Student Cluster Competition 2014

— Report —

Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

Vorgelegt von: Anna Fuchs, Jakob Lüttgau, Jennifer Nissley, Marian Tietz, Johann Weging
Betreuer: Michael Kuhn

Hamburg, den 30.09.2014

Abstract

The International Supercomputing Conference (ISC[1])is an annual global conference and exhibition for high performance computing. New and established companies in the high performance computing field are able to present their products and discuss the needs of their potential customers. Additionally, several workshops are held to communicate the knowledge gained in research groups. One major event is the anouncement of the new TOP500 list of the 500 best performing supercomputers world-wide.

Another event at the ISC is the Student Cluster Competition (SCC) where student teams from different universities from around the world come together and compete for the best performing cluster system. This report summarizes the experiences of the team from the University of Hamburg at the SCC of ISC'14 and highlights what can be done better in the future.

[1] `http://www.isc-events.com/isc14/`

Contents

1 Introduction and Motivation

In July 2014, the 3rd HPCAC-ISC Student Cluster Competition[1] during the ISC'14 Conference and Exhibition took place in Leipzig.

We were participating to – first and foremost – gain experience, both technically and personally. We wanted to prove and – of course – improve our skills and were looking forward to working together as a team. The Student Cluster Challenge was a very good opportunity to get to know the international HPC scene, which is not easily possible for students. It was also a good chance to compare our own methods with the other teams. Additionally, it was very interesting to have the chance to work with this kind of hardware and resources, because they are generally not easily accessible.

1.1 Task and Rules

The 11 teams, each consisting of 6 undergraduate students from all over the world, built a small prepared cluster in a real time competition. They tried to reach best performance results with their systems under certain conditions. There were three categories of awards to be given - three prizes for Overall Winners, the highest LINPACK performance and the Fan Favorite prize.

The power budget was limited to 3 kW on one monitored PDU. Exceeding this budget was penalized down to disqualification from the competition or parts of it. It was not allowed to restart the system or its components and also to change any of them at any time of the competition, save as otherwise permitted. The adviser was not allowed to actively support the team during the day.

Three applications were published some months in advance as well as the used benchmarks were known from the beginning of the competition. One addition application was published on the immediately preceding day.

[1] `http://www.isc-events.com/isc14/student-cluster-competition.html`

2 System

Author: alle

2.1 Hardware

The preliminary cluster configuration is given below. The hardware is provided by sysGen[1].

- 1 × Head node
  - 1 × Intel Xeon Ivy Bridge-EP E5-2609v2, 2.5 GHz, 4 cores
  - 32 GB main memory
  - 2 × 120 GB solid-state drive
  - 4 × 2 TB hard disk drive, 7.200 RPM
  - Mellanox ConnectX-3 InfiniBand, FDR (56 Gbit/s)
- 8 × Compute node
  - 2 × Intel Xeon Ivy Bridge-EP E5-2680v2, 2.8 GHz, 10 cores
  - 128 GB main memory
  - 500 GB hard disk drive, 7.200 RPM
  - Mellanox ConnectX-3 InfiniBand, FDR (56 Gbit/s)
- Mellanox InfiniBand switch, FDR (56 Gbit/s), 12 ports
- Netgear Gigabit switch, 24 ports

[1] `http://www.sysgen.de/`

2.1.1 Power Consumption

The following table shows the power consumption given by the sponsor.

Node type	Count	Current (A)	Power (W)
Head node	1	0.87 A	200 W
Compute node	8	1.30 A	300 W
InfiniBand	1	0.29 A	300 W
Gigabit Ethernet	1	0.29 A	21 W
Sum		11.85 A	2,921 W

To ensure that the values are accurate we did current measurements with two voltmeters, which revealed deviated results due to enabled turbo boost. To ensure we would not exceed the limit, turbo boost has been later disabled.

2.2 Operating System

Author: Jakob Lüttgau

This section is giving a brief overview of different considerations made before settling with Qlustar. There are many different choices for a cluster operation system available. Thriving for resource exploitation may make choosing the options with the most fine grained control over every aspect the optimal choice because any unnecessary overhead could potentially be removed. This would favor a custom tailored system.

On the other hand the cluster needs to be flexible to react to unforeseen problems than can arise during operation as well as during solving the surprise application that is part of the student cluster challenge.

The amount of different technologies involved should not be neglected as it requires considerable efforts in e.g. system administration, storage and network technologies to name a few to setup a working cluster. Beware of further raising the bar by introducing unrealistic ambitions.

Not every aspect of the system can neither needs to be fully understood. In retrospect it became obvious that there will be not a lot of time to troubleshoot problems during the competition, especially with constraints that the system or even parts of must not be restarted.

Special considerations were made towards Cent OS or Ubuntu/Debian based systems, as the first is popular in industry (e.g. Amazon Web Services) and the second comes with a large user base and is already in production in the cluster at the workgroup "Scientific Computing" at the University of Hamburg.

Qlustar is a fairly new linux distribution targeting cluster installations by shipping with software support for many common HPC related demands. Qlustar is based on Ubuntu/Debian and integrates the experience of the software engine BeoBox which is

also developed by Q-Leap. BeoBox already powers multiple cluster installations for many years.

The distribution as of Qlustar Version 8.1 comes with lightweight node operating systems as well as many management and monitoring facilities preinstalled. Utilities to add cluster hardware and manage node provisioning also allow to adjust and setup up network setup such as the InfiniBand fabric.

2.2.1 Installation and Setup

An installation image of the distribution can be downloaded free of charge from the qlustar homepage `https://qlustar.com/download`. The trail version, does not have any time constraints but the cluster management utilities can not be used without obtaining a license.

Qlustar is installed only on the head node. The compute nodes fetch their boot images via netboot using PXE (Preboot Execution Environment), which needs to be configured in the node BIOS/node firmware. Installation on the head node is straight forward thanks to a guided installation wizard. On first boot the post-install script should be executed to fully setup Slurm, Nagios and Ganglia. In environments with strict internet security policies this can cause problems because the post-install script requires internet access to download the most recent software.

The next steps require a Qlustar license. Without a registered copy of Qlustar the InfiniBand fabric, node image provision and authentication and monitoring need to be configured manually. No operating system images will be served to booting nodes on the network without beeing registerd to Qlustars cluster management tool " qluman".

Registering new nodes using the qluman, build with Python and Qt, however is very easy. A special network discovery frontend is provided which lists upcoming nodes seeking a DHCP lease. Nodes can than added by their MAC address, a fixed IP can be set as well a template which specifies which image to feed to the node as well as other configuration options.

The changes to node and cluster configuration are kept in a database but to take effect qluman needs to generate and overwrite affected system configuration files. This effects NIS, NTP, Hostfiles, InfiniBand aswell as the Slurm and Nagios/Ganglia configurations.

2.3 InfiniBand

InfiniBand (IB) is one of the most performant computer network technologies, which is widely used in industrial high performance computing. Nowadays InfiniBand is used to interconnect within and among of computer systems. It is characterized by featuring high throughput at low latencies (about 1-2 μs). There are two manufacturers of switches and host bus adapters for InfiniBand - Intel and Mellanox.

FDR InfiniBand, which is not the fastest available one, provides up to 14 Gb/s per lane and up to 56 Gb/s per port, since 4x lanes are common (up to 12x available).

The main difference between InfiniBand and the early Ethernet is the used topology. The particular nodes interconnect via network switches. The total throughput is higher due to multiple physical links connected over switches.

2.3.1 Evaluation

The reference performance values of the sponsored InfiniBand are as follows:

	Mellanox 56Gb/s FDR IB
Throughput	6.8 GB/s
Latency	0.7 μs
Message Rate	137 Million msg/sec

Table 2.1: Reference Mellanox IB FDR56 native performance

First of all the real native performance should be measured, prior to involvement of further libraries like MPI (see Section 5.3). To check the state of the IB use `ibstatus`:

```
1  $ ibstatus
2
3  Infiniband device 'mlx4_0' port 1 status:
4  default gid:     fe80:0000:0000:0000:0025:90ff:ffdf:39a9
5  base lid:        0x6
6  sm lid:          0x1
7  state:           4: ACTIVE
8  phys state:      5: LinkUp
9  rate:            56 Gb/sec (4X FDR)
10 link_layer:      InfiniBand
```

Graphics 2.1 and 2.2 show the results of perftest[2] benchmark done by Julian Kunkel.

