# Bits & Bytes

No. 177 RZG Computer Bulletin July 2004

Computing Center of the Max Planck Society and the Institute for Plasma Physics*

## Xeon-based Linux cluster vs. IBM Regatta for small jobs

The IBM Regatta based supercomputer with High Performance Switch ("Federation Switch") is heavily overloaded, especially with large jobs up to 512 processors. This causes real bottlenecks for rather small jobs requiring only up to 32 processors. Therefore benchmarks have been carried out testing how well current Intel based Linux systems are suited to host parallel 16-processor jobs. Five codes have been selected from the spectrum of applications on the Regatta system as sample codes with different characteristics with respect to the parallelization strategy and the amount of communication involved. On the Linux side up to 8 nodes of a Blade Center with a Gigabit-Ethernet connection are used. Each blade is provided with two 2.8-GHz-Xeon processors and 4 GB of memory.
We suggest three different scenarios how to run the two processors on each blade: for memory intensive jobs occupying more than 50 % of the allocatable physical memory it is appropriate to use only one processor (case I); for two processes which can run simultaneously on each blade both processors should be used; here we distinguish between case II of two sequential jobs and case III of two processes of a parallel job using MPI or OpenMP.
On the Regatta side we investigated the scenario of 32 processes running concurrently on the 32 processors of one node. This scenario corresponds to the efficient way how the Regatta nodes are used at the computing center. Of course, optimal performance would be achieved if a sequential job would run on an "empty" node with 31 processors idling, but this scenario is only of academic interest.
Refraining from all details the benchmark results can be summarized in the following table which shows the performance between the Linux cluster and an IBM Regatta node. For sequential 1-processor runs we used case I and for parallel 16-processor runs we used case III on the Linux cluster. Both the relative performance between the Linux cluster and the Regatta node and the speedup on the two machines are listed.

| | relative performance Xeon:Power 4 | | Speedup $t_1 : t_{16}$ | |
|---|---|---|---|---|
| Code | 1 proc. | 16 proc. | Xeon | Power 4 |
| CPMD | 1.8 | 0.95 | 9.1 | 17.0 |
| TORB | 1.4 | 1.0 | 11.4 | 15.3 |
| GENE9 | 0.8 | 1.0 | 10.5 | 13.5 |
| Wien2k | 2.0 | 0.5 | 2.6 | 9.8 |
| SDTrimSP | 1.6 | 0.9 | 6.3 | 11.8 |
| average | 1.5 | 0.9 | 8.0 | 13.5 |

Table 1: relative performance and speedup of the Xeon-based Linux cluster vs. IBM Regatta.

The single-processor performance of the Xeon-based Linux cluster is on the average about 1.5 times higher than that of the IBM Regatta node, while the relative performance for 16 processors is 0.9 on average. The performance of a sequential job on the Linux cluster drops by about 20 % if one proceeds as described in case II. This is due to the fact that two jobs occupying concurrently the same blade have to share the access to the memory which causes a bottleneck. However, the effect is moderate and causes on the one side the relative performance to drop from 1.5 to 1.3 on average. On the other side, the speedup of the Linux cluster increases from 8.0 to 10.0 on average. It increases because for case II the memory bottleneck is already part of the one-processor run.
Note, however, that for very communication-intensive codes like Wien2k the IBM Regatta is superior to the Linux cluster when using 16 processors, as the high amount of communication can be handled much better by the IBM Regatta system than by the Gigabit-Ethernet network of the Linux cluster. Thus, for communication-intensive codes faster interconnects have to be considered, and tests are in progress.
All in all, the benchmark results show clearly that for small jobs using 16 processors the Linux cluster can be regarded as a fully adequate alternative to the IBM Regatta provided the parallel program in question is not too communication-intensive and memory-demanding (< 2 GB per processor).
A more detailed discussion on the benchmark codes and methods and on the results can be found in what follows.

*Max-Planck-Institut für Plasmaphysik, Boltzmannstraße 2, D-85748 Garching bei München, tel.: +49(89) 3299-01, e-mail: benutzerberatung@rzg.mpg.de, URL: http://www.rzg.mpg.de/
**Editorial: Dr. Roman Hatzky, Tel. -1707**

## The benchmark codes

**CPMD,** a parallelized plane wave/pseudopotential implementation of Density Functional Theory, particularly designed for ab-initio molecular dynamics. For the benchmark carried out here the MPI parallel version of the code is used.

