Linpack HPL Performance on IBM eServer 326 and xSeries 336 Servers

Dr. Douglas M. Pase
IBM xSeries Performance Development and Analysis
3039 Cornwallis Rd.
Research Triangle Park, NC 27709-2195
pase@us.ibm.com

Abstract

In this paper we examine the performance of the Linpack benchmark on IBM® eServer™ 326 (e326) and IBM eServer xSeries™ 336 (x336) systems. The e326 is powered by AMD Opteron™ processors, and the x336 is powered by Intel® Xeon processors. We consider system factors that affect benchmark performance, such as processor and network performance, and show that it is not sensitive to memory performance. We also consider ways to tune the benchmark. In particular we examine what are good values for P, Q, N, *and* NB, *and conclude that optimal values for* NB *must be determined through experimentation, although many values of* NB *can be eliminated through simple analysis.*

1. Introduction

Linpack HPL (High Performance Linpack) is a benchmark that measures the performance of a node or cluster using a work load that is characteristic of some scientific and technical applications. It solves a dense system of linear equations. The system is represented as a matrix that has been divided into smaller pieces, called tiles, and distributed across the available processors. Performance is reported as the number of floating-point operations per second achieved by the system. The user is able to vary the size of the problem and a number of parameters that describe how the problem is to be solved. Some of the parameters describe how the problem is decomposed into tiles, and how those tiles are distributed.

Linpack itself is a collection of subroutines that analyze and solve linear equations and linear least-squares problems [1][2]. 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 [3] to solve a general dense system of linear equations using LU factorization. Both the library and the benchmark are freely available from the NetLib Web site [4]. LU factorization [5] is similar to a technique commonly taught in college Linear Algebra courses, called *Gaussian elimination*, but it is significantly improved to increase numerical stability and performance. The

Linpack library has largely been replaced by Lapack, which extends and improves upon the routines [6]. In Lapack, 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 double-precision High Performance Linpack (HPL) benchmark over a wide range of problem sizes. Our benchmark implementation uses the high-performance BLAS created by Kazushige Goto [7].

The most important Linpack measure is R_{max}, which is the maximum measured Linpack performance, in gigaflops per second (GF/s). The value N represents the number of equations used to obtain R_{max}. Another measure is R_{peak}, which is the theoretical peak performance for the system, usually obtained by multiplying the total number of processor cores, the processor clock frequency and the theoretical number of 64-bit vector floating-point results per clock. The efficiency of a system can be obtained by dividing R_{max} by R_{peak}. The final measure used is $N_{1/2}$, which represents the number of equations required to obtain one half of the performance of R_{max}. This value indicates the robustness of the system. A system that has a small $N_{1/2}$ is able to operate well over a wide range of problems.

Linpack HPL is primarily a processor-core-intensive benchmark. It heavily exercises the vector floating-point unit of each processor core. The faster the processor core and the more cores that are available, the better the benchmark performance. Data is moved from memory into cache and used heavily from cache, so larger caches do help. With current cache sizes of 1MB or more, most of the current working data set already fits into cache, so memory performance is not seen as a major factor in determining Linpack performance.

A factor that *does* affect benchmark performance is how the tiles are defined and distributed. Linpack alternates between computation and communication when solving a system of equations. The work to be done in the computation phases depends on the number and size of the tiles to be solved. How the system of equations is tiled affects how the workload is balanced across the system, and how efficiently each tile can be processed. These factors can be significant. Tile definition and distribution are determined by the problem size, N, and the number of tiles that are used. This, and the speed of communication, affects how much time is spent in communication. As problem sizes increase, the amount of computation grows faster than the cost of communication, and the impact of communication performance diminishes accordingly.

These experiments include both single-node tests and tests using a cluster of nodes. For the single-node tests, communication takes place within shared memory, so communication speed is not a factor. Even so, how the tiles are created and used still has a major impact on Linpack performance that is not easy to predict. Several network types were used for the cluster tests to show the effect networks have on performance.

Linpack is a kernel benchmarks; that is, it focuses on measuring the performance of a small, relatively simple mathematical kernel. In so doing it gives an accurate measure of the performance of a single subsystem of the computer. But Linpack is relatively insensitive to many important factors, so it is not a good indicator of the system or cluster as a whole.