[2] `git://git.openfabrics.org/~shamoya/perftest.git`

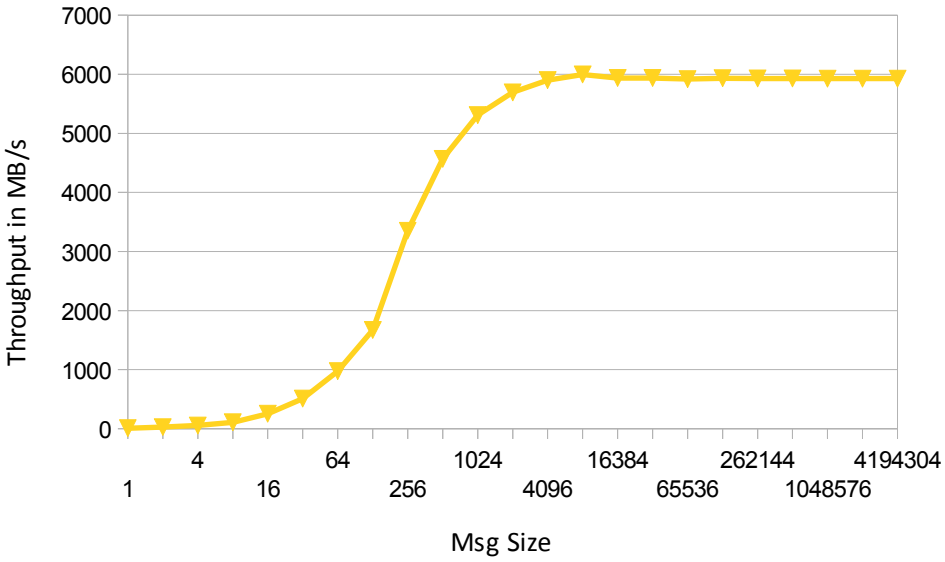

Figure 2.1: Writing performance of InifiniBand

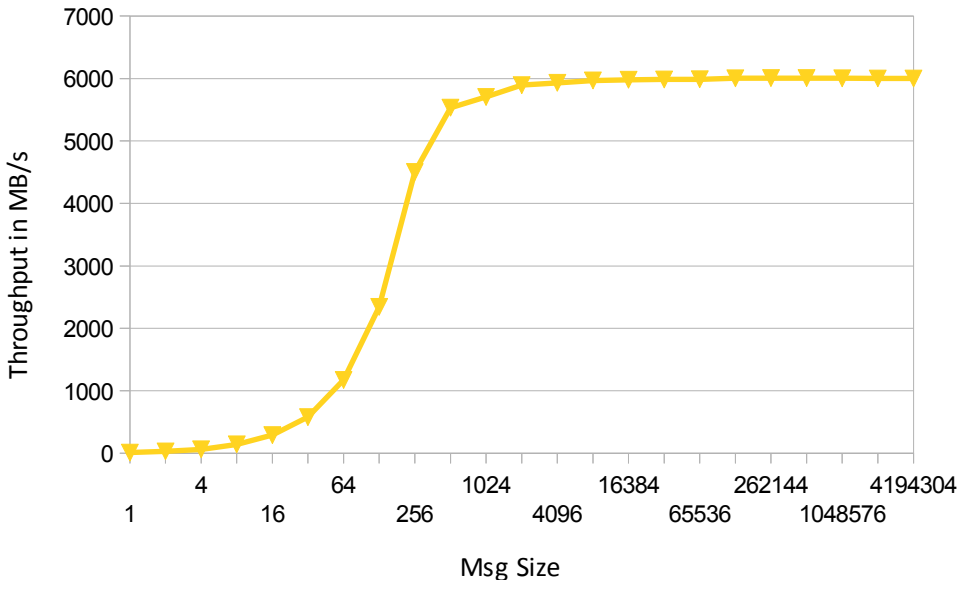

Figure 2.2: Reading performance of InifiniBand

The initial results without tuning are about 0.8 GB/s weaker then the reference performance. Investigation of the bottlenecks showed, that it is helpful to use more streams and that touching the buffers adversely affects the performance.

2.4 Software

2.4.1 Batch System

Managing lots of parallel used resources manually is hard, unreliable and insecure. The best way is to use a well implemented, comfortable and robust batch system. One of the widest used ones is SLURM (Simple Linux Utility for Resource Management)[3]. The main tasks of such a a manager are allocating resources and starting, executing and monitoring work on the distributed resources. Additionally, SLURM provides a job scheduler for managing the queue of pending work.

SLURM's core are two daemons - `slurmctld` and `slurmd`. The first one is started on the master or login node, the other on compute nodes.

Qlustar provides SLURM automatically if chosen during the installation. The daemons on compute nodes can be comfortably restarted using Qluman.

Alternatively we could define hostfiles for launching jobs with MPI and doing lots of additional manual work with environment, output and queuing of the jobs.

2.4.2 Modules

To use different library implementations and versions in a more comfortable way there exists a package named "Environment Modules"[4]. This package should not be mixed up with the loadable kernel modules for better dynamic organization of the Linux kernel parts. Environment Modules also provide dynamic modification, not of the kernel, but of the user's environment.

Take note of the fact that Environment Modules package has been renamed to "Modules" by Qlustar.

All required configuration information like paths and environment variables are kept in modulefiles. Every modulefile defines a module, which can then be loaded or unloaded dynamically and atomically. Due to the dependencies of libraries on compilers or other libraries it is advisable to first think up a well organized hierarchical structure of the modules. A special attention has to be paid to the compatibility of different modules.

The first hierarchical level are the compilers. Due to the missing binary interface compatibility of the compilers the further structure tree paths should not intersect. The same applies to all incompatible libraries.

Listing 2.1 shows an example modulefile for MVAPICH2 version 1.9 compiled with debugging options using GCC 4.9 compiler.

Listing 2.1: Modulefile example

```
1 #%Module1.0####################
2
3 set root /musorka/opt/gcc/4.9/mvapich2/1.9_debug
```

[3] https://computing.llnl.gov/linux/slurm/
[4] http://modules.sourceforge.net/

```
4
5 prepend-path PATH             $root/bin
6 prepend-path LD_LIBRARY_PATH $root/lib
7 prepend-path MANPATH          $root/share/man
8 prepend-path PKG_CONFIG_PATH $root/lib/pkgconfig
```

The first line has to begin with the magic characters, which identify this text file as a modulefile. The second line defines the root directory, where the binary for the module content is defined. The rest lines set all necessary paths.

All available modules are listed with the `module avail`. The commando to load the module looks as follows:

Listing 2.2: Load a module

```
1     module load gcc/4.9/mvapich2/1.9_debug
```

All of the already loaded modules can be looked up with the command `module list`.

Beware of loading conflicting modules like different MPI implementations or several compilers at the same time: their correct work is not guaranteed.

When launching jobs via SLURM you usually do not need to specify any modules in the batch script, since SLURM exports all the environment options including set paths and variables. That means, the job would be launched with the modules loaded on the login node (where from you usually submit the jobs). Anyway, it could be helpful to specify the required modules in the batch script to prevent any confusion with wrongly loaded or missing modules. To do so the modules have to be installed at least on that node, which will provide the batch script options and load the modules. Because this node is any from the list of involved nodes, you usually install the necessary software on all of them.

Sharing the system with lots of people makes it complicated to rebuild libraries, that the others need. To test different configurations of libraries you may want to build the library locally. To avoid conflicts with other modules the right paths have to be adapted or the local libraries can also be specified as modules. For this purpose the `MODULEPATH` has to be extended by the path to the local build.

3 Compiler

Author: Johann Weging

This Chapter is about the compilers used to compile the programs and benchmarks to run on the cluster. The two compiler looked at are the Gnu Compiler Collection (GCC) (gcc14b) and the Intel C++ Compiler (ICC) (icc14).

3.1 GCC

3.1.1 Building and Installation

Since Qlustar comes with GCC 4.6 it is necessary to install a newer release manually. The latest release of GCC (4.9.0 at this time) is chosen to be installed. GCC can be downloaded from one of these mirrors *https://gcc.gnu.org/mirrors.html* (gcc14a).

The installation process is shown in listing 3.1, *aptitude* and *make install* need to run as *root*. First some build dependencies have to be installed, *aptitude* will take care of that. After GCC is downloaded it has to be extracted and configured for building. Some of the interesting configuration flags are *–with-arch, –with-tune* to determine which CPU architecture should be used as default when generating code. When *configure* finishes a simple *make, make install* is enough to compile and install it.

Listing 3.1: Building and installing GCC 4.9.0.

```
$ aptitude install libmpfr-dev libgmp3-dev libmpc-dev \
  flex bison zlib1g-dev

$ tar -xf gcc-4.9.0.tar.gz
$ cd gcc-4.9.0

$ ./configure --prefix=/musorka/opt/gcc/4.9/4.9.0 \
  --enable-languages=c,c++,objc,obj-c++,fortran,go,lto \
  --enable-bootstrap --enable-shared \
  --enable-threads=posix --enable-checking=release \
  --with-system-zlib --enable-__cxa_atexit \
  --disable-libunwind-exceptions \
  --enable-gnu-unique-object --enable-linker-build-id \
  --with-linker-hash-style=gnu --enable-plugin \
  --enable-lto --disable-multilib --disable-werror \
  --with-arch=ivybridge --with-tune=ivybridge \
  --build=x86_64-linux-gnu

$ make -j8
$ make install
```

To be able to load the compiler a module file is needed. Listing 3.2 shows the module file used in this setup. It sets the necessary paths to use GCC.

Listing 3.2: GCC 4.9.0 module file.

```
# /etc/qlustar/common/modules/modulefiles/gcc/4.9/4.9.0
set root /musorka/opt/gcc/4.9/4.9.0

prepend-path PATH            $root/bin
prepend-path LD_LIBRARY_PATH $root/lib
prepend-path LD_LIBRARY_PATH $root/lib64
prepend-path MANPATH         $root/share/man
prepend-path PKG_CONFIG_PATH $root/lib/pkgconfig
```

If the setup was successful can be checked by loading the module and do a version check shown in Listing 3.3.

Listing 3.3: Checking the GCC setup.

```
$ module load gcc/4.9/4.9.0
$ gcc --version
  gcc (GCC) 4.9.0
  Copyright (C) 2014 Free Software Foundation, Inc.
  This is free software;
  see the source for copying conditions.
  There is NO warranty; not even for MERCHANTABILITY or
  FITNESS FOR A PARTICULAR PURPOSE.

// gcc -v will give you a more complete overview
```

3.2 ICC

3.2.1 Building and Installation

To download and install the ICC it's necessary to create a account at the Intel Developer Zone (int14a). After the account is activated, it's needed to sign in at the Intel Registrationcenter (int14b). After the sing in it is possible to register the compiler by entering the seiral number on the right. Refresh the page and the compiler can be downloaded.

After the download is finished the ICC needs to be extracted. For the installer to function properly the *cpio* command is required.

```
$ aptitude install cpio
$ tar -xf l_ics_2013.1.046_intel64.tgz
$ cd l_ics_2013.1.046_intel64
$ ./install.sh
```

The ICC installation dialog will start and ask which user should be used to install the ICC. The installation will be performed as *root* to install the ICC system wide. The installer will check if everything which is needed is installed on the system. Probably some of the optional dependencies are missing but this can be skipped. After accepting the license agreement the ICC needs to be activated. The simplest way is to use the same serial number again, and enter it into the installer by choosing the default action.

Now the installation can be customized. The only thing that has to be changed is the install path. The comopnents to be installed ca be changed but don't have to. The install directory is */musorka/opt/icc/2013.1/2013.1.046*.

```
1. Start installation Now [default]

2. Change install directory
   ↪ [/musorka/opt/icc/2013.1/2013.1.046]
```

```
3. Change components to install [Custom]
4. Show pre-install summary
```

3.2.2 Tips for the Future

It's quite complex to get the ICC up and running and to figure out which library to use and when. Instead of using a module file it would be easier to just use the *iccvar.sh* script inside the *bin* folder. Another problem is to figure out which threading and math library to use. Using the *AVX*-math library failed and it was not clear what to do to get it to work.

3.3 Conclusion

The GCC is easy to set up and use and works very well for the most applications. All programs which are run on the cluster where build with GCC. The ICC is complex and hard to setup. May it will generate faster code? But one has to put a lot of effort into it to get it up and running.

4 Libraries

Author: Marian Tietz

4.1 Overview

This chapter reasons about the choice of mathematics libraries we made for the applications and benchmarks of the cluster challenge. Since optimal hardware utilization is a major key to success, choosing the right libraries for the tasks is important.

Over the course of this chapter a selection of libraries and test results, on a test cluster as well as the final competition cluster, are presented.

