access speed imbalance. This is especially noticeable when the front side bus connection to the chipset must remain a fixed speed due to other considerations. For this design approach, microprocessors connected to the system bus require large L2 caches, consuming significant fractions of the overall CPU silicon, to buffer the imbalance.

## Implications for Computational Chemistry and Informatics Programs

Since modern computational chemistry and to a degree, modern informatics programs are able to run across multiple CPUs, a simple method of increasing program visible IPC would be to increase the number of existing CPUs applied to the computation. For shared memory parallel programs using OpenMP or pthreads, using an additional CPU is a simple matter. For distributed memory parallel programs using MPI, additional CPUs are also relatively simple to use. Hence this design should be simple to use with existing programs, without relying upon a compiler developer to adopt new technology. This effectively lowers a barrier to additional processor cycle usage.

A possible bottleneck which is currently seen in various multiple-CPU designs, concerns sharing of consumable resources such as SMP memory bandwidth. Shared resources represent bottlenecks, as access to these shared resources may require access serialization. Such effects are often visible when running applications such as AMBER or GAMESS, in terms of marginal increase in performance when running 2 threads per dual CPU node instead of 1 thread per dual CPU node.

An important question for computational chemistry and informatics end users is whether or not this will impact their runs on dual core processors. The figures below are block diagrams of the single and dual core AMD64 processors.

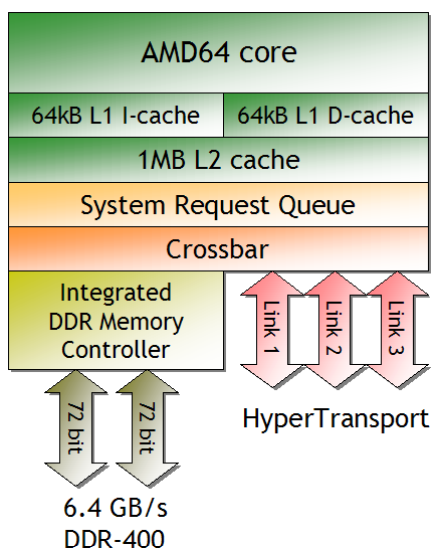


Figure 1: Single core AMD64 block diagram

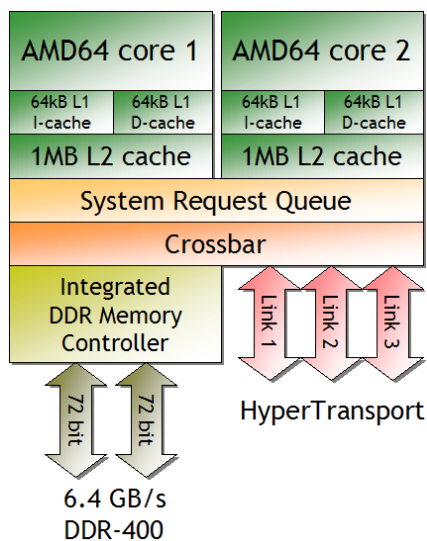


Figure 2: Dual core AMD64 block diagram

Within the dual core AMD64™ processor, the cores are connected through a queue and crossbar switch. (This switch was incorporated in the original processor architecture announced in 2003.) The two cores on a dual core processor share a single on-chip memory controller. If the memory controller resources are overused during chemistry and informatics applications execution, a significant and noticeable drop in performance relative to an “equivalent” single core design would be expected. Conversely, if performance degradation is minimal or nonexistent then this bottleneck is not likely a problem for these codes running similar simulations.

### **Benchmark measurements:**

The major focus of this work is to compare similar machines, specifically to explore the performance of dual core processors as compared to single core processors on identical test cases. Several applications have been selected for this effort, and in all cases, identical binary executable images were used. The applications used for this work include GAMESS (November 2004), AMBER 8, NCBI BLAST, and HMMer from Washington University. Single machines were used for testing, to remove the effects of network fabrics from the results. MPICH v1.2.6 was used with the ch\_shmem device using the PathScale 2.1 compiler suite.

### **Software environment:**

The Portland group 6.0-2 and PathScale 2.1 compilers were used to generate binaries for these tests. All tests were run using the 64-bit environment. Binary executables were tested for correctness using standard test cases supplied with the programs, or by using well-known and well-understood test cases. Optimization options were used that generated the fastest correct results, and the ACML 2.5.1 libraries were used.

AMBER 8 requires MPI, and mpich v1.2.6 was used with the ch\_shmem device using the PathScale 2.1 compiler suite. GAMESS was built using the Portland Group 6.0-2 compilers specifying sockets-based DDI as the distributed memory technology. NCBI BLAST and Washington University’s HMMer were built using the Portland Group 6.0-1 compilers. NCBI BLAST 2.2.10 was built using Scalable Informatics LLC Opteron support patches. HMMer 2.3.2 was built from unchanged source. BLAST and HMMer were built with pthread support. Each code was compiled using the compiler generating the fastest code. As noted earlier, identical binary images were used to run the single and dual processor benchmarks.

All machines are running Linux based upon SuSE 9.x with a 2.6.11-rc4 kernel. No kernel or system tuning was performed.