2. System Performance

Linpack is a good indicator of how much performance can be obtained using 64-bit vector floating-point operations. Scalar operations can be substituted for vector operations, but when both are available the vector operations are usually faster. In this benchmark the operands are very often cached, so the operations tend to be processor-core-intensive. As such, performance is almost entirely dependent on core frequency. Performance also generally increases with problem size, rapidly for small sizes, more slowly for larger problems, so memory size is important. The benchmark is affected only slightly by cache size, memory speed and network performance.

Figure 1 compares the performance of several systems over a variety of problem sizes, processor frequencies, processor cores, and cache sizes. Many different experiments were run for each problem size, and only the best results for each system are reported. Both single-core and dual-core AMD Opteron processors were used, as well as Intel Xeon processors with 1MB and 2MB L2 caches. All AMD Opteron processors have a 1MB L2 cache, and all Intel Xeon processors have a single core with Hyperthreads disabled. Systems are listed in order of increasing performance.

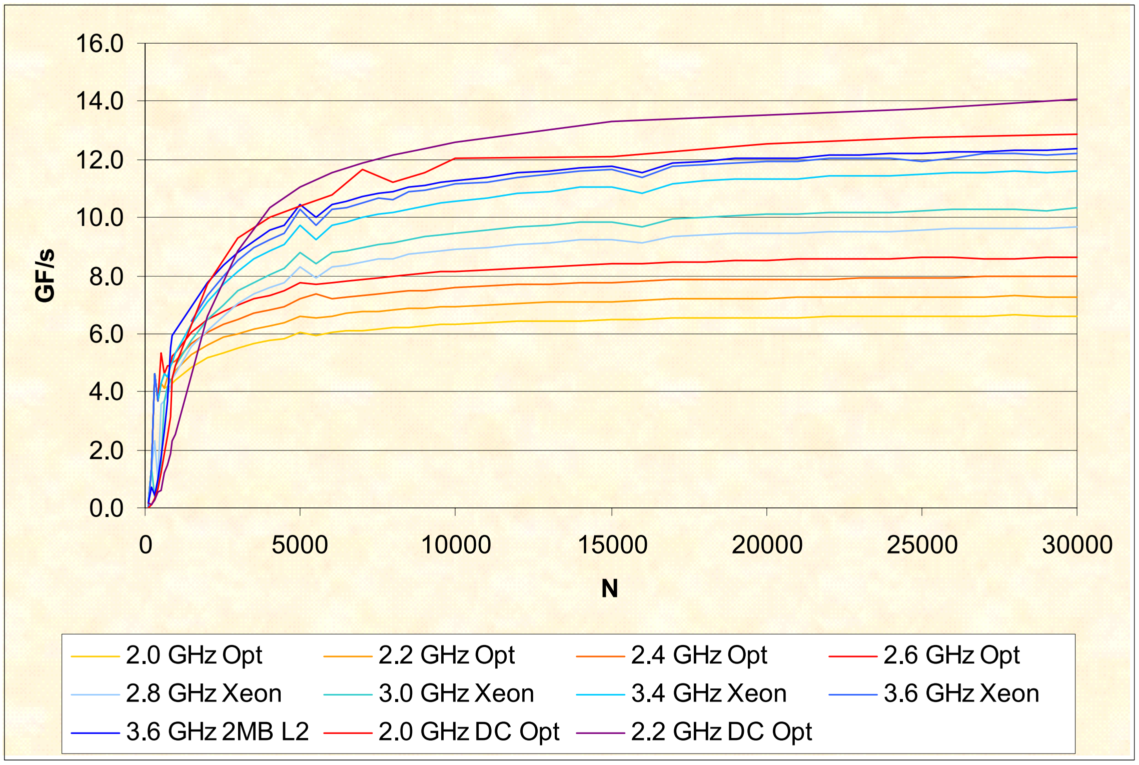

Figure 1. Single-Server Dual-Socket Linpack Performance

The most notable feature of Figure 1 is the similarity of the results. All processors are from the x86-64 and EM64T processor architecture families, and so have certain things in common. The most important point of commonality is that they use a single Streaming SIMD Execution (SSE2) vector unit in each processor core. The efficiencies are strikingly similar, which makes comparing system performance as simple as comparing processor frequencies and cores. A single-core processor with 50% greater clock frequency will also have 50% greater Linpack performance than a similar single-core processor. A system with four processor cores and the same clock frequency will have about twice the performance of a similar system with only two cores. Although finding the right combination of tuning parameters may not be easy, the potential performance of the benchmark is quite clear.