4.2 Available libraries

Three library categories, namely LAPACK [1], BLAS [2] and FFT [3], were necessary for the benchmarks and applications. Several implementations exist for all categories, which can vary widely in performance and ease of use. Initial research yields the following possible implementations of the different APIs for our platform:

LAPACK	BLAS	FFT
Intel MKL	ATLAS	FFTW2
LAPACK	GotoBLAS2	FFTW3
ScaLAPACK	OpenBLAS	Intel MKL
PLASMA	Intel MKL	

These implementations were filtered for up-to-dateness and their likelihood of being efficient on the cluster. The reasoning behind the filtering is explained in the following sub-sections. After filtering out the promising candidates, the table presented above changed to the following:

LAPACK	BLAS	FFT
Intel MKL (11.1)		FFTW2 (2.1.5)
LAPACK		FFTW3 (3.3.4)
	OpenBLAS (9c51cdf)	Intel MKL (11.1)
	Intel MKL (11.1)	

[1] Linear Algebra PACKage
[2] Basic Linear Algebra Subprograms
[3] Fast Fourier transform

4.2.1 LAPACK implementations

netlib LAPACK [4] is fairly up-to-date but criticized for its lack of scalability on distributed-memory systems. For this reason, ScaLAPACK[5] was developed. ScaLAPACK's latest release is from May 2012 and therefore cannot support current CPU architectures, which was the reason to rule ScaLAPACK out.

PLASMA's documentation states its purpose is to eliminate the shortcomings of LAPACK and ScaLAPACK on multi-core systems and their inability to efficiently utilize accelerators. It does not, however, serve as an replacement for ScaLAPACK since it developer for shared-memory systems. This and the fact that PLASMA still does not fully implement all functionality of LAPACK rules PLASMA out as well. The only remaining implementation that meets our up-to-dateness and performance criteria is therefore the netlib LAPACK implementation and the implementation optimized by Intel in the MKL.

4.2.2 BLAS implementations

BLAS is used by LAPACK to do the heavy-lifting computations. A general rule is: if BLAS is fast, LAPACK will be fast. For this reason, a highly optimized BLAS implementation is important to achieve good results.

GotoBLAS2 is officially discontinued [6] and only supports Intel Nehalem, which renders it useless for our purposes. The source code of GotoBLAS2 was used to create OpenBLAS which is still actively maintained and supports Intel SandyBridge as well as the Haswell platform. It also incorporates the netlib LAPACK implementation but can be used with PLASMA if so desired.

ATLAS also implements some LAPACK functions but is mainly a BLAS implementation. The benchmark (Don13a) suggests that ATLAS is outperformed by OpenBLAS and Intel's MKL and is therefore not used in further tests.

According to benchmarks made by a JuliaLang contributers and others, OpenBLAS compared well to Intel's MKL, even outperforming MKL in several tests. (Joh13) (Don13b) While Intel's MKL still performs well, OpenBLAS seems to be a very good alternative. In conclusion, OpenBLAS and Intel MKL were selected as candidates for the BLAS implementation.

4.3 Initial testing

Since the competition cluster hardware was not available to us in the beginning, we used a different cluster to test the libraries using HPL as the benchmarking tool.

The cluster we used is a Intel Westmere cluster with 10 nodes, each having 12 cores and 12 GB RAM per node. The nodes are connected using Ethernet, so latency intensive tests were expected to be slow.

[4] `http://www.netlib.org/lapack/`
[5] `http://www.netlib.org/scalapack/`
[6] `https://www.tacc.utexas.edu/tacc-software/gotoblas2`

As this cluster is a few years behind current CPU architectures, the results can only be seen as a hint, not as a general rule. It was clear that further testing on the final hardware was necessary. However, the results of the Westmere benchmarks still carry some significance and hint in what direction to go in terms of software selection.

The software was used for testing besides the libraries was HPL in version 2.1 and OpenMPI 1.6.5. OpenBLAS was compiled using GCC 4.7.2 and Intel ICC 2013.1.046. HPL was configured with a process grid of $P \times Q = 10 \times 12$, 192 blocks and a problem size of 100,000 (<80% of memory usage).

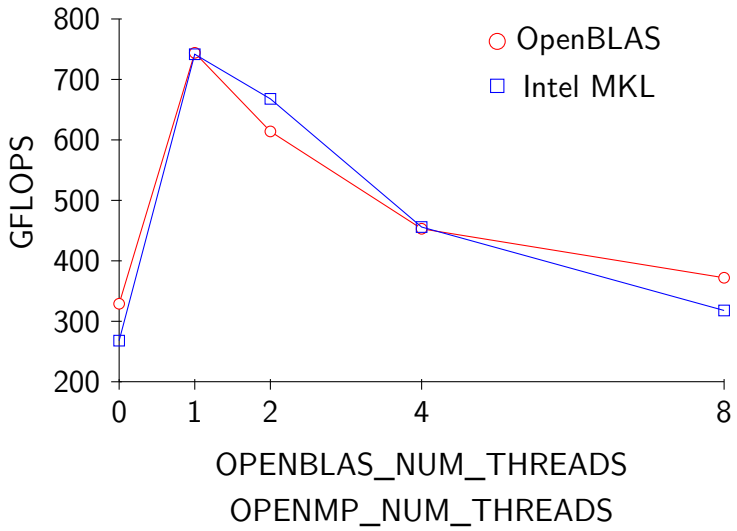

Figure 4.1: HPL performance using OpenBLAS with different threading configurations on the Westmere cluster in comparison to Intel's MKL using the same setup.

Hyper-Threading did not prove to be effective when testing with HPL. It is assumed that the CPU utilization is already high enough for HT improve anything.

FFTW was not benchmarked on the initial test cluster.

4.4 Library specific optimizations

For OpenBLAS no specific compile time optimizations were applied since these are applied automatically depending on the `PLATFORM` parameter supplied when building the library. It also became clear that OpenBLAS performs better when using only one thread, which is achieved by setting the environment variable `OPENBLAS_NUM_THREADS=1`.

For Intel's MKL, similar performance characteristics were observed when setting `OPENMP_NUM_THREADS=1` as shown in Figure 4.1. No compile time optimizations were done when linking with MKL except when otherwise stated by Intel MKL's Link Advisor[7].

For FFTW2 threading and MPI support was enabled at compile time but no further optimizations were applied. FFTW3 on the other hand was configured to use SSE2 and AVX instructions. The complete configuration line was:

[7] `https://software.intel.com/en-us/articles/intel-mkl-link-line-advisor`

```
--enable-sse2 --enable-avx --enable-threads --enable-mpi \
              --with-combined-threads
```

4.5 Final hardware testing

This section presents the results of testing the selected software stack on the cluster competition hardware. The software setup changed slightly with the final setup: OpenMPI was mostly replaced with MVAPICH and the GCC version was 4.9 instead of 4.7.1. The most significant change performance-wise seemed to be the MPI layer.

HPCC was used to test the performance. HPCC uses HPL but, amongst other things, benchmarks the FFT as well. HPCC adjusts itself according to the HPL configuration, which was similar to the one used in initial testing: N=100,000, $P \times Q = 8 \times 20$, NB=192. The process grid was arranged so that P matches the number of nodes and Q matches the number of cores. While the problem size is sub-optimal, it allows for quick testing while being moderately accurate in terms of the machine's actual performance. In the following the HPL and FFT performance is presented, starting with the HPL performance of OpenBLAS with GCC and Intel's MKL with ICC, as shown in Figure 4.2.

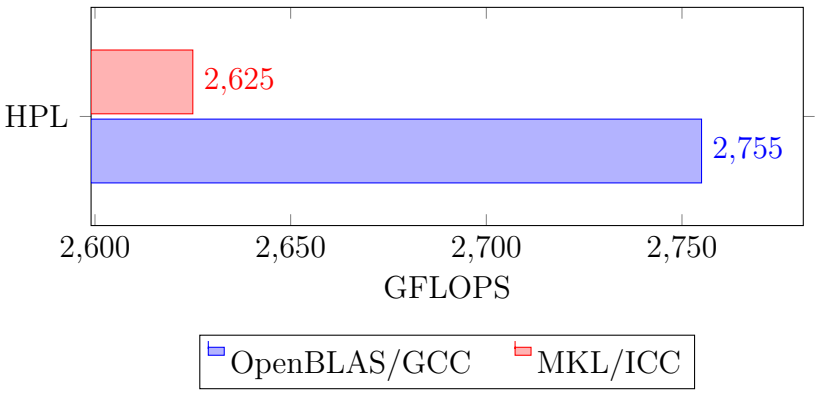

Figure 4.2: Performance comparison between HPL compiled with GCC 4.9, OpenBLAS and compiled with ICC, MKL.

The comparison shown in Figure 4.2 shows that OpenBLAS with GCC 4.9 performs a little bit better than Intel's MKL. It has to benoted that, by mistake, the MKL benchmarks were not run with `OMP_NUM_THREADS=1`, which according to section 4.4, would have been better. It is also noteworthy that MVAPICH2 was invoked with the options `-map-by socket -bind-to hwthread` but no such optimizations were applied to IPMI.

This section presents the results of the HPCC run that was used as the final result for HPCC at the competition. No changes were made to the configuration presented before except for increasing the problem size to N=296,400. The HPL benchmark achieved 3.106 TFLOPS, the FFT results are presented in Figure 4.4.

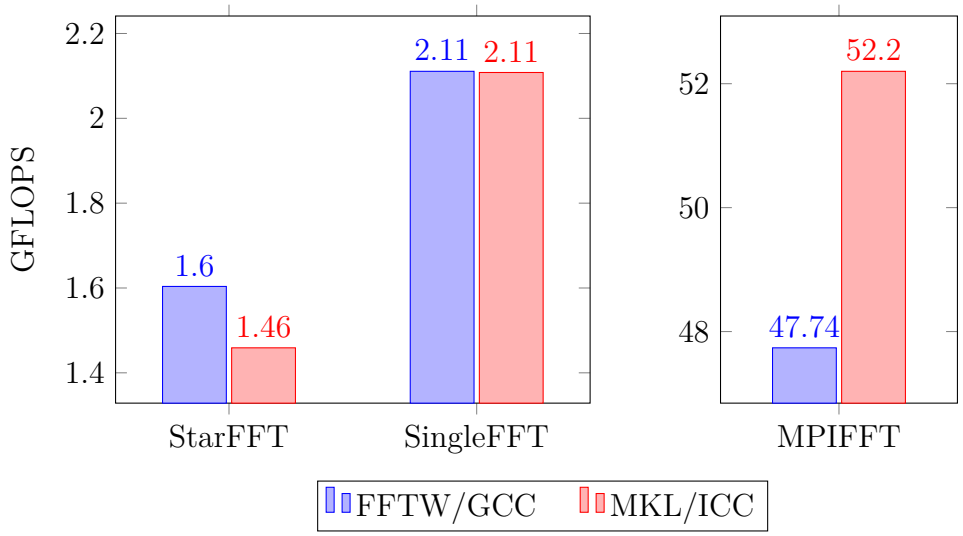

Figure 4.3: Performance comparison of FFT performance using HPCC results. FFTW 3.3.4/GCC 4.9 and using MVAPICH2 MPI implementation is compared to ICC / Intel's MKL 11.1 and Intel MPI. The problem sizes set by HPCC were $\text{FFT}_N = 8388608$ and $\text{MPIFFT}_N = 536870912$.