**TORB,** a global nonlinear simulation code for the time evolution of ion-temperature-gradient driven instabilities in fusion plasmas. A $\delta f$ particle-in-cell (PIC) method is used to solve the coupled system of gyrokinetic equations. The code is very well parallelized and achieves a nearly optimal speedup on up to 512 processors on the IBM Regatta system. For the benchmark done here the communication implemented via MPI is dominated by particles exchanged between neighboring processors.

**GENE9,** a code for nuclear fusion theory in tokamaks. GENE (Gyrokinetic Electromagnetic Numerical Experiment) uses a finite-difference method to solve a nonlinear system of partial integro-differential equations. It is parallelized using MPI and OpenMP.

**Wien2k,** a quantum-chemical code for calculating electronic configurations in crystals. The code contains as a main part an eigenvalue solver which is parallelized by means of the freely available ScaLAPACK library. It is in the nature of the underlying algorithm that this part of the code is highly communication-intensive. The parallel implementation is based on the MPI communication library.

**SDTrimSP,** a Monte-Carlo code to simulate target-sputtering processes. Contrary to many other Monte-Carlo codes the amount of communication is in this case not negligible, because the target is updated regularly which makes it necessary to sum up the partial results from all processors from time to time. But the communication, which is established via MPI, is clearly not the dominating part of the code.

All codes are written in Fortran.

## Benchmark systems and methods

The systems used for the benchmarks are rather small. They consist of up to 16 processors and have the following characteristics.

**IBM Regatta:** One dedicated p690 node with 32 (Power 4) processors at a clock rate of 1.3 GHz and 64 GB of memory is used. Therefore, the MPI communication is handled via shared memory without the high-performance communication network of the Regatta being involved. To simulate realistic conditions as many copies of a given program are started as to make all processing elements (PEs) busy. To obtain e. g. the runtime for a 4-processor program 8 copies of this program are run simultaneously. All codes are compiled with the IBM XLF Fortran compiler.

**Linux Blade Center:** Up to 8 nodes of a Blade Center with a Gigabit-Ethernet connection are used. Each blade is provided with two 2.8-GHz-Xeon processors and 4 GB of memory. The RZG supports two compilers for this architecture; to generate the benchmark codes the Intel Fortran Compiler v8.0 alternatively the Lahey/Fujitsu Fortran Compiler v6.2 together with the communication library mpich are used. Different operation modes are tested by starting one, two or even four processes per dual-processor node. The latter is an attempt to make use of the so-called hyperthreading capability of the Xeon processor, which means that two processes share the resources of one processor simultaneously and gain from a skillful pipelining mechanism.

## Results

The gathered benchmark results are presented with respect to the following aspects: the quality of the two Linux compilers, the performance of the diverse operation modes on the Linux cluster and a comparison of the performance for the two architectures.

### Comparison between the two Linux compilers

Three of the codes were tested with both Linux compilers, and the observation is that the code generated by the Intel Linux compiler is generally significantly faster than that generated with the Lahey/Fujitsu compiler. Refraining from all details the result is that the performance gain is about 1.6 with the Intel Fortran compiler. However, the Lahey/Fujitsu compiler has the better runtime diagnostic capabilities to detect e. g. uninitialized variables, array bound violations and inconsistencies in argument lists of functions and/or subroutines. A detailed comparison between both compilers in respect to performance and diagnostical capabilities can be found under www.polyhedron.com .

### Performance on the two architectures

In the following the execution times of the benchmark codes on the two architectures are presented in comparison. All data is in seconds. For the Linux cluster, only those results obtained with the Intel compiler are cited as they are the faster ones. For the IBM Regatta the execution times are given in dependence of the number of employed processors. In case of the Linux cluster the results are arranged in first order with respect to the number of occupied nodes and in second order with respect to the MPI tasks started per node.

