Mathematical Modelling and Analysis 2005. Pages 525–533
Proceedings of the 10th International Conference MMA2005&CMAM2, Trakai
 ISBN 9986-05-924-0

AN EVALUATION OF THE PERFORMANCE OF VGTU PC CLUSTER

V. STARIKOVIČIUS, R. ČIEGIS and G. ŠILKO

Vilnius Gediminas Technical University

Saulėtekio al. 11, LT-10223 Vilnius, Lithuania

E-mail: rc@fm.vtu.lt; vs@sc.vtu.lt; gs@sc.vtu.lt

Abstract. In this work we have tested the performance of VGTU PC cluster "Vilkas" in order to identify the main characteristics of the system and to find for what type of computational jobs the cluster is well adopted. HPC Challenge suite of benchmarks was used for that purpose. The results of tests are presented and discussed. We analyze the influence of different compilers and hyper-threading technology on the performance of the cluster. Some recommendations for future upgrade of the system are given.

Key words: PC clusters, evaluation of performance, benchmarks

1. Introduction

The availability of increasingly powerful commodity microprocessors and high-speed networks is making clusters of computers the fastest growing approach for building cost-effective high-performance parallel computing platforms. Since 2000, the number of clusters in the Top 500 list (http://www.top500.org) has grown from 11 (2,2%) to 304 (60,8%) – and this is only in the richest part of HPC world.

PC clusters are built from commodity-of-the-shelf personal computers connected together through some fast but standard network using open source software. Thus PC clusters are a good alternative to specialized supercomputers when we consider the price to performance ratio. For many sites PC clusters offer affordable tool for supercomputing.

The main drawback of PC clusters is a reduced communication capacity between processors. Thus developing parallel algorithms for such systems we should take into account the disbalance in computational and communication powers of PC clusters. Therefore it is important to test the performance of the cluster in use and to define for what type of computational jobs it is well adapted (see, e.g. [1]).

Two suites of benchmarks are most popular for testing the performance of parallel computers. HPC Challenge benchmark suite was released by the DARPA HPCS program [6]. It provides seven different benchmarks, that bound the performance of many real applications as a function of memory access characteristics. The NAS benchmark suite comprises five kernels and three applications, representing a specific part of CFD applications [2]. In this paper we restrict to the analysis of results obtained for HPC Challenge benchmark.

Cluster configuration

Vilkas is a cluster build from 16 personal computers at Vilnius Gediminas technical university. Each node has Intel Pentium 4 processor running at 3.2 GHz with 16 KB L1 cache and 1 MB of L2 cache, 1 GB of DDR-RAM (Double Data Rate RAM), 800 MHz Front Side Bus, and integrated Intel Pro/1000 CT Gigabit Ethernet card. Processor supports Intel Hyper-Threading technology. Intel's Hyper-Threading technology makes a single physical processor appear as two logical processors. All computers are connected together with Gigabit Ethernet switch.

Cluster was installed using "Rocks Cluster Distribution" v 3.3.0 (Makalu). It is based on `RedHat Linux Enterprise 3.0` with kernel version 2.4.21-20. All installed software is free and public except for the commercial Intel compilers (Fortran and C/C++).

The benchmark programs were compiled with GNU or Intel compilers and LAM-MPI implementation of MPI [3]. A more detailed information about *Vilkas* cluster is presented at `http://vilkas.vtu.lt`.

The rest of the paper is organized as follows. In Section 2, we describe the benchmarks from HPC Challenge suite and present the performance results obtained on our cluster together with their analysis. Finally, Section 3 concludes the paper.

2. HPC Challenge Benchmark Suite

A list of Top 500 parallel computers is compiled twice per year by using results of *High Performance Linpack* (HPL) benchmark, which measures the floating point rate of execution for solving a linear system of equations of order n. The operation count for Linpack test is $\frac{2}{3}n^3 + \frac{1}{2}n^2$, i.e. one order higher than the number of data involved in communication. Such test is useful for testing the peak performance rate of the parallel system, but many real applications have memory access patterns much more challenging than HPL algorithm.

