John von Neumann Institute for Computing

Low-level Benchmarking of a New Cluster Architecture

Norbert Eicker, Thomas Lippert

published in

Parallel Computing: Architectures, Algorithms and Applications ,
C. Bischof, M. Bücker, P. Gibbon, G.R. Joubert, T. Lippert, B. Mohr, F. Peters (Eds.),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **38**, ISBN 978-3-9810843-4-4, pp. 381-388, 2007.
Reprinted in: Advances in Parallel Computing, Volume **15**,
ISSN 0927-5452, ISBN 978-1-58603-796-3 (IOS Press), 2008.

http://www.fz-juelich.de/nic-series/volume38

Low-level Benchmarking of a New Cluster Architecture

Norbert Eicker[1] **and Thomas Lippert**[1,2]

[1] Jülich Supercomputing Centre, Forschungszentrum Jülich
52425 Jülich, Germany
E-mail: {n.eicker, th.lippert}@fz-juelich.de

[2] Department C, Bergische Universität Wuppertal
42097 Wuppertal, Germany

JULI is a first of its kind project at Forschungszentrum Jülich accomplished during 2006. It was carried out to develop and evaluate a new cluster architecture. Together with the companies IBM, QLogic and ParTec a state-of-the-art cluster system was designed based on PPC 970MP CPUs and a this time novel PCIe version of the InfiniPath interconnect. Together with the ParaStation cluster management system these components guaranteed an effortless handling of the system resulting in start of full production by end of 2006.

We present results of synthetic, machine-oriented benchmarks shedding light on the capabilities of JULI's architecture. Problems revealed include the latency of the inter-node communication which suffers from many stages between CPU and host channel adapter (HCA). Furthermore, general issues like the memory-bandwidth bottlenecks of the dual-core processors—which will become more severe on upcoming multi-core CPUs—are discussed.

1 Introduction

The JULI[a] project is the first in a series of investigations carried out at the Zentralinstitut für Angewandte Mathematik (ZAM) of the Forschungszentrum Jülich (FZJ). The mission of these projects is to develop and evaluate new state-of-the-art cluster platforms to prepare the upgrade of FZJ's general purpose supercomputer system in 2008.

Following this philosophy, the JULI project aims at integrating very compact compute hardware with lowest power consumptions, smallest latency interprocessor communication, and best possible process management systems. As project partners from industry we have chosen IBM Development Lab Böblingen (Germany)[1] for node hardware and various software components, QLogic[2] as contributor of the InfiniPath interconnect and corresponding MPI library and ParTec[3] with their ParaStation cluster middleware.

A first meeting for preparation of JULI was held in November 2005 only shortly before the official start begin of 2006. A first prototype of the project's hardware solution was presented at IBM's booth during ISC 2006 in Dresden. Already mid of August a first version of the system has been installed in Jülich. Accompanying an extensive benchmarking period, parts of the hardware have been upgraded for performance optimization. In December 2006 production started.

Within this paper we will present and discuss results of synthetic low-level benchmarks on JULI. This will shed light on the most important hardware features of the system, like effective latency and effective bandwidth of memory and communication hardware or performance of the FPU. These parameters are the basis for any performance estimate in high-performance computing[4].

[a] **JU**elich **LI**nux Cluster

The paper is organized as follows: In the next section a brief overview of the JULI cluster is given. In Section 3 we will introduce the two synthetic low-level benchmarks used to analyze the cluster and present the attained results. We end with a summary and draw conclusions from the presented results.

2 Hardware and Software

2.1 Compute Nodes

JULI consists of 56 IBM JS21 BladeServers. Each BladeServer is equipped with 2 Dual-Core PowerPC 970MP CPUs running at 2.5 GHz. The PowerPC CPU has a pipelined, super-scalar architecture with out-of-order operations. Each core features two 21-stage floating-point units (FPU). Every FPU allows one double-precision multiply-add operation per cycles. All this leads to a theoretical peak-performance of 10 GFlop/s per core.

Both cores have their own hierarchy of two caches. While the L1 cache is segmented into 32 kB for data and 64 kB for instructions, the 1 MB L2 cache is used for both data and instructions. The main difference besides its size is the latency of the caches access. While it takes 2 cycles to fetch data from L1 cache, the processor has to wait 14 cycles until data from the L2 cache are available.