Another feature that stands out is that performance increases (nearly) monotonically with problem size. Performance grows rapidly until N is in the range of 5000 to 10000, after which performance continues to improve much more slowly. In all cases $N_{1/2}$ is well below 5000, and in most cases it is even below 1000. This is significant for two reasons. First, it shows that these systems are able to obtain a significant fraction of their achievable performance with very small problem sizes. Stated differently, near maximum performance can be achieved on many problems with modest investment. It isn't an unusual occurance achieved only after heroic efforts.

The second reason this is significant is that it shows two of the older Linpack versions, DP and TPP, do not accurately reflect system performance. Linpack DP and Linpack TPP use small, fixed problem sizes of 100 equations and 1000 equations, respectively. It is very clear from the diagram that both show performance in a range that is heavily dominated by factors other than floating-point performance. Unfortunately, both benchmarks are still occasionally used by customers trying to assess server performance. When this happens, it should be pointed out that very small Linpack runs will probably not measure what they believe it will, and that better alternatives are available.

A third feature, one that is a little harder to see, is the comparison of cache sizes. Among the systems tested is one with two 3.6 GHz Intel Xeon processors, code named "Nocona", with a 1MB L2 cache. Another system is nearly identical, but it uses two 3.6 GHz Intel Xeon "Irwindale" processors, with a 2MB L2 cache. The difference in performance is measurable and consistent, but it is quite small, only about 1%. This is due, in part, to the fact that the BLAS are so well tuned that their cache hit ratios are already quite high, and larger caches have little opportunity for improvement. It is also due to the fact that the BLAS libraries are tuned for a specific memory hierarchy, and the library we used was tuned for the 1MB caches. Using a library tuned for the larger cache would improve performance, perhaps by as much as 5%.

The next diagram, Figure 2, shows that Linpack is *not* especially sensitive to the memory performance of the system, because of the high cache utilization already mentioned. This set of experiments compares the performance of two e326 clusters, one with 333 MHz Double Data Rate (DDR333) memory installed, the other (actually the same cluster) with DDR400 memory installed. The cluster with DDR400 memory has better than 20% greater memory bandwidth than the cluster with DDR333 memory installed, yet the Linpack performance difference between the two clusters is less than 1%. This is a clear indication that memory performance does not play a significant role in determining Linpack performance.

Another indicator that Linpack is insensitive to memory performance can be seen by comparing an Opteron-processor-based system with one based on a Xeon-processor. Opteron processors have significantly better memory throughput than Xeon processors, by greater than a factor of two, and yet the Xeon processors are at least as efficient as the Opteron processors. If memory throughput were a factor, the Xeon-processor-based systems would be significantly less efficient.