Thus the HPC Challenge benchmark suite was released by the DARPA HPCS program [6]. It provides seven different benchmarks that bound the performance of many real applications as a function of memory access characteristics [5]. This suite stresses not only processors, but also the memory

system and the interconnect. It gives wider range of metrics for comparison of different HPC architectures and for evaluation of their suitability for specific real-world applications.

2.1. HPL benchmark

HPL solves a system of linear equations of order n:

$$\mathbf{A}X = F, \quad \mathbf{A} \in \mathbb{R}^{n \times n}, \quad X, F \in \mathbb{R}^n$$

by computing LU factorization with row partial pivoting. This test stresses the floating point performance of a parallel system. However, a system's communication bandwidth also significantly influences the overall Linpack performance.

To understand better the impact of different compilers and Hyper-Threading on general performance of cluster, we have tested first a single node. Figure 1 shows the HPL performance results on a single node running code compiled with different compilers with different number of processes.

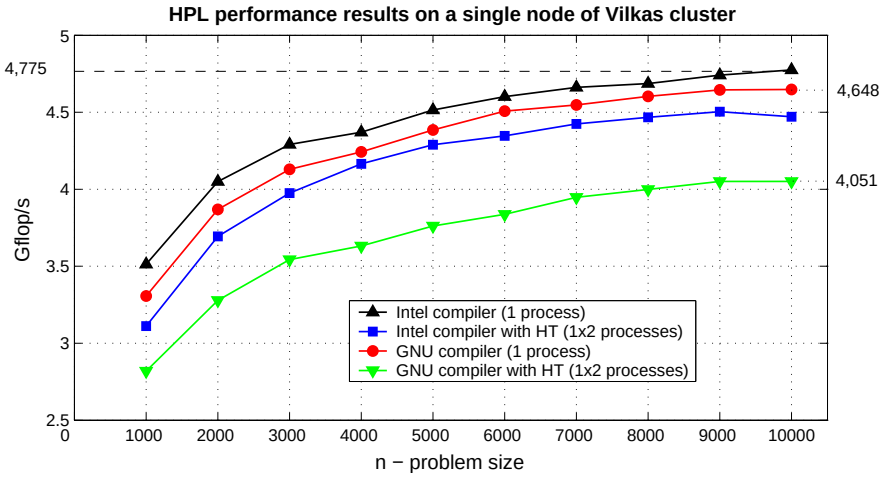

Figure 1. The HPL performance results on a single node.

In each case we have conducted a series of HPL runs from small problem size to large. As it is known for Linpack test, the larger size of the problem specified for execution, the better is the performance of the system in all cases. The results show that utilization of two logical processors given by Hyper-Threading technology by running the HPL benchmark with 2 processes decreases the performance of the single node around 5% for Intel compiler and around 13% for GNU compiler.

Such a good performance can be explained by the fact that the HPL benchmark has quit simple flow of instructions, which keeps processor's floating-

point functional units including the SSE2 almost fully utilized during the execution. This leaves very little room for improving the Linpack performance by using 2 processes with Hyper-Threading to increase CPUs' resources utilization. On the contrary, besides the overhead of running two processes instead of one, those two processes can start compete for processor resources slowing each other [4]. It seems that Intel compiler deals with that danger better than GNU compiler, what is to be expected obviously. However, because of discussed specifics of HPL, Intel compiler was only around 2,5% better than GNU reaching 74.6% of theoretical peak performance of single node (which is equal to $2 \times 3.2 = 6.4$ Gflop/s due to SSE2).

Next, we have performed similar series of HPL runs on 16 nodes. The results are presented in Figure 2. They show that the main trend has been changed. Doubling of processes using the Hyper-Threading improves the performance of the cluster. The obvious explanation is that now processes have to perform communications and sometimes wait for the data. This creates the possibility for overlapping of some communications by computations by running two processes (threads) on a single physical processor with enabled Hyper-Threading. Such overlapping will increase the utilization rate of the processors' execution resources. Therefore, the total performance of the whole cluster can be improved.