4.6 Conclusion and Outlook

Benchmarking found that OpenBLAS and Intel's MKL are very well on-par performance wise. OpenBLAS was very easy to integrate and use but similar things could be said about Intel's MKL when using tools such as the Intel MKL link advisor. It was found that limiting the number of threads to use in the respective BLAS implementations to 1 was most efficient when spawning as much MPI processes as there are cores. Since, by mistake, some of the measurements with Intel's MKL did not have this restriction enabled it would be interesting to see these benchmarks redone with the thread limitation active.

In hindsight, the exclusion of ScaLAPACK in subsection 4.2.1 seemed premature. Even though it is reasonable to say the ScaLAPACK release from 2012 cannot accommodate optimizations for CPU architectures from 2013, it is not known whether these optimizations are that crucial. ScaLAPACK was specially designed for distributed-memory systems, which means that it may perform better than LAPACK, even though it is not as optimized. Also, Intel MKL provides its own version of ScaLAPACK, possibly including architectural optimizations. For the future, it would make sense to include ScaLAPACK in the benchmarks.

When using accelerators, PLASMA in corporation with OpenBLAS seems to be a good choice since PLASMA explicitly attempts to improve accelerator utilization.

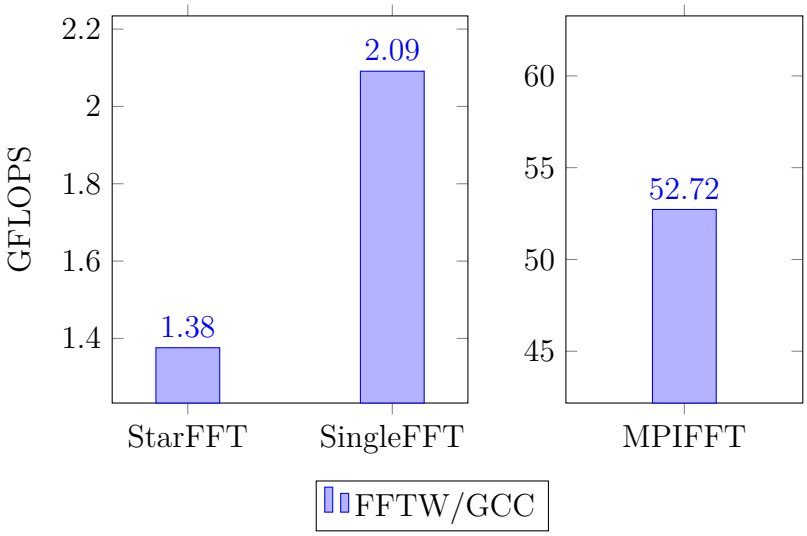

Figure 4.4: FFTW performance on the final cluster with GCC 4.9, FFTW 3.3.4 and MVAPICH2 acquired by running HPCC. The problem sizes set by HPCC were $\text{FFT}_N = 134217728$ and $\text{MPIFFT}_N = 8589934592$.

5 MPI

Author: Anna Fuchs

This Chapter gives an overview over different implementations of MPI, their differences, advantages and disadvantages.

5.1 Overview

MPI - Message Passing Interface - is not a library as it is, but a standardized specification for users and developers of message passing. MPI is the de facto standard in HPC[1].

There is a number of most used implementations of MPI which differ in performance, cost, network support, documentation, some interface details, etc. Here are the most used and best supported implementations:

- OpenMPI[2]
- MPICH2[3]
  - Active developing branch of MPICH
- MVAPICH-2[4]
  - MPICH derivative by Ohio State University
- Intel® MPI Library[5]
  - MPICH derivative
- IBM MPI[6]
  - MPICH derivative for Blue Gene Systems

[1] High Performance Computing
[2] `http://www.open-mpi.org/software/ompi/v1.8/`
[3] `http://www.mpich.org/`
[4] `http://mvapich.cse.ohio-state.edu/news/`
[5] `https://software.intel.com/en-us/intel-mpi-library`
[6] `http://www-03.ibm.com/systems/platformcomputing/products/mpi/`

Table 5.1 compares some of the most useful characteristics of these implementations with regard to decision-making whether to use them for a project. The first criterion is the cost. Nearly all MPICH derivatives except MVAPICH2 are commercial. For some of them, like the derivative by Intel, it is possible to get an academic license for tests and evaluation. Next important aspect is the ability of all implementations to support applications running over InfiniBand. Once available only on big and expensive systems today InfiniBand enjoys increasing demand, which makes it an ubiquitous interconnect of tomorrow (see more in Section 2.3). The dates of last releases often show how actively projects are developed. All of the listed implementations have been updated in 2014. When making a decision whether to use one of the implementations, one more important fact could be the backwards compatibility. Not only is the compatibility between different implementations not a common practice, but also even consecutive implementations are not always compatible (they are often considered only with regard to forwards compatibility). The Table shows the earliest version of the same implementation with which the latest release is backwards compatible (see more in Section 5.2). The last chosen criterion is the implementation of the MPI-3 standard, which was published on 21.09.2012.

This list of criteria is not full and complete. It neither claims to show the only way to rate the above-mentioned criteria. A lot of details have been left out, such as supported operation systems, compilers or language versions. For this project the first two columns were the key ones. The full support of MPI-3 standard was of no relevance, since, most likely no application and definitely none of all published benchmarks would use its functionality.

The final selection of the implementations to be tested in future consisted of three candidates - OpenMPI, MVAPICH2 and Intel MPI with the provided license. Since the latest releases of MVAPICH2 are based on the latest ones of MPICH2 and include more optimizations, MPICH2 has not been looked into.(MV2c)

Implementation	Open source	InfiniBand support	Last stable release	MPI-3 support
OpenMPI	yes	yes	1.8.1 on 23.04.2014	yes*
MPICH2	yes	yes	3.1.2 on 21.06.2014	yes*
MVAPICH2	yes	yes	2.0 on 20.06.2014	yes*
Intel MPI	no	yes	5.0 on 17.06.2014	yes*
IBM MPI	no	yes	9.1.2 on 20.02.2014	yes*

Table 5.1: Most important characteristics of various MPI implementations.

* particulary implemented

5.2 Compatibility

The interfaces of all functions are well described and standardized, but not the calling environment and some functions like the `mpirun` function.

The `mpirun` command is used as a job launcher, and since it is not part of the MPI standard, it differs in interface and functionality.

Due to that fact, there is no guarantee that an application with components compiled by using different MPI implementations will work. During the period of preparation for the SCC it was detected that OpenMPI 1.8 and MVAPICH2-1.9 definitely do not work together. When using SLURM for scheduling the resources, the typical error in this case looks as follows:

Listing 5.1: Possible error message when launching a job using MVAPICH-1.9 and OpenMPI 1.8.1 via SLURM

```
1  srun: cluster configuration lacks support for cpu
     ↪ binding
```

Nevertheless there have been an announcement at SC13[7] about the ABI[8] Compatibility Initiative, the goal of which is to make different implementations work together: „ABI, or application binary interface, is the low-level interface between two program modules. An ABI determines such details as how functions are called and the size, layout and alignment of datatypes. With ABI compatibility, programs conform to the same set of runtime conventions.“(MPIa) The collaborators at that time were MPICH v3.1, IBM MPI v2.1, Intel® MPI Library v5.0 and Cray MPT v7.0.0, which are all derivations of MPICH. For the present moment it looks like ABI compatibility standards are not going to be developed in the near future.

5.3 MPI over InfiniBand

MPI needs to be adapted to InfiniBand and its support becomes common practice. Since the InfiniBand architecture does not define an API, there are different fabrics implementations made by organizations like OpenFabrics Alliance[9].

„In general, MPI performance is dominated by interconnect fabric latency and to a less extent on bandwidth. A general rule of thumb is to use the default compiler optimizations and avoid overly aggressive optimizations.“(Int)

Using MVAPICH2 with Mellanix InfiniBand, RoCE, or iWARP network adapter there is no need to change the configuration. The MVAPICH team strongly recommends to use of the OFA-IB-CH3 interfaces for the Mellanox InfiniBand adapter, which is default to Linux.(MV2b). For the QLogic InfiniBand adapter following should be done:

[7] `http://sc13.supercomputing.org/`
[8] Application Binary Interface
[9] `link`

```
$ ./configure --with-device=ch3:psm
```

5.4 Build

This section gives a short overview over the general approach of building MPI, which, however, is not different from compiling any other library.

Before building MPI some additional libraries should be installed.

```
aptitude install libibmad-dev
aptitude install libibumad-dev
aptitude install libpciaccess-dev
aptitude install libhwloc-dev
```

`libibmad-dev` is a developer package for Infiniband Management Datagram (MAD) library. It provides low layer InfiniBand functions to be used by the InfiniBand diagnostic and management programs(libb).

`libibumad-dev` is a developer package for InfiniBand Userspace Management Datagram (uMAD) library. It provides uMAD functions which sit on top of the uMAD modules in the kernel. They are used by InfiniBand diagnostic and management tools(libc).

`libpciaccess-dev` - package library for PCI support, required by hwloc. Instead of `libpciaccess` the `libpci` packages can also be used, which is a question of using GPL[10] or other licenses. In case neither of them can be found, PCI support will be disabled by OpenMPI.(liba). PCI, is a local computer bus for attaching hardware devices(pci)

`libhwloc-dev` - hwloc for MVAPICH2. OpenMPI already includes hwloc. See more about hwlock in Section 5.5.2.

The configuration then should be performed with enabling PCI support. For further tuning options see Section 5.5.1.

```
wget http://mvapich.cse.ohio-state.edu/download/
                          mvapich2/mv2rc2/mvapich2-2.0rc2.tgz
tar -xfz mvapich2-2.0rc2.tgz
cd mvapich2-2.0rc2/

./configure --enable-libpci
   ↪ --prefix=/musorka/opt/gcc/4.9/mvapich2/2.0

make -j8
make install
```

```
wget http://mvapich.cse.ohio-state.edu/download/
                         mvapich2/mv2rc2/mvapich2-2.0rc2.tgz
```

[10]GNU General Public License

```
3 tar -xfz mvapich2-2.0rc2.tgz
4 cd mvapich2-2.0rc2/
5
6 ./configure --enable-libpci
    ↪ --prefix=/musorka/opt/gcc/4.9/mvapich2/2.0
7
8 make -j8
9 make install
```

5.5 Tuning Options

There are several available options for tuning of the implementations. Lots of them can improve the performance, some of them, like debugging output, help to improve understanding and work with MPI. Sometimes such options exclude each other, since more detailed debugging comes at the expense of the performance.