For the JS21 blades two varieties of DDR2 memory are available as main memory; slower SDRAM modules, running at 400 MHz, and faster ones, clocked with 533 MHz. In fact, we were able to test both types of memory. This enabled us to study the memory sub-system in detail and to analyze the effects of the different memory-speeds on both synthetic low-level benchmarks and real-world applications.

2.2 Networks

The nodes are interconnected by means of three networks. Two networks are implemented by Gigabit-Ethernet technology. They are responsible for I/O and management activities. The third network is for the actual applications, i.e. MPI. Here a 10-Gigabit technology is used: QLogic's implementation of the InfiniBand[7] standard called InfiniPath[5,6]. Its most remarkable feature is an extremely low latency compared to other implementations of this standard. This is due to the fact that the protocol is not handled by a full-fledged CPU on the HCA but is mapped to the logic of a state-machine implemented within an ASIC.

Nevertheless the wire-protocol is perfectly conforming with the InfiniBand standard. As a result, standard InfiniBand switches can be employed. In the case of JULI, we choose a Voltaire ISR 9096 switch with three line-cards summing up to a total of 72 ports.

2.3 Software

The software configuration of JULI mostly follows the main-stream in cluster computing. Linux is used as the operating system. We chose the PPC-version of SLES 10. For compatibility reasons, gcc is available. Nevertheless IBM's XL compiler is the default since it provides significantly better optimization for the PowerPC-architecture. We chose the IBM XL C-compiler for the tests we present within this article. Nevertheless for comparison reasons we also ran the tests using the gcc compiler and found no significant difference.

This is no surprise since these low-level benchmarks directly test hardware capability and should be insensitive towards optimizing compilers by design.

Of course some deviations from the main-stream were necessary: As MPI-library QLogic's InfiniPath-MPI was used rather than MPICH. It supports the InfiniPath interconnect in a most efficient way. Furthermore the process management does not employ the usual ssh/rsh constructs but relies on ParaStation, jointly developed by ZAM and ParTec Cluster Competence Center, Munich.

3 Benchmarks

In order to determine some of the crucial performance parameters of the system at low level, we employed two synthetic, machine-oriented benchmark-codes. The results enable us to set expectations on the performance of real world application.

While the effects of system characteristics on the performance of actual real world applications in general are hard to predict, the knowledge of these characteristics is important in order to be able to set limits on performance expectations. As an example, if it is possible to get an estimate which memory-bandwidth is actually achievable in practice, this will set an upper bound on the performance expectations of any algorithm which stresses the memory subsystem of a computer.

The tests of real world applications were done in parallel with the analysis presented here; the results are presented in a separate publication[11].

3.1 FPU Performance

In order to determine the on-node capabilities of JULI we ran the `lmbench`-suite[8,9] containing low-level performance benchmarks on one of JULI's compute-blades.

The latencies and throughput values of the PPC 970MP processor from the data-sheet are redisplayed in columns 2 and 3 of Table 1. We confirmed the specifications by carrying out the corresponding test within the `lmbench` suite.

op.	latency	throughput	single	double
add	6	2 / cycle	2.38	2.38
mult	6	2 / cycle	2.38	2.38
div	33	2 / 28 cycles	13.1	13.1

Table 1. FPU execution times in nsec for PPC 970MP from `lmbench`

The benchmark results are presented in columns 4 and 5. Based on a cycle-time of 0.4 nsec—in correspondence with a 2.5 GHz processor clock—the results agree with the values in the data-sheet. Here one has to consider that `lmbench`'s way to explore the floating-point performance does by intention not make use of the pipelining features of modern CPUs. Thus the time measured for `add` or `mult` in fact is the actual latency of the corresponding operation. In a sense the reported numbers are a kind of worst case performance for the given architecture besides further limitations that might have been introduced by restrictions of the memory sub-system, etc.

3.2 Memory Characteristics

Another set of results from the `lmbench` suite is discussed with the aim to understand the memory sub-system of the JS21 blades. The corresponding numbers are presented in Table 2.

As mentioned above, during the JULI project the compute-nodes have been equipped with two different types of memory-modules: JULI started with 400 MHz DDR2 SDRAM modules and upgraded to faster memory running at 533 MHz. This enables us to make interesting comparisons concerning the effects of the memory-bandwidth.