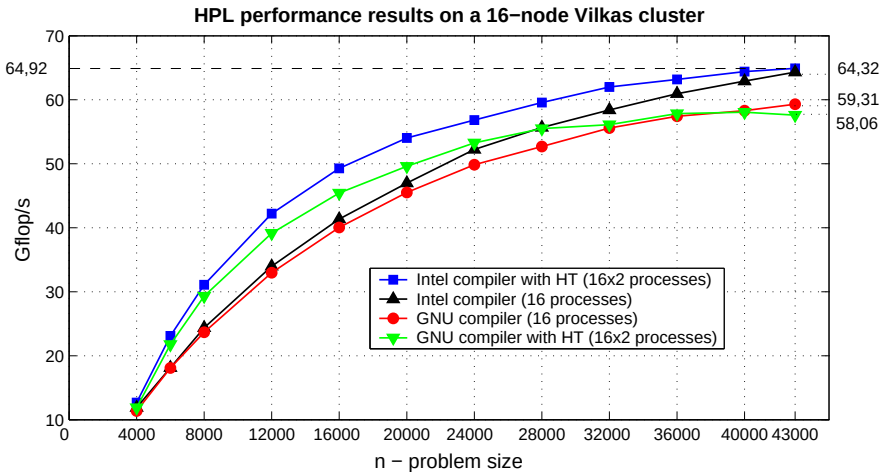

Figure 2. The HPL performance results on 16 nodes.

Again, the results show that the code produced by the Intel compiler is superior than the code produced by the free GNU compiler. However, approaching the maximal problem size, this positive effect of Hyper-Threading seems to be compensated by the negative effects (for example, contention for the memory).

Top performance with Intel compiler was around 8.6% better than with GNU compiler reaching 63.4% of theoretical peak performance of 16-node cluster (102.4 Gflop/s). Reaching 60% of theoretical peak performance is estimated as a very good result for a cluster [4].

2.2. DGEMM benchmark

This benchmark measures the floating point rate of execution of double precision real matrix-matrix multiplication. The exact operation performed is

$$\mathbf{C} = \beta\mathbf{C} + \alpha\mathbf{AB}, \quad \mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{R}^{n\times n}, \quad \alpha, \beta \in \mathbb{R}.$$

The operation count for such operation is $2n^3$. The benchmark is run in embarrassingly parallel manner, when all nodes perform multiplication independently and at the same time, i.e. without any communication (but shared-memory effects might occur on SMP nodes). The arithmetic average rate is reported. Thus this test stresses the floating point performance of separate nodes of a parallel system.

The result obtained with Intel compiler is 5.182 Gflop/s (or 82.912 Gflop/s for a whole cluster of 16 processors), and with GNU compiler - 5.017 Gflop/s. It is clear that DGEMM has even simpler work flow than HPL and higher utilization of SSE2 units, therefore the top performance is even closer to the theoretical peak performance of a single node, i.e. 81.0%. The difference between two compilers is 3.2%.

2.3. STREAM benchmark

STREAM is a simple synthetic benchmark that measures sustainable memory bandwidth (in Gbytes/s) and computation rate for four simple vector kernels:

$$C \leftarrow A, \quad B \leftarrow \alpha C, \quad C \leftarrow A + B, \quad A \leftarrow B + \alpha C,$$

where $A, B, C \in \mathbb{R}^n$, $\alpha \in \mathbb{R}$. It is implemented in a very straightforward way without sophisticated optimizations. STREAM thereby produces results that correspond to memory bandwidth expected from an ordinary user application. It is run in embarrassingly parallel manner - all computational nodes perform the benchmark at the same time, the arithmetic average rate is reported.

The obtained results show no difference between compilers. Benchmark version included in HPCC Suite gives to the compiler even less control (and possibilities to optimization) than original one [5]. Results presented in Table 1 are better or comparable with results shown by many systems based on Intel Xeon, Intel Itanium 2, AMD Opteron processors (see [6], where extensive results for different computer architectures are presented). However, Pentium 4 on 800 MHz FSB with DDR400 memory (theoretical peak bandwidth 6.4 Gbytes/s) cannot even compare with Cray and NEC systems.

Table 1. STREAM benchmark: memory bandwidth results (in Gbytes/s).