In general there are two types of tuning options - runtime and configured. Configured options are set before building the library and influence the environment used. Runtime options can be passed as a parameter while running the application - normally between the `mpirun` command and the application call.

Those options differ for different implementations. The following section describes some of the most helpful, partly tested and evaluated options in detail.

5.5.1 Configured options

OpenMPI provides default building configuration. The default options are: (ope)

- all optimizations enabled
- shared libraries
- building components as standalone dynamic shared object (DSO) files
- trying to find support for all hardware and environments by looking for support libraries and header files in standard locations, proceeding to the next stage if the support cannot be found

The default options for MVAPICH2 are: (MV2a)

- build both, shared and static libraries
- all debugging information disabled
- registration cache enabled

- chose hydra as process manager

Further options are: (MV2b)

- `--disable-shared` - creates only static libraries
- `--enable-g=all --enable-error-messages=all` - controls the amount of debugging information.
- `--enable-fast` - configure for fastest performance at the cost of error reporting and other development aids. This option disables all debugging information.
- `--enable-shared` - build with shared libraries, try to automatically detect the type of shared library support required.

When using MPI as a shared library, the application code related to the library makes a reference to the code used. When using static libraries, the required code is directly linked to the application code at compile time. Finally, it is something like copying and integrating a part of the library code. In general, shared libraries have some additional runtime load costs, while static libraries have larger binaries. The performance rates have to be investigated.

5.5.2 Runtime options

Once built the library, there is still some leeway for tuning the performance.

Process Affinity

Modern complex and hierarchical topologies have a major impact on the application performance. So optimizations of mapping and binding the processes to the hardware are necessary. When starting a parallel application MPI creates a map, pairing processes to processors and memory. To bind the processes means to define the hardware level where the processes or threads can run, owing to the ability of migration the processes by the operation system. There can be more or less useful strategies, but commonly no general solution exists.

To improve the understanding of following details, Figure 5.1 shows an exemplary hardware structure of a server node. It consists of boards, NUMA nodes and a sockets in variable number, the hierarchy of which differs depending on the architecture. In the example the three cache levels and the core are on the socket. The core consists of processing units, which enable hyper or hardware threads.

Not every implementation provides any default values of process affinity as there are no specifications in the MPI standard about the binding or mapping polices. However there are lots of options, which can and should be set manually.

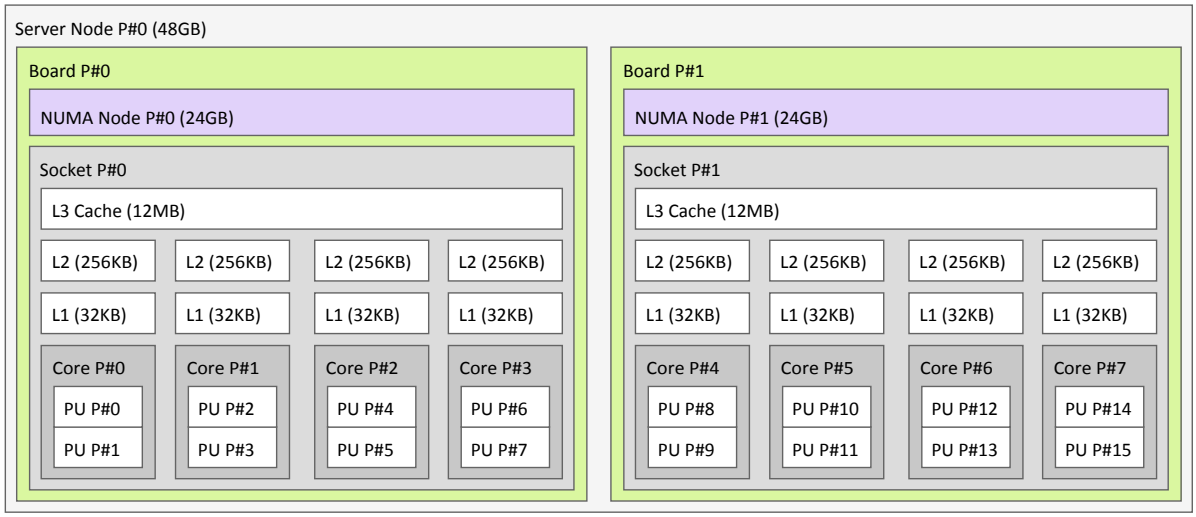

Figure 5.1: An example of hierarchical architecture of components (arc)

MVAPICH2 provides a resource manager and launcher called Hydra. It natively interacts with several resource managers and launchers. Moreover, it is configured for using *hwloc* (MV2d). The Portable Hardware Locality (hwloc) software package provides an abstraction to obtain the hierarchical map of key computing components (hwl). It is released and developed in collaboration with OpenMPI, but lots of other projects support it(hwl14).

Table 5.5.2 lists all binding and mapping options supported by the `mpirun` command using hwloc. The bold highlighted objects are defaults for the corresponding option.

Option	Object	OpenMPI	MVAPICH2
--map-by	slot, hwthread, core, L1cache, L2cache, L3cache, socket, numa, board, node, sequential, distance, ppr	**socket***	**none**
--bind-to	slot, hwthread, core, socket, numa, board, node	**core**	**same as mapping or none**

Table 5.2: Mapping and binding options

* More specific information about the default values beginning form the 1.7 version of OpenMPI: (Squ13a)

- In case the number of processes <= 2, processes mapped by core
- In case the number of processes > 2, processes mapped by socket

The following items explain every object provided by the options (Squ12)

- `slot` - indicates how many processes can potentially be executed on a node. For the best performance, the number of slots may be chosen to be the number of cores on the node or the number of processor sockets. When using SLURM OpenMPI would anyway set the slot number provided by the resource manager. The default is 1.
- `hwthread` - possible hardware thread, which runs on a single logical core - is synonymous to hyper thread.
- `core` - equivalent of a physical core. One physical core can support multiple logical cores or hardware threads (acc14). When passing this options the binding would look like one process per core on the first node, then one process per core on the next node.
- `L1cache, L2cache, L3cache` - levels of cache. On Intel architectures, L3 is larger and commonly shared among all of the cores of one socket, while L2 and L1 are core exclusive. This differs according to the architecture.
- `socket` - mapping by socket can be useful in cases where the main memory is limited. Having less processes than cores, one or few processes would use the whole socket memory.
- `numa` - the true position of allocated memory can have a substantially impact on performance. This option ensures explicit binding and mapping by a NUMA domain, especially for such architectures in which NUMA level does not correspond with the socket or the third cache levels (Rei09)(Cas14).
- `board` - processor board, an explicit option for nodes with several motherboards (DMRF11).
- `node` - server node, opposite to the core binding, node binding means one process on the first core of each node, then one process on the second core of each node, etc.
- `sequential` - reads the provided hostfile line by line, assigning processes to nodes in whatever order the hostfile specifies (mpib).
- `distance` - specifies an additional option of mapping the processes, which allows to indicate in which distance the options like core or socket bindings have to be applied.
- `ppr` - „processes per“, specifies the exact number of elements per object.

The options can be modified by adding a colon symbol and any number of processing elements to each processor. Moreover options `SPAN` (for load balancing), `NOOVERSUBSCRIBE` (forbidding more processes than) and `OVERSUBSCRIBE` are possible. Example: `--map-by ppr:4:node` (mpib).

Note, that the support for process binding depends on the underlying operating system. It is possible, that some process binding options are not available on every system. After mapping and bounding all of the processes, the library assigns ranks in `MPI_COMM_WORLD` (oCSU14) (MV213).

Several bindings or mappings can be passed in a row, which would mean that what comes first, must be done first. For example, „bind to node, bind to socket“ would mean that the first bind must be done to the node and the second bind must be done to the socket, and so on. Especially for hybrid programming it makes most sense to define both bindings and mappings. For example, mapping by socket and binding to socket enable migrations of threads among the socket, which could perform better than the default binding to a core. At the same time, biding in general can degrade performance when it inhibits balancing of the load by the operation system(ope14).

Absence of binding and mapping will highly likely cause a negative impact on the performance. If no restrictions are set, it means that the kernel is either allowed or, at least, not forbidden to migrate the MPI processes to a different core. When run on modern Intel architectures this would cause level 1 and 2 cache misses due to their affiliation to the processor core. Sometimes the operation system would oversubscribe some resources, like sockets, leaving the other available ones idle (Squ13b).

Measurements of latencies between ranks of an MPI application can help to understand the structure. Since the latencies between nodes are commonly higher than within a node it makes most sense to investigate the communication speed between sockets, cores, and CPUs regarding the cache levels. The processes should then be placed according to their affinity - keeping close processes which share, synchronize or communicate a lot.

Nevertheless, the decision making can be hard due to competing properties. Compared to two processes on different sockets, binding two processes to two cores on one socket will enable cache sharing, but reduce the memory bandwidth. In general, it depends on the machine structure and the application needs, so there is no single recipe for combination of options to provide the best performance.

A more reliable method instead of deductive measurements would be analyzing the algorithms and trying to find out if and how far they are already adapted to locality of the hardware. However, this way is sometimes far too complicated for the common use case of application's performance tuning.

The provided binding can be shown with the option `--report-bindings`, which makes each rank print its binding to the standard error output. These options are known by OpenMPI only.

MVAPICH2 also provides its own binding polices - bunch and scatter, while bunch is default. The police can be set by option `MV2_CPU_BINDING_POLICY=bunch`. Bunch groups the processes on cores of one socket, while scatter almost uniformly distributes them over cores on all sockets. Correct option could improve latencies of the point-to-point communication (oCSU14).

Launching

There are two possibilities to launch a parallel job - to use resource manager, the other is to make it manually. The benefits of resource managers like SLURM are convenient passing of parameters, implicit monitoring and lots of features, that make the work easier. The cost is often the longer launching time due to the load coming from additional services. On the other hand, manual launching requires a greater effort and more attention due to a higher risk of semantical and syntactical errors.

The difference in runtime is first noticeable using around 8 nodes. In conditions of the competition, when even queuing of the jobs was optional, there is no need to tune of job launching time.

Both implementations provide many more tuning options, which, nevertheless, affect the performance on bigger systems. For example, there exists an MVAPICH2 option for scanning the system for heterogeneity before launching - `MV2_HOMOGENEOUS_CLUSTER`. If the fact of homogeneity of the system has already been established, this option can be set to 1 to skip the verification. Either way, the unnecessary verification does not affect clusters with number of nodes in a single-digit range. Other optionS refer to buffer sizes, USE of shared memory for different MPI functions and many other things (MV2b).