MHz	procs	cores	BW [MB/sec]		latency [nsec]		
			read	write	L1	L2	mem
400	1	any	2750	1740	1.19	5.2	40.7
	2	0 1	4000	2280	1.19	5.24	60.5
		0 2	4480	2295	1.19	5.24	52.2
	4	0 1 2 3	5090	2280	1.19	5.24	99.5
533	1	any	2830	1810	1.19	5.2	39.1
	2	0 1	4020	2590	1.19	5.24	60.0
		0 2	4870	2730	1.19	5.24	45.3
	4	0 1 2 3	6141	2635	1.19	5.24	81.6

Table 2. Results for bandwidth and latency from `lmbench`'s memory test suite.

The upper part of the table shows results obtained with the older and slower memory (400 MHz), the lower part results are obtained with the faster modules (533 MHz). The four lines of each block show the outcome of the test running with 1, 2 or 4 instances simultaneously. Column 2 denotes the number of instances. The difference between the two lines referring to two instances is due to the cores that are used within a single test as indicated in column 3. For this purpose the two processes conducting the operations were bound to core 0 and 1 or 0 and 2, respectively, by using the `sched_setaffinity` functionality of the Linux kernel.

Columns 4 and 5 of Table 2 show the results for consecutive reads and writes to the main memory testing the memory-bandwidth of the system. The results for read operations are significantly larger than for write operations. In HPC practice this should be no major problem since for most algorithms the reading access to memory dominates the write operations.

It is clearly visible that for all applications with a performance-characteristic that is sensitive to memory-bandwidth we cannot expect a linear scaling within a node. Even for the faster memory the total read-bandwidth obtained for four processes is only twice as large as the one we see for one process.

The last 3 columns of Table 2 show latencies as determined by `lmbench`. As expected we see no dependence on the type of memory used when testing L1 or L2 cache as displayed in column 6 and 7, respectively. In the last column we present the latencies to access the main memory. Here effects concerning the type of memory only show up if the instances of the test make use of both sockets.

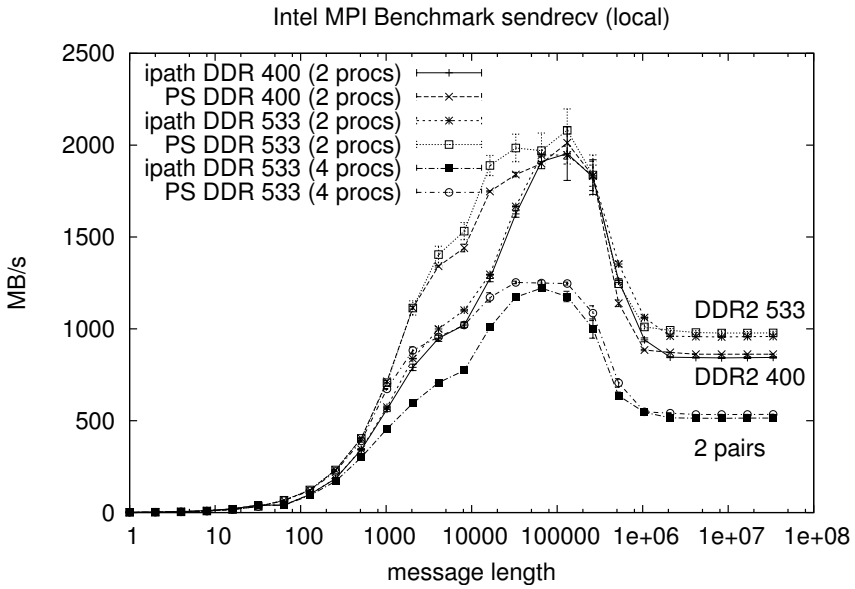

Figure 1. Intra-node MPI-communication bandwidth for ParaStation-MPI (PS) and InfiniPath-MPI (ipath). Tests utilizing two processes where executed using both types of memory – DDR2400 MHz and DDR2533 MHz. Tests with four processes only use the faster memory.

Comparing these numbers to the ones from the data-sheet given in Section 2.1, we see that in practice accessing the L1 cache takes 3 cycles instead just 2 cycles. On the other hand we see that data from L2 cache is already available after 13 cycles in most cases – in contrast to the 14 cycles guaranteed by the documentation.

3.3 The Intel MPI Benchmark

Depending on the programming model, often MPI intra-node communication is required within a parallel application. Therefore we investigate both intra-node and inter-node communication performance. We employ the Intel MPI Benchmark (IMB)[10].

3.3.1 Intra-node communication

Communication between processes allocated to the same node is expected to be highly dependent on the available memory bandwidth. In general, this type of operation will not require the communication-hardware but relies on a segment of shared memory used to copy the data from the address-space of one process to another.