Copy	Scale	Add	Triad
2.598	2.562	3.068	3.206

2.4. Latency and bandwidth benchmark

This benchmark measures latency and bandwidth of two different communication patterns. First, it measures the single-process-pair latency and bandwidth. The benchmark is designed to measure the performance of system interconnection network. So, different processes are assigned to different processors. Ping-pong communication is used on a pair of processes. Many different pairs are investigated and the maximal, minimal and the average values of latency (in *usec* = *micro* second = 10^{-6} second) and bandwidth (in Gbytes/s) over all pairs are reported. This test uses MPI standard blocking *send* and *receive* routines.

Next, the benchmark measures the parallel all-processes-in-a-ring latency and bandwidth. Each process sends and receives a message from its left and right neighbours in parallel. Two types of rings are investigated: naturally ordered ring, which is ordered by the process ranks (or MPI ranks), and randomly ordered ring. The communication is implemented with MPI standard non-blocking receive/send and with MPI_Sendrecv in both directions in the ring. The fastest result is used. Average values of latency and bandwidth per process are reported. The results of our measurements are in the Table 2.

Table 2. Latency (in usec) and bandwidth (in GBytes/s) benchmark results.

Parameter	Ping-pong Min	Ping-pong Max	Ping-pong Average	Nat. ordered Ring	Rand. ordered Ring
Latency	61.438	125.131	80.232	186.307	188.475
Bandwidth	0.050741	0.097487	0.063572	0.049580	0.046994

Comparison with results of parallel systems presented in [6] shows a quit poor performance of our interconnect. It is clear that for clusters it is difficult to compete with specially build parallel architectures of IBM with High Performance Switch, Cray with 2 or 3D torus, or SGI with Numalink. We note that some interconnects of clusters such as Myrinet or Infiniband also show very good results. However, they are in the different price level comparing with Gigabit Ethernet.

Our Gigabit Ethernet interconnect with economy switch (D-Link DGS 1224T) shows results comparable in the bandwidth with the results of clusters with Gigabit Ethernet [6]. The bandwidth of randomly ordered ring is even

larger. However, this is most probably due to smaller number of nodes in our cluster. The noticeable difference is in the decrease of bandwidth between randomly and naturally ordered rings: 5.2% in our case and 33.4%, 59.9% for Cisco and HP switches accordingly. However, our latencies are up to 4 times larger [6]. Such latency will create a performance bottleneck on our cluster for applications with a large number of small communications.

2.5. PTRANS benchmark

PTRANS implements a parallel matrix transpose. The exact operation performed is

$$\mathbf{A} \leftarrow \mathbf{A}^T + \mathbf{B},$$

where $\mathbf{A} \in \mathbb{R}^{n \times n}$ is random $n \times n$ matrix, $\mathbf{A}^T$ is its transpose, and $\mathbf{B}$ is another random matrix. The number of basic operations is the same as the number of transferred data. Matrices are stored using two-dimensional block-cyclic distribution.

During parallel execution of this kernel, pairs of processors exchange large messages simultaneously. It returns the data transfer rate (in Gbyte/s) which is calculated by dividing the size of n^2 matrix entries by the time it took to perform the transpose. This benchmark is a useful test of the total communications capacity of the system interconnect.

The results of this benchmark depend on matrix size n and block size r. As it can be expected, no noticeable influence of the compiler was observed. With $n = 21500$ (close to the maximal available memory) and $r = 140$, we have obtained 0.530 Gbyte/s or 0.0331 Gbyte/s per process. These results are comparable with the results of systems with Gigabit Ethernet interconnects [6]. Among supercomputers the leaders are Cray (874.899 and 0.168 Gbyte/s) and NEC (25.183 and 4.197083 Gbyte/s) systems.

Interestingly, when we doubled the number of processes using Hyper-Threading, the overall performance has improved till 0.90675 Gbyte/s with only slight decrease of the data transfer rate per process - 0.0283 Gbyte/s. The amount of data sent over the network in this case is the same as in the first, the number of communications is bigger, and some data have to be copied locally using shared memory. However, it seems that Hyper-Threading technology was able to schedule communications in such a way that some useful work was done by one part of processes when the rest of them were waiting for the data.