### **Benchmark descriptions:**

The AMBER 8 benchmark consisted of the benchmark cases distributed with AMBER 8. Parallel sander was used, though pmemd would have been a good choice as well. The GAMESS benchmarks used the T.U. Dresden benchmark tests<sup>1</sup>. The NCBI BLAST and

---

<sup>1</sup> These benchmarks may be found at <http://www.chm.tu-dresden.de/edv/bench/bench.html>. Other information appears at <http://www.msg.ku.edu/~msg/MGM/links/bench.html>

HMMer benchmarks are modified versions of the baseline tests distributed with BBS<sup>2</sup>, specifically designed to probe the effects of multiple threads on application performance for dual core systems. The nr database<sup>3</sup> used in the BLAST benchmark is from 1-Jan-2005.

All benchmarks were wrapped in BBS xml experiments. These BBS xml experiment files, BLAST and HMMer input decks will be available for download from <http://www.scalableinformatics.com> expected to be available by August 1, 2005.

### Benchmark Platforms:

<i>Platform</i>	<i>Clock speed (GHz)</i>	<i>Nprocessors /Ncores</i>	<i>Memory size (GB) / speed</i>
AMD Opteron™ 275 processor	2.2	2p/4c	8/DDR400
AMD Opteron 848 processor	2.2	2p/2c	8 / DDR400
AMD Opteron 850 processor	2.4	4p/4c	32 / DDR333

The benchmark platforms were prepared and managed by the AMD Developer Center team.

### Benchmark Results

All measurements are based upon execution time as obtained from a wall clock measurement by BBS. Apart from these runs, the systems were largely idle. No special effort to clear memory/disk buffer cache was made. The measurements that are most relevant are the ones the end users would experience for themselves, and it is highly unlikely that an end user would reboot a machine after each run, unmount drives, or run programs to flush buffer cache.

All run times were normalized to single processor/single core (thus single thread) runs of the AMD Opteron™ 275 processor. These single thread runs were performed using MPI on the ch\_shmem device for AMBER, by setting the *DO\_PARALLEL* environment variable to *mpirun -np 1*. The GAMESS *runqms* script was adjusted to enable runs from 1 to 4 threads on a single Linux host. The BBS experiment files do not include the capability for environment adjustment in the version 3 of the tool. This capability will be available in the near term release of version 4, so this variable was set by hand prior to the single thread run.

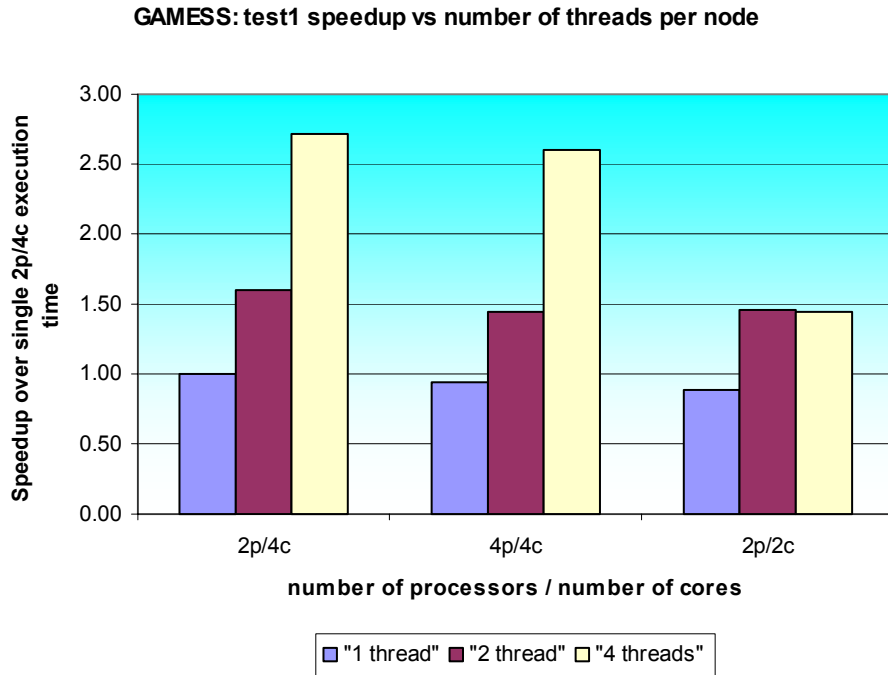
Since the data is normalized as indicated, a value of 1.05 for a score in a particular test would indicate that the test completed 5% faster than the AMD Opteron 275 processor single core test.

<sup>2</sup> BBS is the bioinformatics benchmark system, found at <http://www.scalableinformatics.com/BBS>

<sup>3</sup> Available from [http://downloads.scalableinformatics.com/downloads/benchmark\\_data\\_sets/db/](http://downloads.scalableinformatics.com/downloads/benchmark_data_sets/db/)

## GAMESS:

Test1:



As can be immediately seen, the 2p/4c AMD Opteron™ 275 processor based system was slightly faster than the 4p/4c AMD Opteron 850 processor based system on these tests. This was not explored in detail, though memory speed, process affinity, and similar effects could be factors.

### **Memory Speed**

The AMD Opteron 850 processor based platform used DDR333 memory that has a slightly lower speed than the DDR400 memory (about 16.8% slower than DDR400). The clock speed of the Opteron 850 is about 9.1% faster than the clock speed of the AMD Opteron 848 processor and the AMD Opteron 275 processor. It is possible that the combination of the faster clock and the slower memory on the AMD Opteron 850 processor renders it quite close in performance to the AMD Opteron 275 processor and AMD Opteron848 processor in single thread performance.