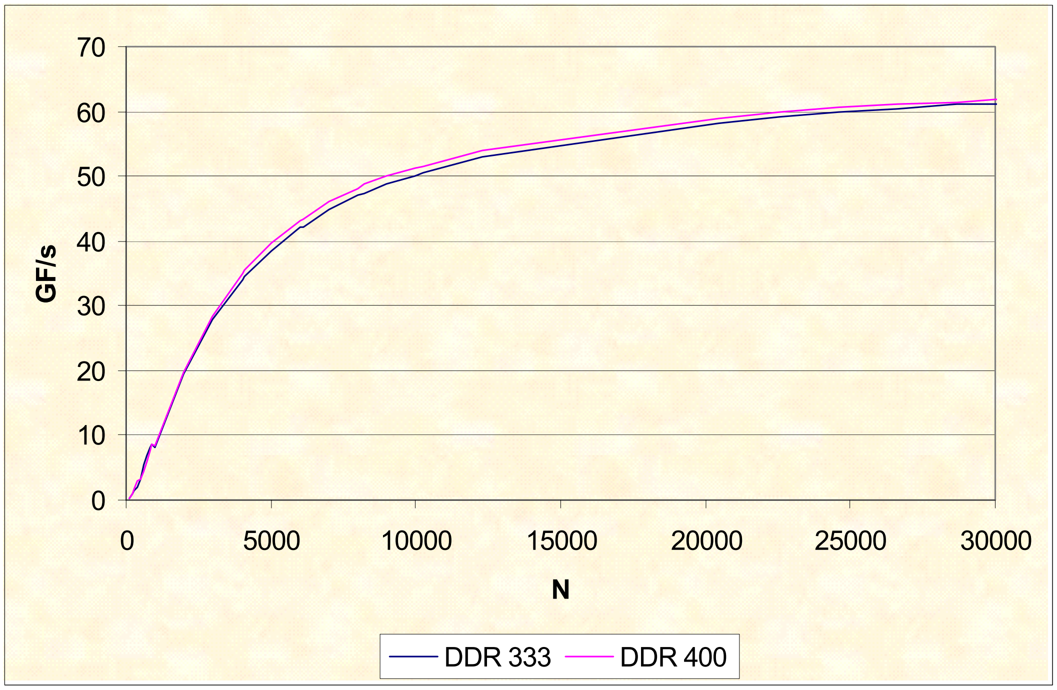

Figure 2. Linpack Sensitivity to Memory Performance

Figure 3 shows that Linpack is fairly insensitive to network performance as well, though not to the same degree as it is to memory performance. In this set of experiments three networks with widely varying performance characteristics were used in a 32-node cluster of 2.0 GHz e325 servers. The low-speed network used was Gigabit Ethernet, using the LAM implementation of MPI. The medium-speed network was Myrinet using single-port D cards and the GM library. The high-speed network was Voltaire InfiniBand, using the VAPICH MPI library. The difference in throughput between the fastest and slowest networks is a little less than a factor of ten.

The diagram clearly shows Linpack is most sensitive to network performance when the problem size is small and communicating is expensive. As the problem size increases, the amount of computation and communication also increase, but computation increases *much* faster than communication. So, in these experiments Linpack using the fastest network is faster than the slowest network by a substantial 37% on a system of 10000 equations. But when the problem size is increased to 90000 equations, the difference between the fastest and slowest network drops to only 4%. At the larger size, the efficiency of the cluster using InfiniBand is 80%, which is only a few percent below the single-server measurements. The cluster using Gigabit Ethernet is about 76.5% efficient, which is also very good, and this is in spite of its slow network.