Therefore IMB is another tool well suited to get a feeling of the capabilities of the memory sub-system of the JS21 blades. Within our tests of the intra-node communication we made use of two different implementations of MPI: on the one hand, we employed QLogic's InfiniPath-MPI library that also supports communication between the nodes. On the other hand, we used an implementation of MPI which is part of ParTec's ParaStation

suite. ParaStation will serve as a reference-implementation of local shared-memory communication.

Figure 1 shows results of IMB's `sendrecv`-test for various combinations of MPI-implementation, type of memory-modules and number of processes. We carried out tests between two processes (i.e. one pair passing messages) with both types of memory and both MPI implementations. Furthermore, results obtained from runs with 2 pairs of processes on the fast memory are presented.

It can be observed that the ParaStation MPI shows consistently better performances than InfiniPath-MPI. This is true for both types of memory and over the whole range of message lengths. In all cases one has to discriminate two regions of results: all message-sizes smaller than about 512 kB will be handled within the caches of the involved CPU[b]. Only messages larger than this threshold will actually be sent via the main memory. Accordingly, only results for larger messages are sensitive to the different memory speeds.

For both regions the ParaStation implementation is superior, even if the difference for large messages is just in the 5 % range. Only in the region of 1 MB messages is QLogic's implementation on par with the ParaStation MPI.

3.3.2 Inter-Node Communication

Inter-node communication makes use of the high-speed InfiniPath network. Since the ParaStation-MPI has no optimized support for this type of interconnect, only results using the InfiniPath-MPI are presented. We ran all tests for both types of memory. However, we saw almost no differences. We conclude that the memory bandwidth is not a bottleneck for the inter-node communication.

By means of IMB's `pingpong` benchmark we determined the network latency on MPI-level. We notice a half round-trip time of less than 2.75 μsec, which is good compared to other InfiniBand implementations. However, the difference to results with InfiniPath on other PCIe platforms—less than 2 μsec—is still substantial. As a comparison, we also ran the `pingpong` test on another Cluster in Jülich. This one is equipped with InfiniPath and two dual-core Opteron CPUs per node. The latencies observed here are as low as 1.7 μsec or 1.9 μsec depending on the CPU used[c].

We conclude that the complex architecture of the JS21 blades and the long path from the PPC CPUs to the InfiniPath HCAs presumably is responsible for this result. In fact, besides a comparable south-bridge in both, the JS21 and the Opteron nodes, the JS21 involves a north-bridge which introduce the additional latency. However, it is beyond the scope of this paper to analyze in detail the actual cause of this problem.

Further tests were carried out in order to analyze the bandwidth of the interconnect. Again we apply the `sendrecv` benchmark of IMB. In Figure 2 one has to distinguish three sets of tests: the two topmost graphs show bandwidth-results for communication between two processes, i.e. one pair of processes passing messages to each other. For the two graphs in the middle two pairs of processes make use of the same HCAs and physical

[b] Since the `sendrecv`-test concurrently sends and receives data to separate buffers, only half of the cache can be used for a single message. Additionally the shared-memory segment occupies further cache-lines.

[c] Within the Opteron nodes the south-bridge hosting the PCIe-subsystem is directly connected by a HyperTransport (HT) link to one of the CPU-sockets (the one with core 0). All other CPU-sockets have to go via the first socket in order to reach the south-bridge.