It was also observed that the scaling of the 2p/4c system is similar to the 4p/4c system. These tests are regarded as fairly memory intensive, so the issue of sharing a single memory controller and a single memory did not have a noticeable impact on overall performance for this test.

### **Processor Affinity**

As indicated previously, processor affinity may play a role. Processor affinity in this context means that each thread that ran on a particular processor index resumes on that index. Respecting processor affinity usually means that more of the processor's cache will contain relevant cache lines when the thread resumes. If the thread resumes on a different processor, the remote cache will need to be flushed,

and cache lines loaded. This costs time, and in cache intensive calculations, the best performance will be obtained from spatial and temporal locality, which keeps as many of the same cache lines in the same processor's cache for as long as possible.

### **Other Factors**

There is another possible effect, specifically where memory is placed relative to the processor accessing it. In the non-uniform memory access (NUMA) architecture, there are latency and bandwidth costs for accessing "remote" memory within a single system image. These costs are mitigated by using high bandwidth and low latency interconnection fabrics such as HyperTransport™ to interconnect processors and memory. However, the operating system needs to be aware of which processor allocates memory for a particular process, as that memory and the process that allocated that memory should remain running on that node of the NUMA system.

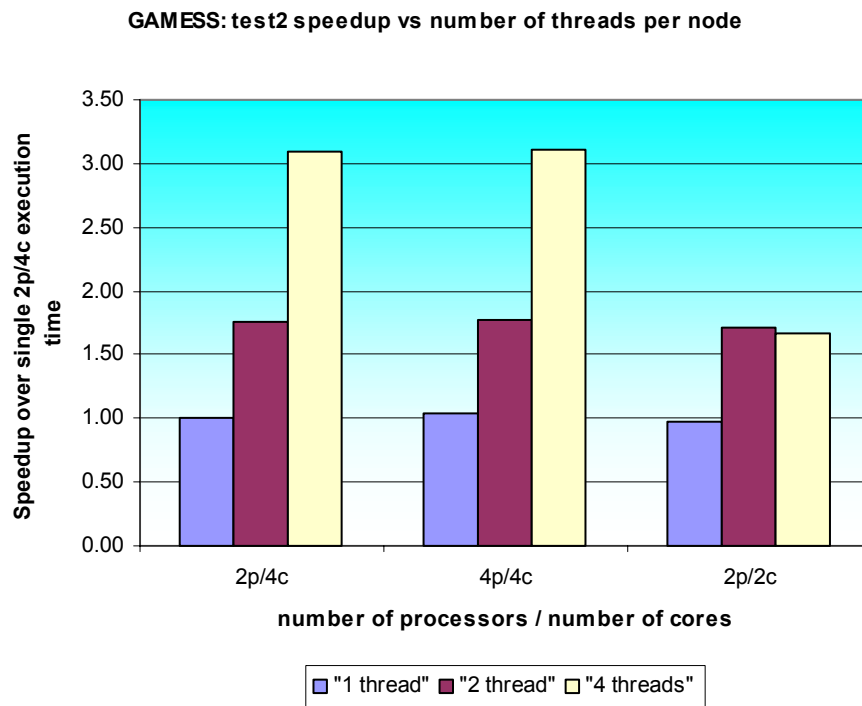
With four memories and four cores to allocate and schedule on, there are more possible sub-optimal combinations of cpu assignment for a particular thread, relative to where the memory is itself allocated. Specifically for each core, there are three suboptimal and one optimal memory placements. Over four cores, this means that there are twelve suboptimal and four optimal possible arrangements. The 2p/4c systems have two memories and four cores, with two cores sharing access to a single memory. Specifically for each core, there is one optimal and one suboptimal choice of memory arrangement. Over the four cores, this leads to four suboptimal and four optimal choices of memory arrangement. Since the dual core processors do not share L2 cache, there could be process affinity effects in addition to the memory arrangement issues.

If the kernel does not pin the process and memory to a particular CPU core, then it is possible that when the threads get scheduled to run again on a core, that it is running on a sub-optimal choice. In this case, the 4p/4c system is more likely (75% probability assuming random assignment of threads to cores as compared to 50% probability) to have a sub-optimal arrangement of threads and memories than the 2p/4c system.

Since the Linux scheduler is generally smarter than randomly scheduling threads and memories, this should not be very noticeable, but it might provide an explanation as to why a 2p/4c system appears to be slightly faster than a higher clock speed 4p/4c system.