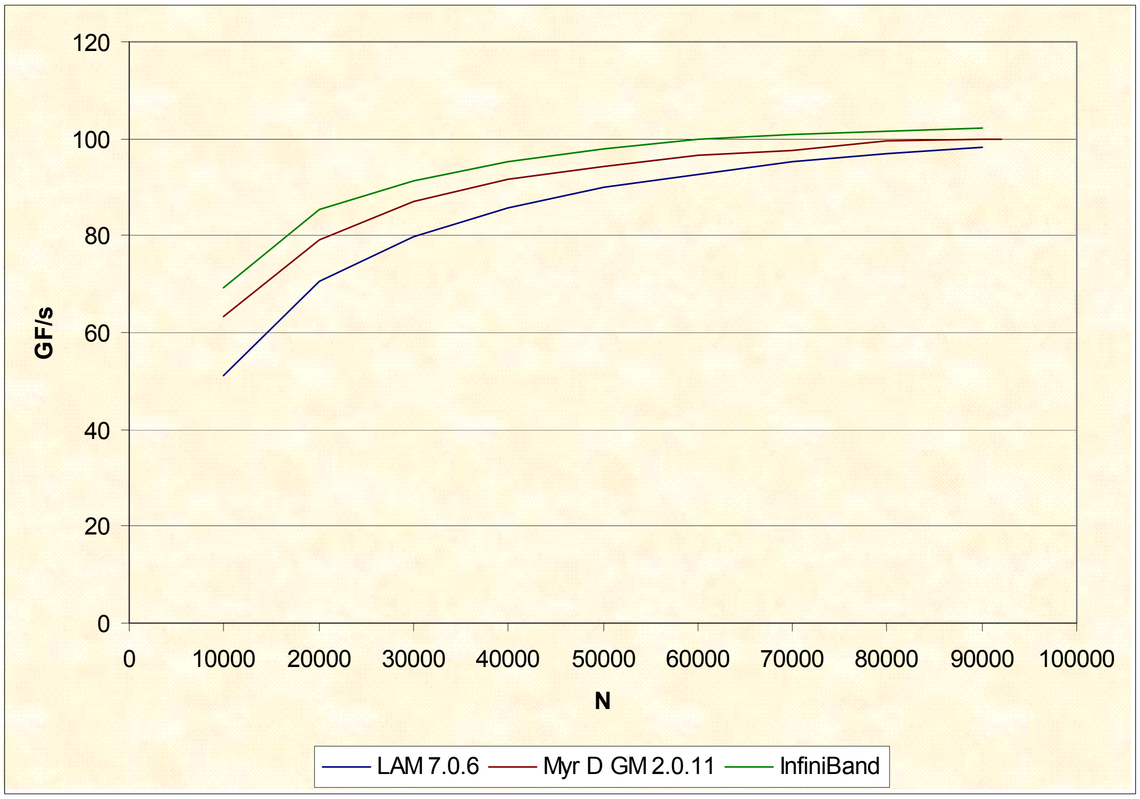

Figure 3. Linpack Sensitivity to Network Performance

3. Benchmark Tuning

Linpack uses a sophisticated algorithm for solving a system of equations on a parallel system. Parallelism is expressed through MPI processes in order to support the greatest variety of architectures.[1] In this algorithm the system is organized as a rectangular grid and the data is distributed across that grid. Nodes in the grid represent MPI processes which are usually mapped directly to processors, that is, there is one MPI process for each processor in the system. So, a 16-node cluster with 32 processors can be organized as a 1-by-32 grid, a 2-by-16 grid, a 4-by-8 grid, an 8-by-4 grid, a 16-by-2 grid, or a 32-by-1 grid of processes. In the benchmark the grid is determined by the variables *P* and *Q* in the configuration file. The product of *P* and *Q* gives the number of MPI processes to be used.

The problem to be solved, the *N* equations in *N* unknowns, is represented by a roughly *N*-by-*N* matrix that is distributed across the processes. The matrix is distributed by first partitioning it into tiles, then distributing the tiles cyclicly across the processors. The configuration variable *NB* gives the number of tiles, or blocks, to be used in each of the two matrix dimensions. Each tile is square,

1. MPI supports a distributed programming model, which can be used on clusters, Symmetric Multi-Processing (SMP) and Non-Uniform Memory Access (NUMA) systems. Other programming models, such as threads, are not able to operate on clusters or do not offer the wide-spread acceptance of MPI.

with *N*/*NB* elements on a side. The tiles are passed to each of the processors in much the same way that a person might deal a deck of cards. Distribution is done this way to level the load across all processes. Cyclic distributions tend to spread the work evenly across all processes, reducing the number of hot spots that occur.

An example of this is illustrated in Figure 4. For the sake of concreteness, assume that the matrix represents a system of 128 equations in 128 unknowns, that *NB* is eight, *P* is two and *Q* is also two. Each tile in the matrix is 16-by-16 in size, and there are 64 tiles to be distributed across four processes. Describing this by rows, the first (blue) tile is given to the first (blue) process, the second (red) tile is given to the second (red) process, the third (blue) tile wraps around the row of processes, so it is given to the first (blue) process. The first row is distributed in this manner until the row is exhausted. The second row of tiles is distributed in a similar fashion across the second row in the process grid. The third row of tiles wraps around to the first row in the process grid, but is otherwise the same as the first two. Distribution continues in this manner, wrapping around as needed, until all of the tiles have been assigned to a process.