2.6. RandomAccess benchmark

RandomAccess measures the rate at which the computer can update pseudo-random locations of its memory - this rate is expressed in billions (giga) of updates per second (GUP/s). This benchmark has three versions. The single node version runs the code locally on a randomly chosen processor. No explicit communication is performed and so the performance of the local memory subsystem is revealed. The embarrassingly parallel version runs the code locally

on each node. No explicit communication is performed (but shared-memory effects might occur on SMP nodes). GUP/s per process are returned. And finally, the MPI version generates the updating sequence locally and then distributes it using all-to-all collective communication. This version of the test evaluates communication subsystem also.

It is clear from the description of the benchmark that it is difficult to expect a good performance of our cluster for the global (MPI) version of this test, because it requires a big number of small communications. Performance results of embarrassingly parallel (EP) and MPI versions are shown in Table 3.

Table 3. RandomAccess benchmark results (in GUP/s).

compiler	EP version	MPI version
Intel	0.00691798	0.00067087
GNU	0.00621339	0.00063834

However, our global results are almost identical to results reported by the clusters with Gigabit Ethernet interconnects [6]. Our local results (with EP version) show that the performance of Pentium 4 node is on the level of nodes with Intel Xeon, Intel Itanium 2, AMD Opteron processors. For this test the difference between compilers can be observed: 10.2% for EP version and 4.9% for MPI version.

2.7. FFT benchmark

FFT measures the floating point rate of execution of double precision complex one-dimensional Discrete Fourier Transform. This benchmark also has three versions. The single node version runs the code locally on a randomly chosen processor. Embarrassingly Parallel version performs the same test but in embarrassingly parallel fashion - the code is run locally on each processor. No explicit communication is performed (but shared-memory effects might occur when several processes run on the same node). Gflop/s per process are returned. Global (MPI) version performs the same test but across the entire system by distributing the input vector in block fashion across all the nodes. MPI version returns the global rate in Gflop/s. The results obtained on our cluster are shown in Table 4.

The difference between compilers is 7.2% for local (EP) version and 2.8% for global (MPI) version. The doubling of processes using Hyper-Threading improves the performance of our cluster. In general, our FFT performance results are also on the level of clusters reported in [6].

Table 4. FFT benchmark performance results (in Gflop/s).

Compiler	EP version	MPI version	EP with HT	MPI with HT
Intel	0.510068	1.8737	0.308627	2.08884
GNU	0.473133	1.8216	0.260984	1.90465

3. Conclusions

The results showed by *Vilkas* cluster are on the level of results reported by the other clusters with similar interconnect (Gigabit Ethernet). We expect a good performance on our cluster of applications with the amount of computations much bigger then the number of data which must be send. More precisely, the number of communications and not their size will create the problems due to the big latency time on our cluster. Hyper-Threading can improve the performance of our cluster for applications with considerable amount of communications by increasing the utilization of processors' execution units. And finally, we plan to extend this work with the results of NAS Parallel Benchmarks and their analysis.

References

[1] K.-J. Andersson, D. Aronsson and P. Karlsson. An evaluation of the system performance of a Beowulf cluster. Technical Report 2001-4, LIU, Sweeden, 2001.
[2] NAS Parallel Benchmarks. http://www.nas.nasa.gov/software/npb/.
[3] W. Gropp, E. Lusk and A. Skjellum. *Using MPI: portable parallel programming with the message-passing interface*. The MIT Press, Cambridge, Massachusetts, London, 1995.
[4] T. Leng, R. Ali, J. Hsieh, V. Mashayekhi and R. Rooholamini. An empirical study of hyper-threading in High Performance Computing clusters. Technical report, 2002.
[5] P. Luszczek, J.-J. Dongarra, D. Koester, R. Rabenseifner, B. Lucas, J. Kepner, J. McCalpin, D. Bailey and D. Takahashi. Introduction to the HPC Challenge benchmark suite. Technical report, 2005.
[6] HPC Challenge Benchmark Suite. http://icl.cs.utk.edu/hpcc/.