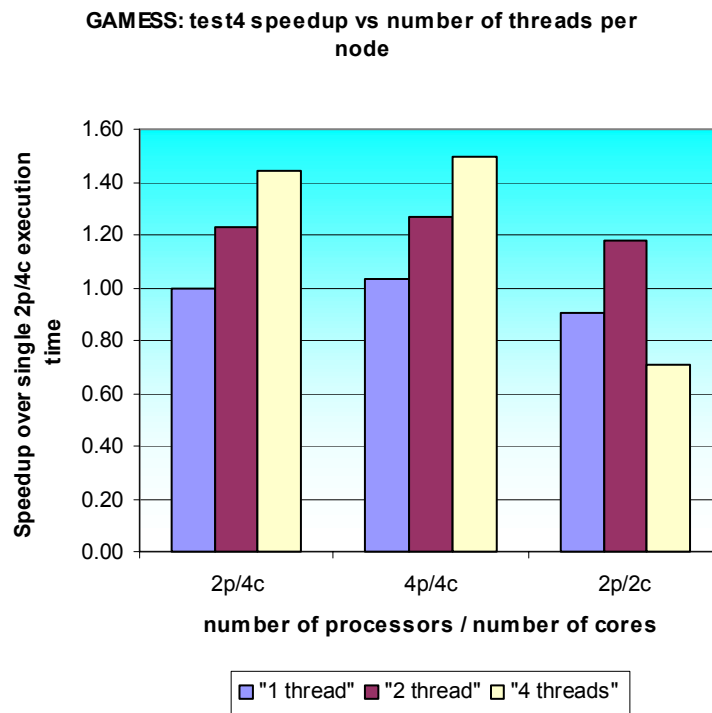
The 2p/2c results from the AMD Opteron™ 848 processor are provided for comparison. The 4-thread runs were not included in these results due to the network effects. The 2p/4c AMD Opteron 275 processor based system and the 4p/4c AMD Opteron850 processor based system used an internal shared memory DDI device, while a four thread run with the AMD Opteron 848 processor based 2p/2c system requires using either a low latency network, or an Ethernet socket layer, which has a noticeable impact on performance. Four threads were run instead on a single AMD Opteron 848 processor based 2p/2c system.

Test2:



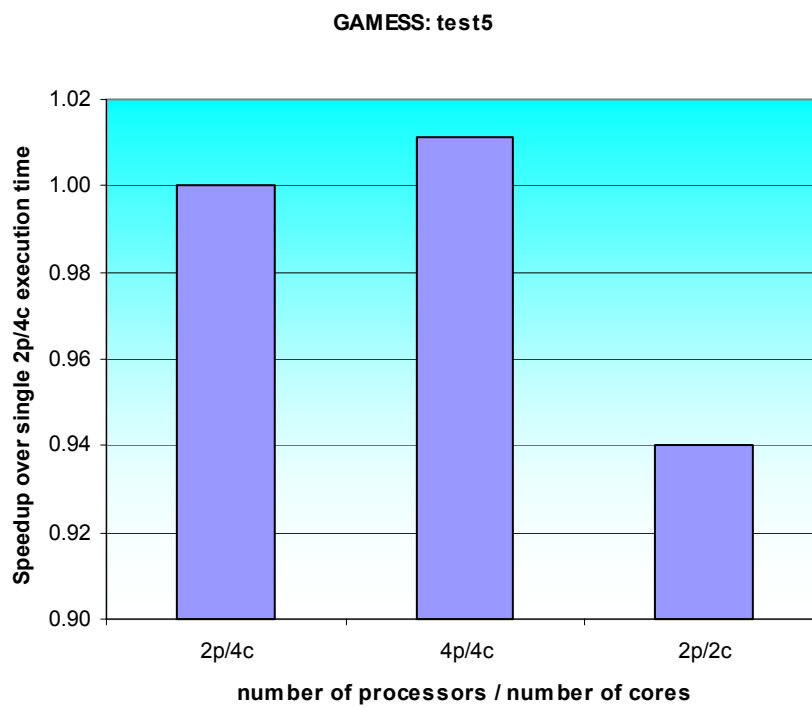
Again in this test, the single thread performance of the AMD Opteron™ 275 processor based 2p/4c system is comparable to the single thread performance of the AMD Opteron 850 processor based system. The 4-thread performance of the AMD Opteron 275 processor based 2p/4c system and AMD Opteron 850 processor based 4p/4c systems are very similar.

Test4:

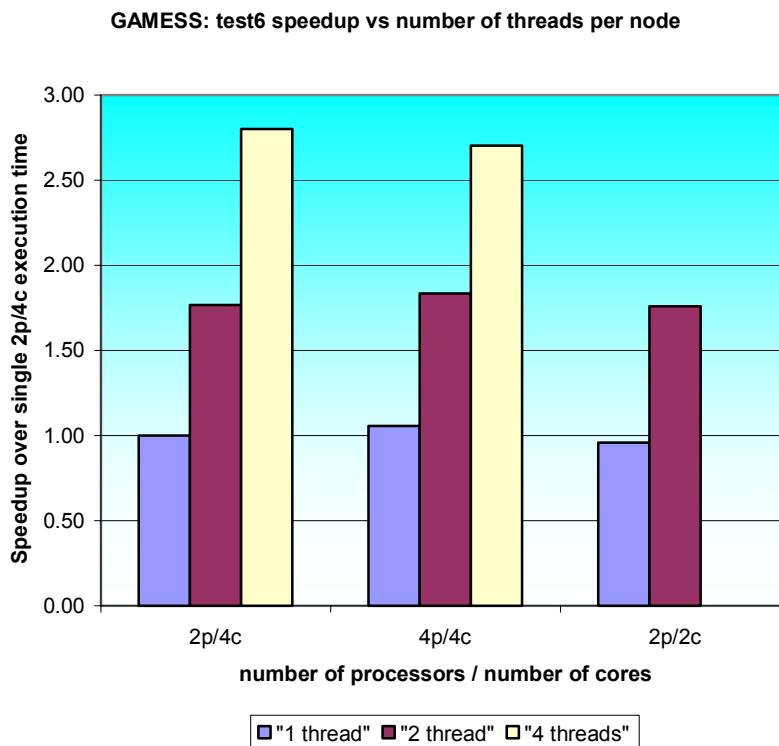


Again, the performance of the AMD Opteron™ 275 processor and AMD Opteron 850 processor are quite similar.