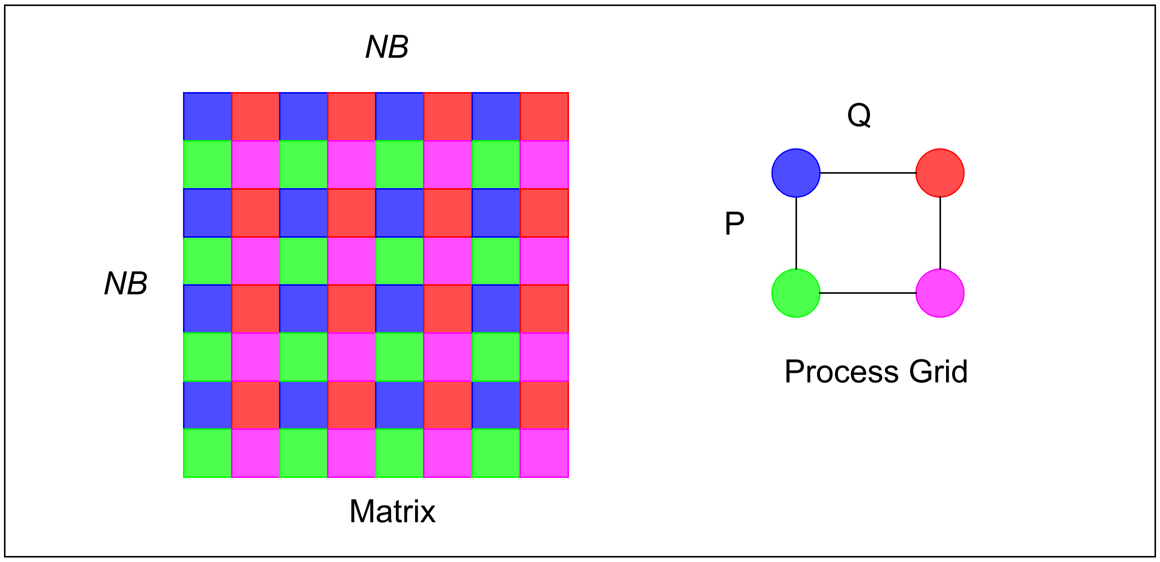

Figure 4. Data Distribution in Linpack

Linpack tends to prefer process grids that are square, or very nearly square. A simple experiment was performed that illustrates this. A 16-node, 32-processor cluster of e326 servers was used to compare different aspect ratios of the process grid. The results are shown in Figure 5. As can be seen in the graph, the 4-by-8 grid outperformed all others, with the 8-by-4 grid coming in second. As the grid became less square, the performance decreased. Also note that Linpack does show a preference that *P* be the smaller of the two values, when *P* and *Q* are not the same.

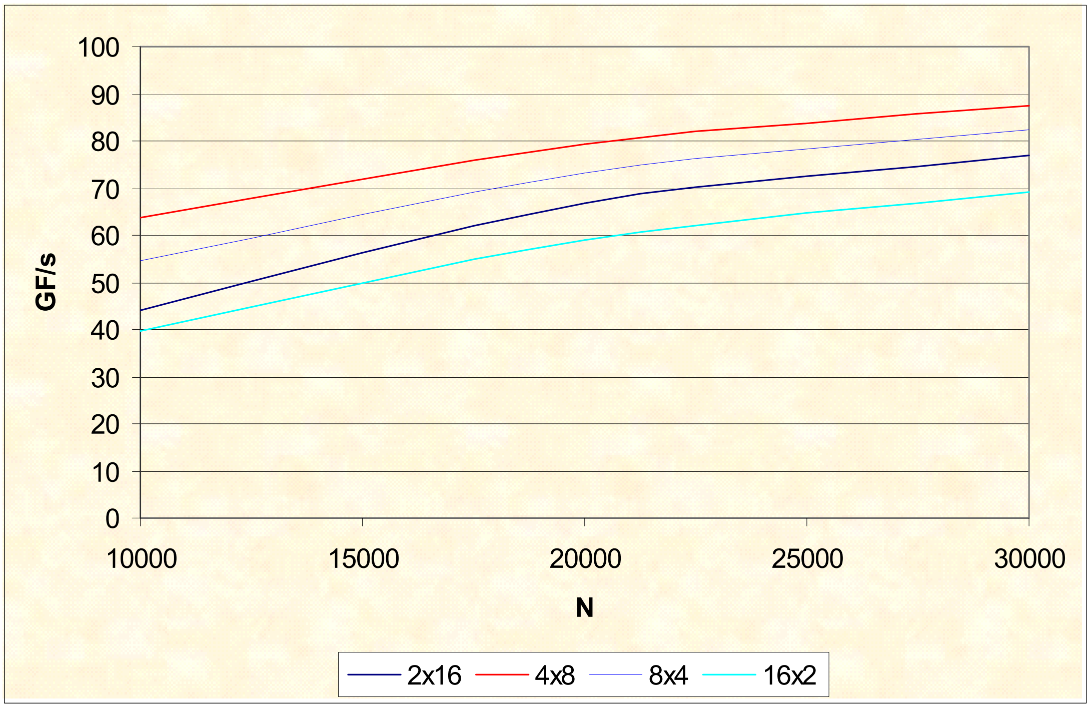

Figure 5. 16-Node, 32-Processor 2.0 GHz e325 (LAM MPI)

We have already seen how increasing *N*, the size of the problem, increases the benchmark performance.[1] Based on how data is distributed we can also see that there is a complex interaction between problem size, tile size, and the size of the process grid. The ideal situation would be that every tile would be exactly the same size, and every process would receive exactly the same number of tiles. Processes receive different numbers of tiles whenever either *P* or *Q*, or both, do not evenly divide *NB*. For example, if *P* is four and *NB* is five, the first process will receive one more tile than the other three in that row, giving it more work to do while the other processes are idle. The tiles are not all the same size whenever *N* is not evenly divisible by *NB*. When remainders occur, the remainder must be placed in smaller tiles, again, creating a load imbalance.