5.6 Evaluation

The implementations were evaluated using the High Performance LINPACK and IMB (Intel® MPI Benchmarks). The versions OpenMPI 1.8.1, MVAPOCH2 1.9 and MVAPICH2 2.0rc were tested and compared.

5.6.1 HPL

For the first results HPL 2.1 was used with the an input file consisting the following lines:

```
1 100128          Ns
2 224             NBs
3 8               Ps
4 20              Qs
```

All the other options has been not modified. One run took about four minutes and was sufficient for a first approximation. The measurements were repeated at least three times. All of the libraries were built with all available performance options and disabled debugging support. All the runs were performed on 8 nodes with 160 processes.

Before the decision of disabling the turbo boost, some measurements were performed in order to find out the potential performance loss. Figure 5.2 shows the results. Nearly 180 GFLOPS were lost.

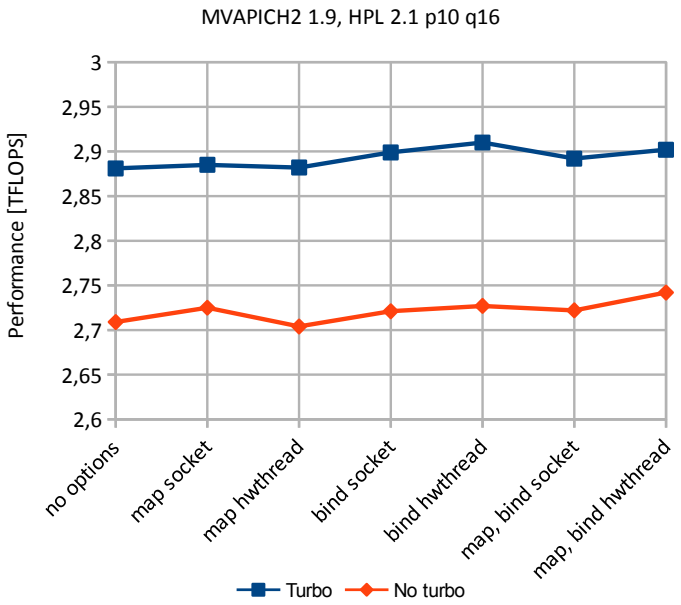

Figure 5.2: Performance measurements of turbo boost using MPVAPICH2 1.9

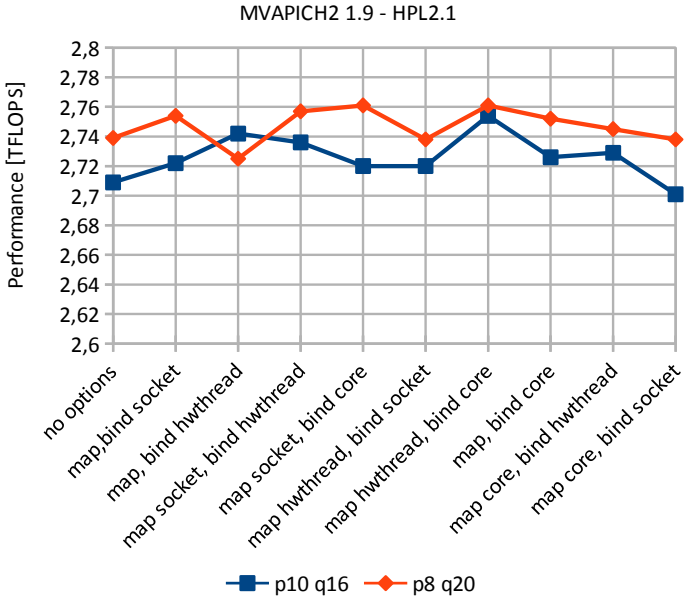

Figure 5.3: Performance measurements of different HPL input data configuration using MPVAPICH2 1.9

Figure 5.3 shows the results of comparison of two different HPL configurations (different Ps and Qs). The initial runs without any options already show better results for the configuration p8, q20. With the exception of one case, this configuration performed also

better with passed options. Only the mapping by hardware thread and binding to core could even out this difference.

The next Figure compares MVAPICH2 1.9 and OpenMPI 1.8.1 using the better HPL input configuration. There is a constant difference in performance results. The „no options“ case of OpenMPI is equal to the mapping by socket and binding to core due to the default configuration.

Mapping by socket and binding to hwthread seem to perform best, but regarding the deviations there does not seem to be a configuration for achieving outstanding results. However, MVAPICH2 was chosen as the implementation used in all applications.

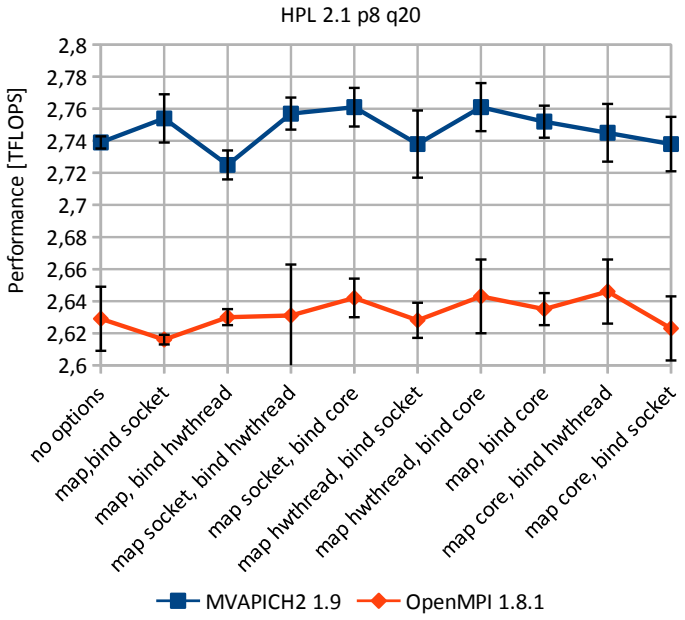

Figure 5.4: Performance measurements comparing MVAPICH2 1.9 and OpenMPI 1.8.1

Figure 5.5 compares the two versions of MVAPICH2. The results are nearly the same, so MVAPICH2 1.9 was chosen for reasons of stability.

Unfortunately non of the options were of benefit during the full HPL run (look up configuration in Section **??**). The results without any mapping or binding options using MVAPICH2 1.9 (3,125 TFLOPS) do not differ significantly from the tuned run with passed options `-map-by socket -bind-to hwthread` (3,124 TFLOPS).

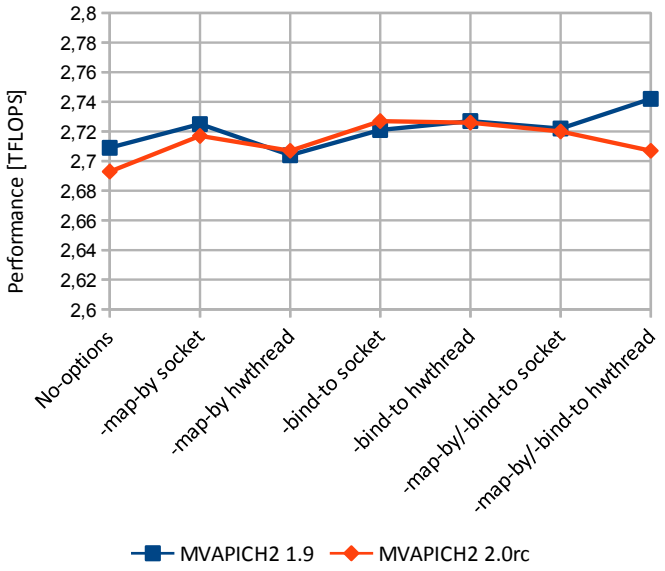

Figure 5.5: Performance measurements of different MPVAPICH2 versions

5.6.2 IMB

This benchmark developed by Intel® performs measurement tests for different communication types. The focus is on the performance of the whole system (including nodes, network and throughput) and the efficiency of the MPI implementation (int13).

The used part references to IMB-EXT - one-sided communications benchmarks.

Table reftab:test4 shows the results of OpenMPI 1.8.1 and MVAPICH2 1.9. Two cases were tested - 40 processes on 20 nodes and 160 processes on 8 nodes. The results are similar in both cases and are very different in terms of the implementations. MVAPICH2 seems to provide terrible performance results, which does not correspond to measurement of further benchmarks. It has to be analyzied whether the reasons are connected to with the wrong output, the wrong calculations or the failing InfiniBand support using this benchmark and MVAPICH2 1.9.

The results of OpenMPI look more realistic, but do not reach the native Infiniband performance of up to 6.8 GB/s. The performance on 8 nodes decreases to a peak of 5208 MB/s.

		OpenMPI		MVAPICH2	
#bytes	#repetitions	t_avg[usec]	Mbytes/sec	t_avg[usec]	Mbytes/sec
0	1000	1.50	0.00	6.85	0.00
1	1000	1.57	2.41	7.16	0.53
2	1000	1.57	4.84	6.85	1.11
4	1000	1.58	9.44	7.07	2.15
8	1000	1.59	19.07	6.76	4.51
16	1000	1.59	38.34	7.40	8.24
32	1000	1.62	74.57	7.07	17.25
64	1000	1.90	127.95	8.98	27.17
128	1000	2.57	188.67	10.12	48.20
256	1000	2.78	349.52	14.40	67.74
512	1000	3.03	640.40	18.69	104.43
1024	1000	3.40	1146.54	30.13	129.58
2048	1000	4.27	1823.18	52.48	148.76
4096	1000	8.01	1945.61	60.38	258.56
8192	1000	10.82	2879.12	72.73	429.23
16384	1000	18.10	3446.00	205.63	303.81
32768	1000	28.52	4371.22	355.23	351.74
65536	640	49.32	5055.78	668.83	373.69
131072	320	98.66	5046.04	1252.09	399.18
262144	160	182.60	5422.81	2546.73	392.30
524288	80	361.23	5416.42	4962.44	402.65
1048576	40	995.44	3902.63	10086.61	396.03
2097152	20	2857.47	2650.59	20323.17	393.23
4194304	10	5976.94	2459.00	40892.87	390.66
0	1000	1.53	0.00	38.41	0.00
1	1000	1.62	2.27	6.81	0.56
2	1000	1.63	4.56	6.78	1.12
4	1000	1.62	9.07	6.75	2.25
8	1000	1.65	17.90	6.67	4.55
16	1000	1.65	36.18	7.18	8.47
32	1000	1.67	70.73	7.17	16.97
64	1000	1.95	122.56	8.27	29.39
128	1000	2.61	185.57	10.04	48.54
256	1000	2.82	340.37	13.91	70.01
512	1000	3.05	627.64	17.89	108.82
1024	1000	3.45	1118.67	27.17	143.23
2048	1000	4.39	1749.31	44.16	176.61
4096	1000	8.06	1921.88	53.65	290.85
8192	1000	11.12	2775.77	71.09	438.71
16384	1000	19.22	3210.42	206.83	302.07
32768	1000	30.10	4108.20	387.20	322.70
65536	640	50.80	4880.29	822.11	303.88
131072	320	102.12	4817.41	1369.47	364.56
262144	160	188.82	5196.36	2619.96	380.69
524288	80	367.39	5208.50	5051.87	393.97
1048576	40	994.69	3881.50	10257.82	388.23
2097152	20	2862.74	2255.56	20514.22	386.42
4194304	10	5945.04	2393.42	40594.12	391.71

Table 5.3: Exchange benchmark with 40 processes on 2 nodes and 160 processes on 8 nodes using MVAPICH2 1.9 and OpenMPI 1.8.1

5.6.3 Third party results

All in this section presented information is provided by Julian Kunkel.