Test 5:



Test 6:

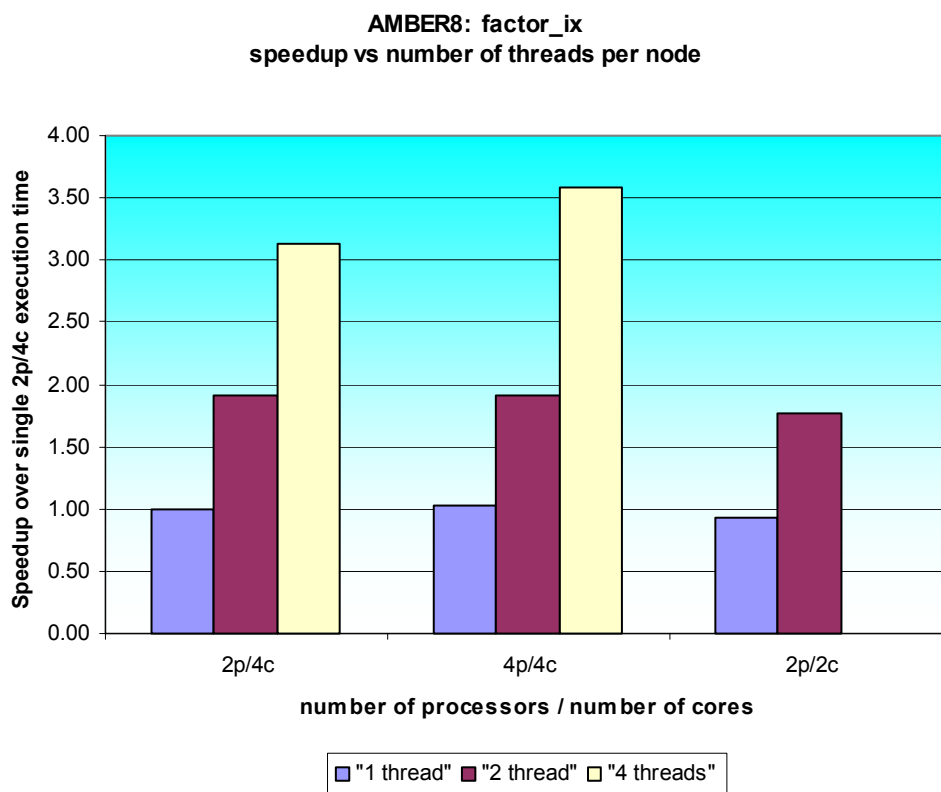


Overall for these tests with GAMESS, it was difficult to distinguish the overall performance between the AMD Opteron™ 850 4p/4c and the AMD Opteron 275 processor based 2p/4c systems.

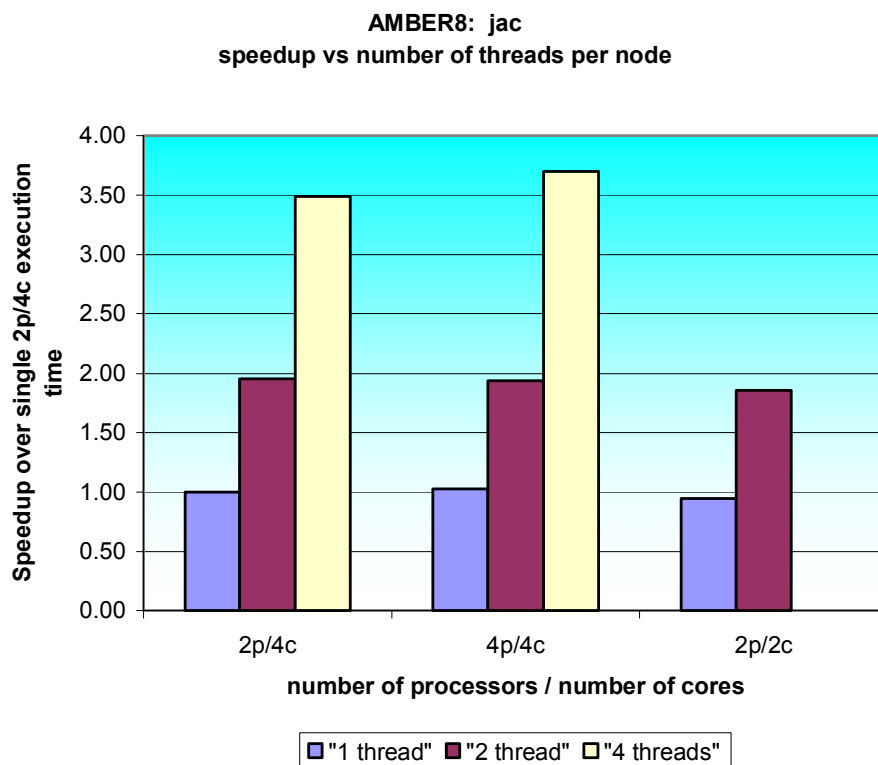
### **AMBER8:**

Amber's design is one which requires high sustained memory bandwidth for performance. For each of the tests we observed the following performance relative to a single thread running on an AMD Opteron 275 processor based 2p/4c system.

factor\_ix:

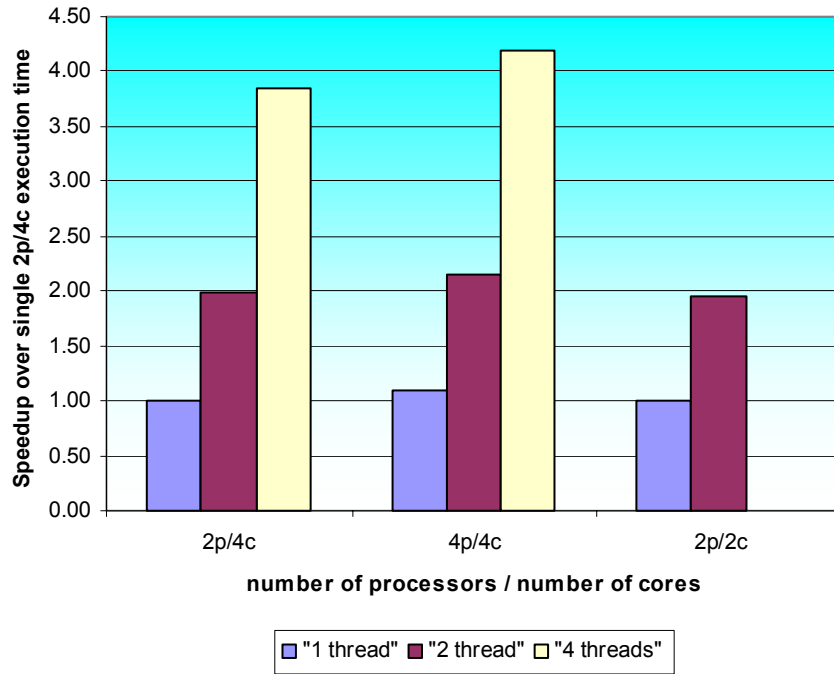


jac:



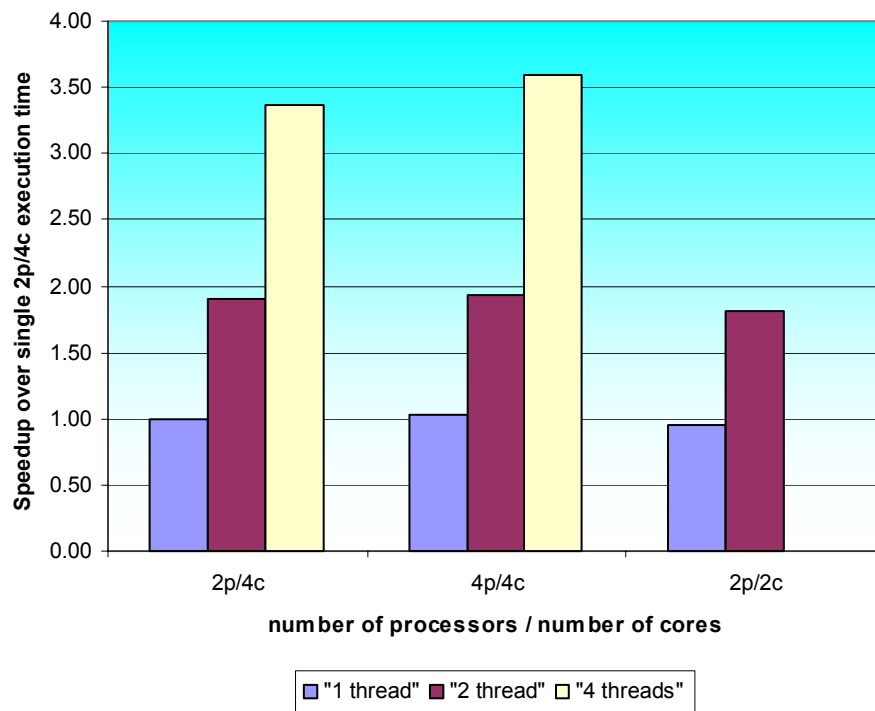
gb\_mb:

AMBER8: gb\_mb  
speedup vs number of threads per node

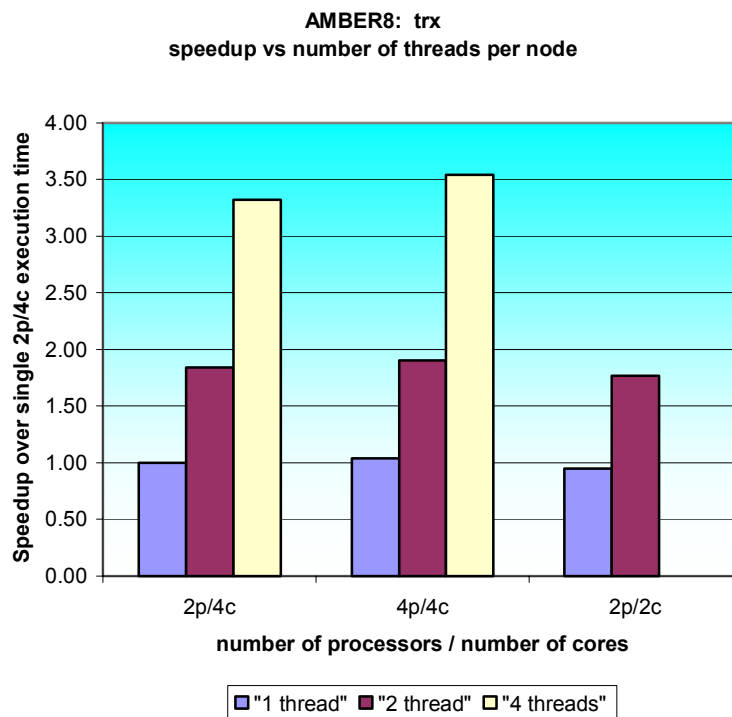


hb:

AMBER8: hb  
speedup vs number of threads per node



trx:



As can be seen in these results, the AMD Opteron™ 850 processor based 4p/4c system is marginally faster than the AMD Opteron™ 275 processor based 2p/4c system. As AMBER8 consumes significant memory bandwidth, the AMD Opteron™ 850 processor based 4p/4c system with four independent memories demonstrates a slight advantage (under 10%) in most of these tests.

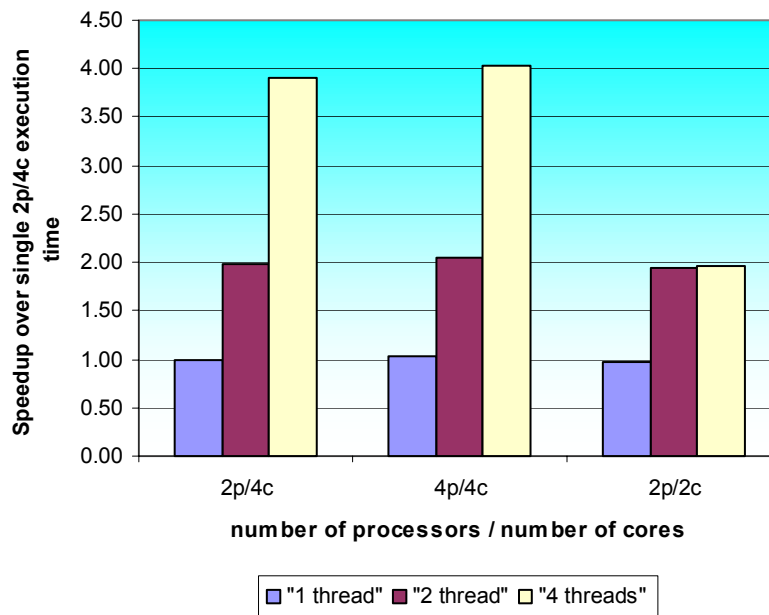
It is worth noting that since AMBER8 does consume significant memory bandwidth, the memory contention issue that could reduce overall performance on a AMD Opteron™ 275 processor based 2p/4c system under heavy memory usage shows a small effect, at most.

### **NCBI BLAST:**

Due to the way NCBI BLAST performs its work, the particular algorithm may be memory latency bound, or processor bound. The blastx algorithm is typically processor bound. Benchmarks are possible using tblast{x,n} which are significantly more processor bound (due to working with five more reading frames than blastx uses), though they were not used for these tests. All the blast algorithms are quite sensitive to cache size, the larger the cache, the better the performance.

What it observed for a simple blastx of 74 ESTs against the nr database, using one, two and four threads is as follows:

**BLAST: blastx**  
speedup vs number of threads per node



The AMD Opteron™ 275 processor based 2p/4c system exhibited similar scaling and performance relative to the AMD Opteron™ 850 processor based 4p/4c system. This test had been originally designed to oversubscribe the threads to the processors to explore performance of SMT systems. Those systems demonstrated performance regressions under excessive threading. As can be seen from the 2p/2c results running 4 threads, performance is approximately constant after oversubscription.