There are other interactions, as well, that are more difficult to predict. For example, partial results must be shared among the processors throughout different stages of the computation, so the number and size of the tiles affect the communication costs. Smaller tiles require more and shorter messages to be sent, which can be a disadvantage for systems with high message passing latencies. Furthermore, communication costs can be affected by how tiles fit into transmission packets. For example, standard Ethernet packets are approximately 1500 bytes in size. Tiles that are slightly smaller require a single packet, whereas slightly larger tiles require two packets, which is noticeably slower. The selection of tile size may affect the factorization in other ways as well.

1. Of course, when the problem size exceeds the physical memory available to the benchmark, process paging occurs and performance drops off dramatically.

Figure 6 plots Linpack performance by number of tiles on a 2.6 GHz e326. Each line in the graph shows the performance for a specific problem size. From the graph it is clear that very small values of *NB* yield very poor performance. But performance rapidly reaches a plateau where it settles into a cycle that seems to be related to the number of processes being used.

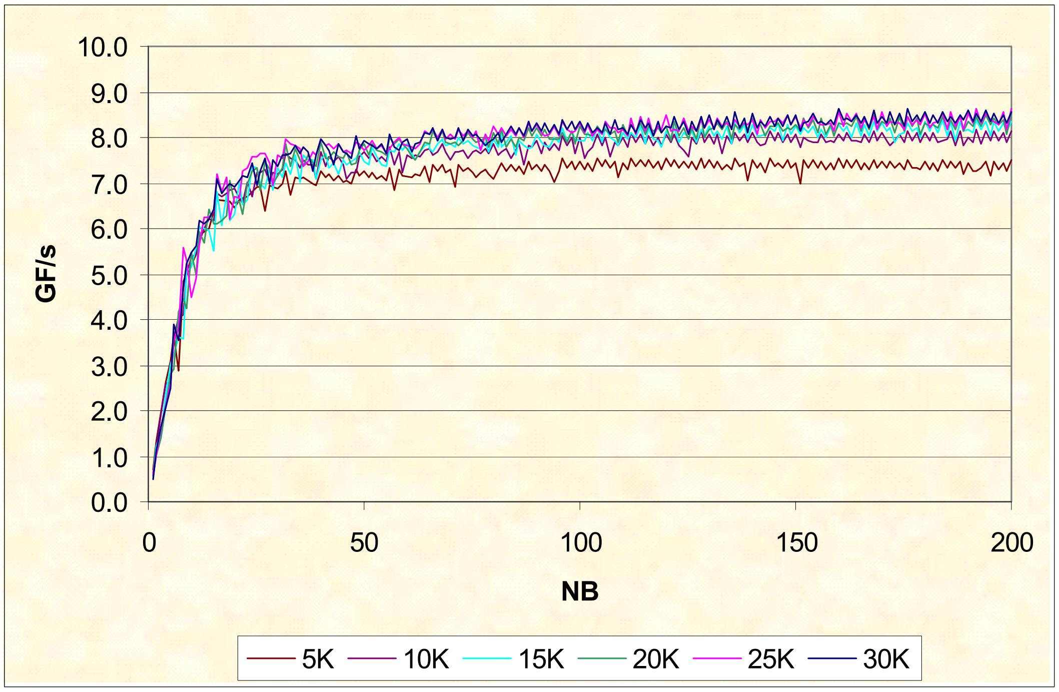

Figure 6. Performance of Multiple Problem Sizes by *NB*

A more detailed graph is shown in Figure 7. In this plot it can be seen that odd values of *NB* generally have lower performance than even values, but the even values of *NB* also vary noticeably from each other in performance. In one attempt to find patterns in the complexity, a Fourier analysis was performed on various samples from these runs, but no obvious pattern emerged. The implication is that an intelligent choice of *P*, *Q*, *N*, and *NB* will bring the benchmark performance close to the maximum performance, within a few percent, but the maximum performance can only be found through experimentation.