OpenMPI 1.8.1, MVAPICH2 2.0rc, MVAPICH2 1.9 with debugging support and Intel MPI 4.1 were compared using the osu-micro-benchmarks[11].

Figure 5.6 shows the results for latency measurements done by

```
1 mpiexec -np 2 --map-by node ./osu_latency
```

To start jobs using Intel MPI `mpdboot` is required. The following Listing shows a session example.

```
1 mpdboot --file=hostfile  -v --totalnum=$(wc -l hostfile|cut
    ↪ -d " " -f 1)
2 mpiexec -nolocal -perhost 1 -np 2  hostname
3 mpdallexit
```

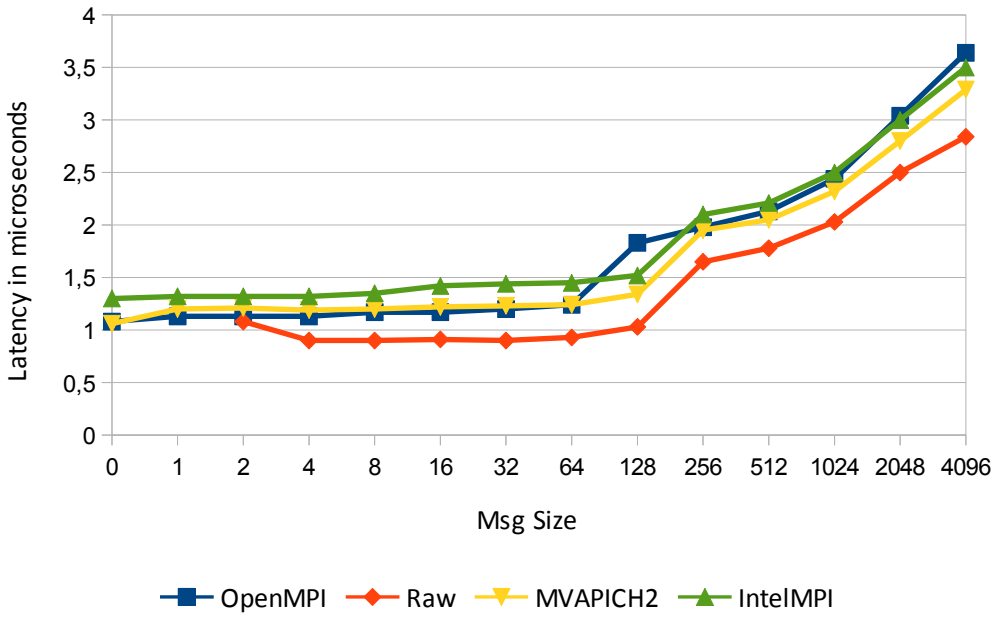

Figure 5.6: Latency measurements using different MPI implementations

MVAPICH2 performs best, while there were no significant differences between the versions 2.0rc and 1.9debug. On larger message sizes (not in the graphic) OpenMPI performed by far poorer (1184 μs vs. 675 μs using MVAPICH2).

The bandwidth benchmark was running with

```
1 mpiexec -np 2 --map-by node ./osu_bw
```

[11] http://mvapich.cse.ohio-state.edu/benchmarks/osu-micro-benchmarks-4.3.tar.gz

Figure 5.7 compares the throughput results of the implementations.

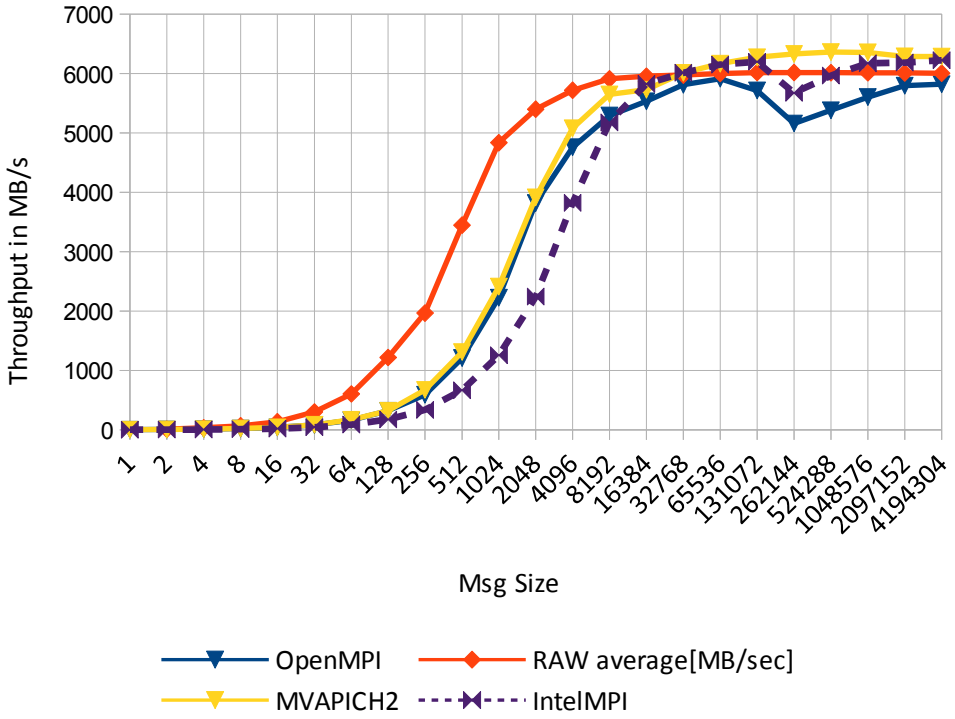

Figure 5.7: Throughput measurements using different MPI implementations

MVAPICH2 also performs best here, while the results are even better then the RAW average. On smaller message sizes its performance is as well as OpenMPI, on bigger sizes it is comparably well to Intel MPI, while OpenMPI and Intel MPI have a performance leak on around 262144 byte message size. The collective communication has not been examined.

6 Benchmarks

This chapter gives the general overview and some tuning advices for the benchmarks used during the competition.

6.1 HPCC

Author: Jennifer Nissley

6.2 Introduction and Motivation

The HPCC-Benchmark is an extension of the commonly used HPL. He combines 7 tests, that are also used indiviually to measure the performance of clusters. This combination is useful for measuring as many different patterns of communication and memory access on the system as possible with starting just one program.
It contains the following tests, which are measured in the units specified:

HPL: solves linear systems, TFLOP/s
DGEMM: multiplies matrices, GFLOP/s
STREAM: fetches data from the RAM and stores them back into it, GB/s(Bandbreite zum Speicher)
PTRANS: transposes matrices and tests communication, GB/s
RandomAccess: update data in the main storage, GUP/s (Updates pro Sekunden)
FFT: one-dimensional Fourier transform, GFLOP/s
b_eff: measures latency and communication-bandwith, GB/s usec (microseconds)

Some of the tests (DGEMM, STREAM, FFT and RandomAccess) are not only performed on the system as a whole, but also on single CPUs, to measure the peak performances that are possible when communication is not needed. DGEMM and STREAM are performed only on single CPUs, FFT and RandomAccess are also performed on the whole system. [8]

HPL is only measuring the maximum number of floating-point operations a system can perform per second. It is called a weak scaling test, because the communication between processes does not increase if the problem size and the number of processes are equally increasing. This is not the case with most of the real programms, that are

used in HPC. They tend to need more communication for a single process when the number of processes is increasing. FFT and PTRANS are testing this scenario. PTRANS distributes the matrix between processes and then transposes the columns with the rows. So not only processes wich hold elements that are neighbors in the matrix need to exchange elements with each other, but every process with every other.

The source code is available on the HPCC website [7]. We used the newest version available at the time of the challenge, hpcc-1.4.3.

To get the benchmark running, one needs to create a build script and a file for the parameters given to the program, like how big the problem should be, the input file. The parameters defined in the input file can also be used to run the HPL alone.

6.3 Build Script

To compile the source code on a specific system, there are some build script examples in the hpcc-folder hpl/setup/. One of them has to be adjusted for the system and then copied into the hpl-folder. All of these files begin with Make. The name after the point is used in the compiling-commando to find the right build-script. To compile the program, one calls Make=name while being in the top hpcc-folder, where the Makefile is located. So it is possible to have more than one build-script and decide at compiling which one to use. This is useful for benchmarking, because the test has to be compiled with different libraries, compilers and compileroptimisations to find the best configuration of the system.

We didnt use different build-scripts, because it was possible to use our module files to change the system variables that specified which libraries should be used. So we didnt have to specify which libraries to use and where they are located in the build script.

Normally, one would specify where the MPI-, Algebra- and FFT-libraries are located. We loaded the right module files before starting the compilation instead. If you want to use FFTE for the Fourier-transform, you dont have to specify it in the build script, because it is already implemented in the hpcc. If FFTW should be used, there has to be a -DUSING_FFTW under CCNOOPT and it has to be specified where to find the library or in our case, the right module has to be loaded.
As compiler we specified mpicc, which calls the right C compiler that is loaded as a module and gives him all of the specified information about optimisations and options that are to be used and links against the MPI-libraries. To use all possible code-optimisations, we used -O3. All warnings have been enabled with -Wall. We used native for march and mtune, so the compiler searches for the right architectural details in the system. It would also be possible to specify them yourself, but there is always the possibility to overlook details, which are important for optimisations.

The Benchmark was tested with different libraries. The Intel-Libraries got left out,

because it couldnt be figured out how to activate the FFT-libraries with them, even after following the instructions on the official Intel-site [9]. Without them, the HPCC cant be compilated. They also got bad results in the testing with the HPL, compared to blas and mvapich or openmpi.