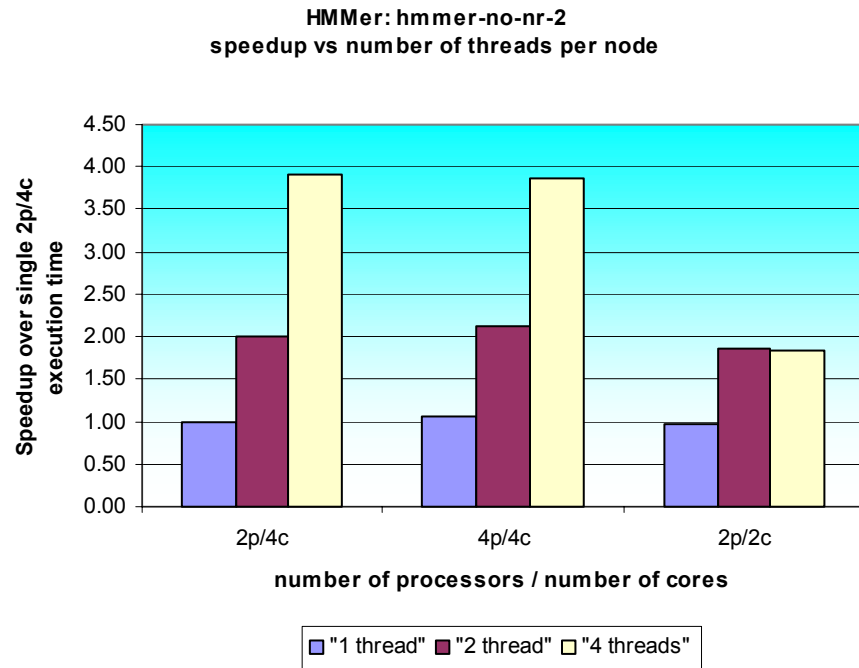
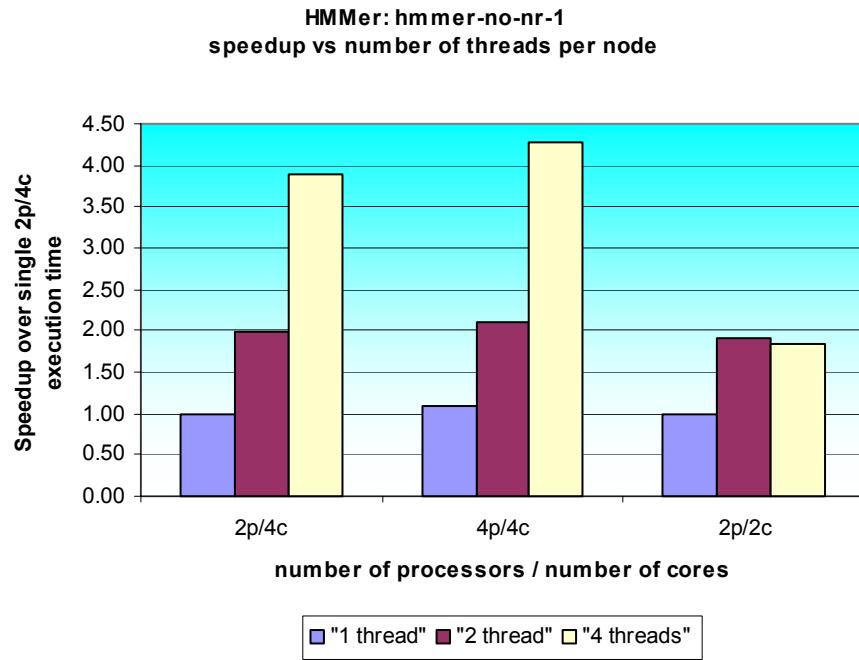
### HMMer:

These benchmarks were based upon a suggestion from Professor Eddy of Washington University in St. Louis, who developed and distributes<sup>4</sup> the HMMer code. Special dual-core versions of the baseline BBS tests were generated for this test.

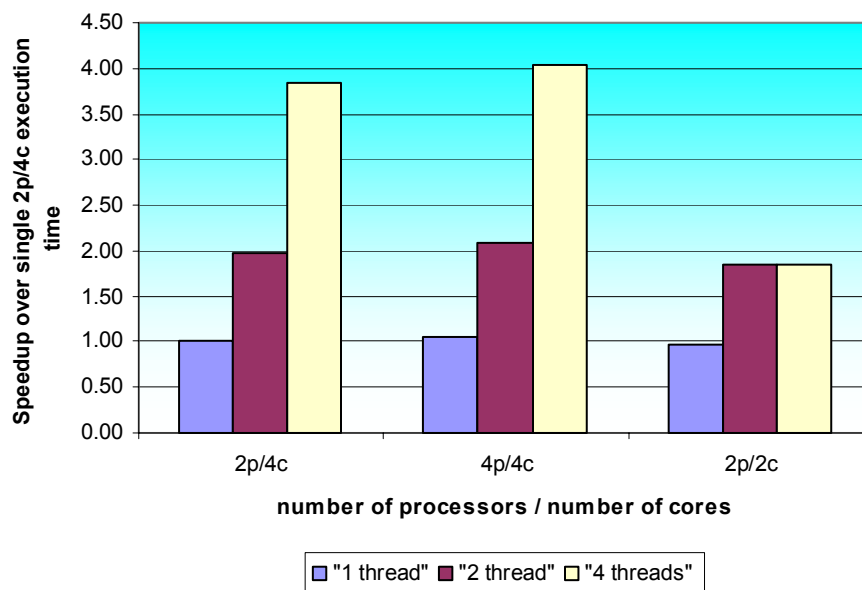
---

<sup>4</sup> See <http://hmmer.wustl.edu>

The performance is as indicated below:



**HMMer: hmmer-no-nr-3**  
**speedup vs number of threads per node**



Again, as observed for NCBI BLAST, the AMD Opteron™ 275 processor based 2p/4c system performed quite close to the AMD Opteron 850 processor based 4p/4c system. These tests did not use databases, instead they generated random sequences and random databases. Therefore, over a wide range of testing, performance will vary from run to run, but not significantly.

## Conclusions:

After building and running the tests, the observations clearly show that the AMD Opteron 275 processor based 2p/4c system does not suffer from memory contention effects for these codes with these tests. The observations also support the notion that for these codes and these tests, scalability is fairly similar between 4p/4c systems and 2p/4c systems. This is important as there are more shared resources in the 2p/4c system, and sharing has a tendency to limit scalability. However, as can be seen from these results with these codes, such effects are likely comparable in size to the measurement noise. They will not likely impact day-to-day execution of these programs(though it is always possible to construct a pathological case that does impact performance). For these codes and these test cases, the potential bottleneck appears not to be a problem.

Also of significant importance, the identical binaries are used across all these systems. No special compilation was used for 2p/4c systems, no special optimization was performed beyond what end users are likely to do on their own, no kernel tuning, or system tuning was performed. The system was usable without any extra effort.

Finally, the overall performance of the AMD Opteron 275 processor based 2p/4c system was quite good, and comparable to single core systems of the same clock speed.