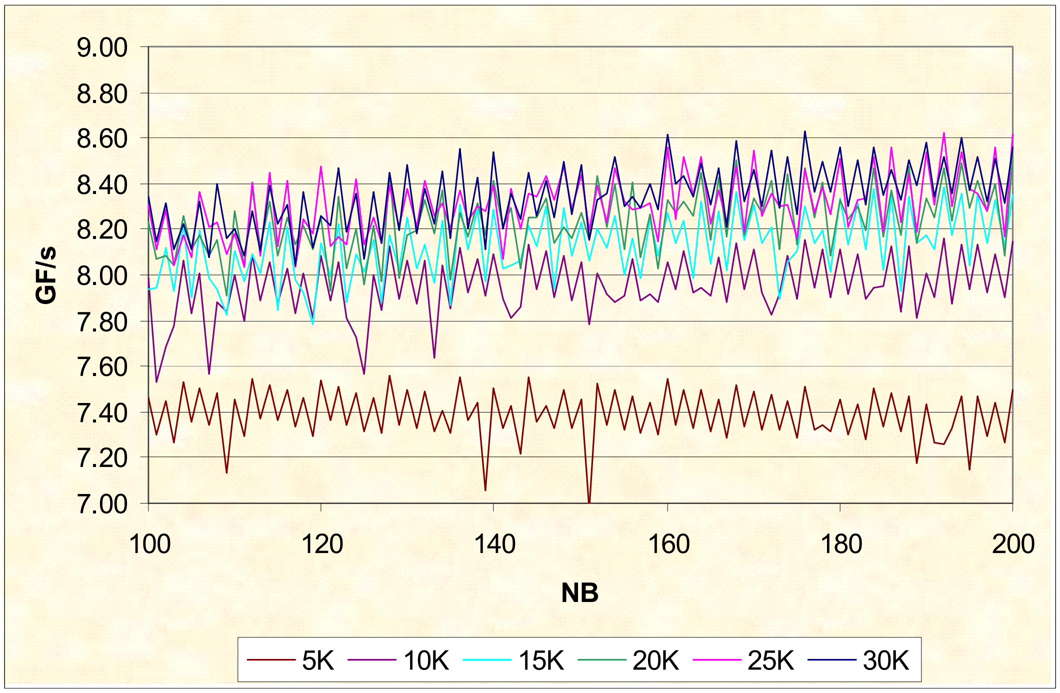

Figure 7. Performance of Multiple Problem Sizes by *NB*

4. Conclusions

Linpack efficiency was extremely similar over all systems and processors that were tested. This suggests that Linpack performance can be estimated to a fair degree of accuracy over systems based on the x86-64 and EM64T architecture.

Performance increased (nearly) monotonically with problem size. Larger problems take longer to run, but they give better performance up to the point where the benchmark begins to page. It is also the case that Linpack does *not* accurately reflect system performance when the problem size is below 10000. So older versions, which use problem sizes of 100 or 1000 equations, should not be used.

Linpack performance is highly sensitive to processor floating-point vector performance, which depends heavily on the number of processor cores and the processor clock frequency. Core for core, the processor with the higher clock frequency generally wins. In a cluster environment its performance is somewhat sensitive to network performance, but that sensitivity diminishes as the problem size increases. The benchmark is less sensitive to cache size, and almost completely insensitive to memory performance.

The benchmark tuning parameters are critical for showing good performance. *P* and *Q* should be as close to equal as possible, but when they are not equal, *P* should be less than *Q*. (*P* multiplied by *Q* gives the number of MPI processes to be used.) *NB* should be an integer multiple of *P* times *Q*, and *N* should be an integer multiple of *NB*. Values of *NB* between 100 and 256 seem to do well, but whether values above 256 will do better or worse requires further investigation. Parameters that meet the above criteria generally perform better than those that don’t, but finding the best combination among those that do, particularly finding the best value for *NB*, requires extensive experimentation.

5. References

[1] Jack J. Dongarra, Piotr Luszczek, and Antoint Petitet, “Linpack Benchmark: Past, Present, and Future,” http://www.cs.utk.edu/~luszczek/articles/hplpaper.pdf.
[2] 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.
[3] J.J Dongarra, Cleve B. Moler, G. W. Stewart, Linpack User's Guide, Society for Industrial & Applied Mathematics, June 1979.
[4] Linpack, www.netlib.org/linpack/index.html.
[5] 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.
[6] 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.
[7] Kazushige Goto, “High-Performance BLAS,” www.cs.utexas.edu/users/flame/goto/.
[8] Douglas M. Pase and James Stephens, “Performance of Two-Way Opteron and Xeon Processor-Based Servers for Scientific and Technical Applications,” ftp://ftp.software.ibm.com/eserver/benchmarks/wp_server_performance_030705.pdf, IBM, March 2005. Also published under 2005 LCI Conference, http://www.linuxclustersinstitute.org/Linux-HPC-Revolution/Archive/PDF05/14-Pase_D.pdf.