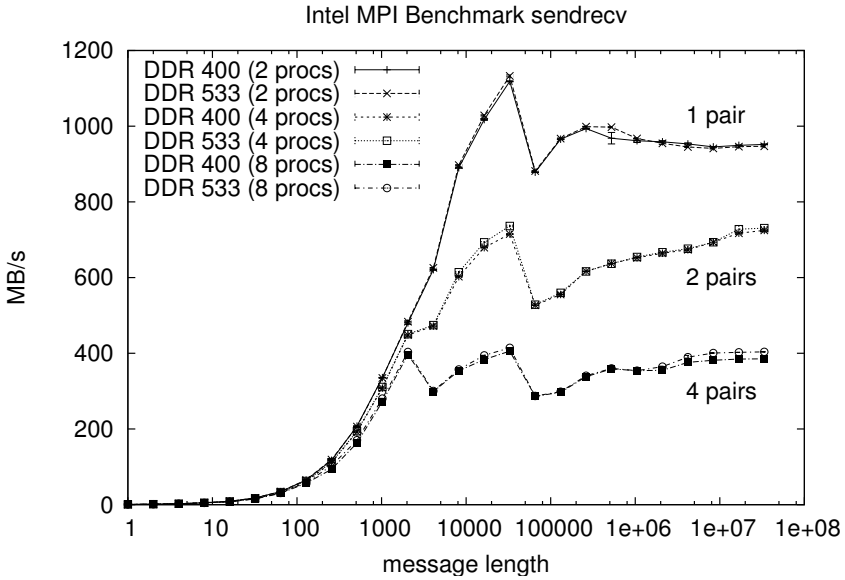

Figure 2. Inter-node MPI-communication bandwidth on JULI. Results for 1, 2 and 4 pairs of processes passing messages to each other concurrently are shown. Bandwidth shown is per pair of processes.

wire at a time. The two graphs at the bottom are for four pairs of processes occupying a single physical wire during the measurement.

Again, two regions have to be distinguished: For messages up to a length of about 2000 bytes the bandwidth-results do not depend significantly on the number of process-pairs passing data. This changes above this threshold where the observed bandwidth per pair of processes is strongly correlated to the number of pairs. In fact not the message-length is the crucial parameter, but the total amount of bandwidth occupied – which in fact also depends on the message-length. For 2k messages we see a bandwidth of about 400 MB/s per pair. Employing 4 pairs this leads to a total (bidirectional) bandwidth of 1.6 GB/s – which is close to the physical limit of 2 GB/s.

Nevertheless one puzzle remains for large messages: While we see a total bidirectional bandwidth of 1.6 GB/s using 4 pairs, this drops to 1.4 GB/s for two pairs and even below 1 GB/s if only one pair of processes generates traffic.

The reason for this behaviour might be a sub-optimal implementation of the actual transport layer within the InfiniPath-MPI. A good indication that this is the case might be found in the fact that we see a break-down of bandwidth for messages of size 64k. At this message-size other MPI-implementation like MPICH switch to a rendezvous-protocol. A more fine-tuned version of the InfiniPath-MPI library might fix this problem.

4 Summary and Conclusion

The conclusions we can draw from JULI extend the project and the type of hardware involved. The finding that dual-core or multi-core architectures will complicate life in HPC

significantly is not surprising: until now the growth-rate of processor performances was slightly larger than the increase of memory-bandwidth. With the appearance of architecture with multiple cores the amount of memory bandwidth required to balance the system for HPC increases by factors significantly larger than 1 for each generation of processors.

Furthermore, the JULI project shows that in HPC we have to minimize the distance between processor and HCA, i.e. we need to keep the complexity of the infrastructure in between as small as possible. In the case of the JULI cluster the latency penalty for inter-node communication already is significant: the time a CPU has to wait for data from an interconnect will become more and more expensive with increasing number of cores: JULI's penalty of 1.0 μsec compared to other PCIe architectures corresponds to $\mathcal{O}(2500)$ cycles of each CPU, a number that already amounts to $\mathcal{O}(40000)$ floating point operations per node.

5 Acknowledgements

We thank Ulrich Detert, who is responsible for the JULI system at FZJ for his continuous support.

We are grateful to Otto Büchner for his kind support of our test on the JUGGLE Opteron Cluster in Jülich.

Furthermore we thank the teams of our project partners, ParTec, QLogic and IBM-Böblingen, who together with the ZAM-team have developed JULI with remarkable enthusiasm and devotion.

References

1. http://www-5.ibm.com/de/ibm/produkte/entwicklung.html
2. http://www.qlogic.com
3. http://www.par-tec.com
4. An overview on available models can be found in: E. Sundarajan and A. Harwood, *Towards parallel computing on the internet: applications, architectures, models and programming Tools*. arXiv:cs.DC/0612105.
5. QLogic InfiniPath Install Guide Version 2.0 (IB0056101-00 C).
6. QLogic InfiniPath User Guide Version 2.0 (IB6054601-00 C).
7. http://www.infinibandta.org
8. L. McVoy and C. Staelin. "lmbench: Portable tools for performance analysis". In Proc. Winter 1996 USENIX, San Diego, CA, pp. 279-284.
   C. Staelin. "lmbench – an extensible micro-benchmark suite." HPL-2004-213.
9. http://www.bitmover.com/lmbench/
10. http://www.intel.com/cd/software/products/asmo-na/
    ... eng/307696.htm#mpibenchmarks
11. U. Detert, A. Thomasch, N. Eicker, J. Broughton (Eds.) JULI Project - Final Report; Technical Report IB-2007-05