**CPMD:**

| nodes | Linux cluster MPI tasks per node | | |
|---|---|---|---|
| | 1 | 2 | 4 |
| 1 | 67 | 53 | 56 |
| 2 | 34 | 28 | 26 |
| 4 | 17 | 13 | 14 |
| 8 | 9 | 7 | 6 |

| PEs | Regatta |
|---|---|
| 1 | 119 |
| 2 | 60 |
| 4 | 28 |
| 8 | 14 |
| 16 | 7 |

**TORB:**

| nodes | Linux cluster MPI tasks per node | | |
|---|---|---|---|
| | 1 | 2 | 4 |
| 1 | 66 | 42 | 38 |
| 2 | 33 | 21 | 19 |
| 4 | 17 | 12 | 11 |
| 8 | 9 | 7 | 6 |

| PEs | Regatta |
|---|---|
| 1 | 90 |
| 2 | 46 |
| 4 | 23 |
| 8 | 12 |
| 16 | 6 |

**GENE9:** The GENE9 code is the only one of our benchmark codes which is implemented with a mixed MPI/OpenMP programming model. The following table shows the execution time on the Linux cluster obtained with the pure MPI program, i. e. when compiling the program without the OpenMP option.

| nodes | Linux cluster MPI tasks per node | | |
|---|---|---|---|
| | 1 | 2 | 4 |
| 1 | 6.9 | 4.5 | 4.8 |
| 2 | 3.5 | 3.0 | 2.4 |
| 4 | 1.8 | 1.5 | |
| 8 | 0.9 | | |

The missing values in the table are due to the fact that the chosen test case did not allow for more than 8 MPI tasks. When OpenMP is added, the results for the Linux cluster and for the IBM Regatta, respectively, are as follows:

| MPI tasks (nodes) | Linux cluster threads per MPI task | | | IBM Regatta threads per MPI task | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 4 | 1 | 2 | 4 |
| 1 | 8.4 | 8.0 | 5.2 | 10.8 | 5.4 | 3.1 |
| 2 | 4.2 | 3.7 | 2.6 | 5.5 | 2.8 | 1.5 |
| 4 | 2.1 | 2.1 | 1.4 | 2.5 | 1.5 | 0.8 |
| 8 | 1.1 | 1.1 | 0.8 | 1.7 | 0.9 | 0.5 |

In this case, only one MPI task per node was started on the Linux cluster.

**Wien2k:**

| nodes | Linux cluster MPI tasks per node | | |
|---|---|---|---|
| | 1 | 2 | 4 |
| 1 | 14160 | 19809 | 19129 |
| 2 | 16980 | 10665 | 11637 |
| 4 | 9300 | 8100 | 7357 |
| 8 | 6000 | 5400 | 5867 |

| PEs | Regatta |
|---|---|
| 1 | 28351 |
| 2 | 20590 |
| 4 | 10244 |
| 8 | 5617 |
| 16 | 2898 |

In case of 1 processor or 1 MPI task, respectively, the sequential version of the code has been used, which works with the much faster sequential LAPACK library instead of the MPI-parallel ScaLAPACK library. This explains the poor speedup (on both architectures) when switching from 1 to 2 MPI processes.

**SDTrimSP:**

| nodes | Linux cluster MPI tasks per node | | |
|---|---|---|---|
| | 1 | 2 | 4 |
| 1 | 503 | 259 | 271 |
| 2 | 262 | 144 | 167 |
| 4 | 145 | 96 | 120 |
| 8 | 91 | 83 | 109 |

| PEs | Regatta |
|---|---|
| 1 | 846 |
| 2 | 434 |
| 4 | 230 |
| 8 | 123 |
| 16 | 71 |

## Discussion

The information included in the above data is quite manifold. As we are interested in a transfer of small jobs from the Regatta to the Linux cluster the most interesting aspects are the evaluation of the different operation modes on the Linux cluster and a comparison of the performance on the two architectures.

### The different modes on the Linux cluster

For all the benchmark codes the **use of 2 processors** in the dual-processor node is advantageous compared to using only 1 processor. The benefit obtained from the second processor varies, however, a lot depending on the code and the number of used nodes. On average, the speedup is about 1.3. This is true for pure MPI and mixed MPI/OpenMP codes. A remarkable performance gain is obtained with the SDTrimSP code. Here the performance of the sheer calculation is nearly doubled as can be seen from the two processor results where the communication is more or less negligible. The reason might be that the amount of memory required by this code is very low (about 5 MB) so that the memory bandwidth constraints arising normally when both processors are active are not an issue for this code. But after all, as none of the codes is becoming slower, it can be recommended to make use of both processors if possible.

The **hyperthreading**, however, cannot be recommended in general. Although there is an overall performance gain of about 10 %, there are also cases where an overloading of the nodes with 4 processes leads to a performance degradation (see e. g. SDTrimSP). The results obtained with the mixed MPI/OpenMP version of the GENE9 code seem to indicate that especially the OpenMP threads can benefit from the hyperthreading. However, further investigations would be necessary to confirm this assumption.

The GENE9 results show that the overhead arising from **OpenMP under Linux** is first of all quite large – compare the runtime for mixed MPI/OpenMP using 1 thread per MPI task with that of the pure MPI program with

1 MPI task per node. A second thread per MPI task which uses the second processor in the node also does not make much effect. Only the hyperthreading, i. e. 4 threads per node and MPI task, makes this mode a bit more efficient: The respective runtimes are comparable with those of the pure MPI program on the same number of nodes, but they are not better. All in all, from the performance aspect OpenMP on the Linux cluster is not convincing up to now. It can, however, always be used as a developing platform for the Regatta.

### Comparison between the two architectures

In order to compare the two architectures the runtimes for 1 and 16 processors, respectively, are extracted from the above data. In case of various possibilities the best result for the respective processor number has been chosen. Note that the 1-processor result on the Linux cluster refers to the case I where only one processor of the dual-processor node is busy. The use of the second CPU would lower the performance by about 20 %. From the runtimes the relative performance and the speedup can be calculated. We obtain the following results:

| | Execution times [s] | | | |
|---|---|---|---|---|
| | 1 processor | | 16 processors | |
| Code | Xeon | Power 4 | Xeon | Power 4 |
| CPMD | 67 | 119 | 7.4 | 7.0 |
| TORB | 66 | 90 | 5.8 | 5.9 |
| GENE9 | 8.4 | 10.8 | 0.8 | 0.8 |
| Wien2k | 14160 | 28351 | 5400 | 2898 |
| SDTrimSP | 519 | 840 | 83 | 71 |

| | relative performance Xeon:Power 4 | | Speedup $t_1 : t_{16}$ | |
|---|---|---|---|---|
| Code | 1 proc. | 16 proc. | Xeon | Power 4 |
| CPMD | 1.8 | 0.95 | 9.1 | 17.0 |
| TORB | 1.4 | 1.0 | 11.4 | 15.3 |
| GENE9 | 0.8 | 1.0 | 10.5 | 13.5 |
| Wien2k | 2.0 | 0.5 | 2.6 | 9.8 |
| SDTrimSP | 1.6 | 0.9 | 6.3 | 11.8 |
| average | 1.5 | 0.9 | 8.0 | 13.5 |

From the sequential or 1-processor runs we learn that the **compute power** of the Xeon processor is on average about 1.5 times higher than that of the Power 4 processor. The outstanding value of 2.0 for the Wien2k code results from the excellent LAPACK eigenvalue solver implementation contained in the MKL library. Why the GENE9 code is below average could not be clarified up to now.

The **relative performance** between the 16-processor Linux system and the 16-processor IBM Regatta system is 0.9 on average. Note, however, that when neglecting the Wien2k code the mean relative performance increases to 0.96, i. e. the two systems are mainly of the same performance. That the Wien2k code is that much below average is due to the high amount of communication which can be handled much better by the IBM Regatta system than by the Gigabit-Ethernet network of the Linux cluster.

This latter point is also reflected in the table for the **speedup**. The speedup achieved with 16 processors is on average 8.0 for the Linux cluster and 13.5 for the IBM Regatta. Especially for the Wien2k code, however, the speedup obtained with the Linux cluster is very low, namely 2.6, compared to 9.8 on the Regatta.

## Conclusion

All in all, the benchmark results show clearly that for small jobs using 16 processors the Linux cluster can be regarded as a fully adequate alternative to the IBM Regatta provided the parallel program in question is not too communication-intensive. For smaller numbers of processors the Linux cluster is even advantageous. With increasing number of processors the better single-processor performance of the Linux cluster is compensated more and more by the slower communication network, until for about 16 processors the breakeven point is reached and the two systems perform equally well.

Renate Dohmen, Roman Hatzky
Jakob Pichlmeier, Reinhard Tisma