We didnt test all of the preprocessor-options, that can be specified in CCNOOPT and CCFLAGS, because this would have taken too much time and the performance is mostly affected by which libraries and compilers are used and by the options in the input file. Possible options we could have used are: HPCC_FFT_235: sizes of vectors are $2^x * 3^y * 5^z$, without it they are only powers of 2
HPCC_FFTW_ESTIMATE: FFTW is called with FFTW_ESTIMATE, without it is called with FFTW_MEASURE, makes the performance worse in most cases, but the runtime can get quicker and the distribution in memory better
HPCC_MEMALLCTR: own custom memory allocator is used
USE_MULTIPLE_RECV: use multiple non-blocking Receives at once, instead of one at a time
RA_SANDIA_NOPT: use another implementation of RandomAccess, which uses a virtual Hypercube
A_SANDIA_OPT2: use another implementation of RandomAccess, which is optimized for a multiple of two of processors
RA_TIME_BOUND_DISABLE: use RandomAccess without a time out, not recommended in most cases
USING_FFTW: use FFTW instead of FFTE

6.4 Input File

The problem sizes are only specified for HPL and PTRANS, for the other tests the sizes are adjusted to take the same space in memory. The standard practice here is to fill the whole main memory, to let the processor run as long as possible. This way, the pipelines of the processor have enough time to get filled and after they have been filled the processor has as much time as possible to run on full speed to compensate for the slow start. The problem size also cant be too big, because otherwise parts of the matrix have to be put on hard drives and getting them back to work with them wastes a lot of time. Because there are also processes from the operating system and program-code in main memory, it can only be filled around 80-95% with the hpcc matrices. On the website [10], it is recommended to use smaller sizes than the sizes that can be used if you run the HPL alone, because FFT needs more space than the other tests. 80% was the size that was recommended by most of the websites on the topic.

In the hpcc-folder, there is a text file hpccinf.txt, that has to be modified. The most important lines are:
3: name of the file, in which the output should be written

4: where the output should be directed to, possibilities are stderr(7), stdout(6) and a file (neither 6 nor 7)
5: number of problem sizes
6: size of matrix N, matrix will be NxN big
7: number of block sizes
8: block sizes
9: how the processes should be distributed among the nodes, only important if there is more than one process on one node: row or column (row is recommended)
10: number of process grids
11-12: process grids (P=rows, Q=columns)
for PTRANS:
33: number of additional matrix sizes (if necessary)
34: matrix sizes
35: number of block sizes
36: block sizes

More information on that can be found under [1].
Our final input file listed following options: Ns = 296400, NBs = 192, Ps = 8, Qs = 20.

To test the different libraries against each other, we used a small size at first, because the test can take a few hours, these were:

Compiler	Algebra	MPI	FFT
gcc	blas	openmpi	ffte
gcc	blas	openmpi	fftw
gcc	blas	mvapich	ffte
icc	mkl	intelmpi	ffte (in MKL)

(FFT stands for Fast Fourier Transform)
FFTE obtained better results than FFTW with openmpi, therefore and because a lot of sources say FFTE runs better on a lot of CPUs [12], only FFTE was used after this test. Openmpi obtained poorer results than Mvapich. The combination of Intel libraries with the icc was better than the combination gcc + openmpi, but worse than gcc + mvapich. The results were much better with 230.000 as a problem size than with 100.000.

We took the block size that was recommended on [3] and already proofed to be beneficial for the HPL. The blocking size should be as big as possible to avoid too much communication between processes. If it is too small, the matrix get parted into small pieces and those are distributed among the processes, so if they calculate something in the matrix, they need to get a lot of elements from other processes, there is a lot of communication. [2] It also should be small enough to let as much processes work simultaniously as possible, since one block cannot be divided and distributed between processes.

The process grid serves as a communication pattern for the processes. The HPL does exchanges of data after every iteration, therefore, every column of the grid does a broadcast to every other column. The processes in the columns transmit data between them

in single messages. The runtime of PTRANS is dependant on the process grid [4]:

number of iterations: $\frac{kgV(P,Q)}{ggT(P,Q)}$, number of communication pairs: ggT(P,Q), if $ggT(P,Q) > 1$

For our cluster there where the possibilities (8,20) and (10,16). We chose these two options, because it was recommended, that P and Q should be as even as possible and Q should be a little bit bigger than P. It was calculated which one would be better suited for PTRANS:

kgV(8,20) = 40 , ggT(8,20) = 4 ⇒ iterations: 10, communication pairs: 4
kgV(10,16) = 80 , ggT(10,16) = 2 ⇒ iterations: 40, communication pairs: 2

The grid we chose, (8, 20) was also the one that was best suited for our Infiniband-wiring, P=8 nodes with Q=20 processors each. This is beneficial, because every node is a column in the grid, which broadcasts to every over column over Infiniband, which is fitted to transmit such a big message. The single processes can then use the short routes on their node to transmit the data over small messages between them. We proved this theory with the HPL and the HPCC by comparing with other process grids.

The rest of the input data was generated with the website [3]. Testing all combinations would have taken too much time.

We calculated how big the problem size would have to be to fill 80% of the main memory:
$\sqrt{\frac{128*10^9*8bit*8}{64bit}} * 0.8$ (GB = Gigabyte) = 320.000. (Every element of the matrix needs the space of 64 bit. The total main memory is $128 * 10^9$ Byte per node, of which we have 8. We take 80% of it, so *0.8. By taking the square root of the whole space N*N, we get the problem size N.)

The calculation should have been done with GiB instead of GB, because vendors normally mean GiB when they use GB, so instead of 10^9, we should have used 2^30:
$\sqrt{\frac{128*2^30*8bit*8}{64bit}} * 0.8 = 331.588$
On our final run, we used N=296400, which was given in the table on the website [5] (with the input: 8 cores, 20 processes per core, 128 RAM per core). This value takes about the same space and is better divisible by the blocking factor 192, so that the data can be distributed among the processes, so that every process has the same amount of data in every iteration and no time is wasted by one process waiting for the others. However, we looked at the wrong column, it should have been N=296448.

We also tried to use Hyperthreading (two threads on one core), but the runtime didnt get any better through it. The grid had to be changed, because we had more threads, we used: P=8, Q=40.

6.5 Code Optimization

The following functions are allowed to be replaced by own optimized ones:

HPL: HPL_pdgesv(), HPL_pdtrsv()
PTRANS: pdtrans()
STREAM: tuned_STREAM_Copy(), tuned_STREAM_Scale(), tuned_STREAM_-Add(), tuned_STREAM_Triad()
RandomAccess: Power2NodesMPIRandomAccessUpdate(), AnyNodesMPIRandomAccessUpdate(), RandomAccessUpdate()
FFT: fftw_malloc(), fftw_free(), fftw_create_plan(), fftw_one(), fftw_destroy_plan(), fftw_mpi_create_plan(), fftw_mpi_local_sizes(), fftw_mpi(), fftw_mpi_destroy_-plan()

There are some rules to follow. Calculations have to be done with 64 bit precision. New algorithmns for the calculations have to be approved by the HPCC-commitee and they need to get to the same result as the current algorithmns. For example, the Strassen algorithmn is forbidden in HPL because it produces a less precise solution. It is forbidden to use knowledge about the result or to use a more intelligent algorithmn, that doesnt need as much calculations. It is allowed to do the calculations in a way that is better suited for the machine your working on, but you cant just use less calculations. If you want to use your own libraries or compilers, they have to be available to the public or they have to be made available a few months after the submission of the results. This critearia is also met if they can be bought somewhere, they dont have to be free. The point here is only to ensure nobody uses libraries that arent tested or accessible for validation.
So in our case, it would have also been possible to use FFTW3 instead FFTW2, which the HPCC is capable of using without code modification. The FFT library is only used in the FFT-test and all of the function in which it is used are allowed to be modified. These code modifications are necessary, because FFTW3 has another Interface than FFTW2. FFTW3 has a whole new syntax, so the function calls cant simply be substituted. If one has a full understanding of the code, it is possible by using the directions on the website [11]. Because we didnt fully understand the decisions the programmers of the code made and didnt know what would be beneficial for our system, we didnt do this.

6.6 Results

The results of the test can be found in a file hpccoutf.txt. It is automatically created in the hpcc-folder.
Here are some interesting parts of the results from our final run at the cluster challenge:

```
1 StarDGEMM_Gflops=1.06518
2 SingleDGEMM_Gflops=1.19744
```

```
3
4  PTRANS_GBs=1.53021
5
6  StarRandomAccess_GUPs=0.0963877
7  SingleRandomAccess_GUPs=0.167188
8
9  Minimum Copy GB/s 3.074954
10 Average Copy GB/s 3.119951
11 Maximum Copy GB/s 3.179126
12 Minimum Scale GB/s 3.086960
13 Average Scale GB/s 3.120812
14 Maximum Scale GB/s 3.236323
15 Minimum Add GB/s 3.502806
16 Average Add GB/s 3.571931
17 Maximum Add GB/s 3.627865
18 Minimum Triad GB/s 3.569321
19 Average Triad GB/s 3.628885
20 Maximum Triad GB/s 3.906788
21
22 HPL:
23 ================================================
24 T/V        N      NB   P  Q   Time         Gflops
25 ------------------------------------------------
26 WR11C2R4  296400 192  8  20  5589.36      3.106e+03
27 ------------------------------------------------
28 Finished   1 tests with the following results:
29            1 tests completed and passed residual checks,
30
31 tarSTREAM_Copy=3.11995
32 StarSTREAM_Scale=3.12081
33 StarSTREAM_Add=3.57193
34 StarSTREAM_Triad=3.62889
35 SingleSTREAM_Copy=11.2584
36 SingleSTREAM_Scale=11.3952
37 SingleSTREAM_Add=11.5407
38 SingleSTREAM_Triad=11.6622
39
40 MaxPingPongLatency_usec=3.24845
41 RandomlyOrderedRingLatency_usec=14.4998
42 MinPingPongBandwidth_GBytes=3.80608
43 NaturallyOrderedRingBandwidth_GBytes=1.20773
44 RandomlyOrderedRingBandwidth_GBytes=0.216748
45 MinPingPongLatency_usec=0.804663
46 AvgPingPongLatency_usec=1.88951
```

```
47 MaxPingPongBandwidth_GBytes=7.44992
48 AvgPingPongBandwidth_GBytes=5.35664
49 NaturallyOrderedRingLatency_usec=2.09808
```

6.7 Conclusion

We didnt optimize every parameter for our cluster, but we did for the ones that where most important for the performance as stated by our sources. Testing every possible combination would have cost to much time and wouldnt have greatly affected the result. We didnt understand the code and the process of optimizations for a certain architecture good enough to find useful code optimizations. In our understanding of the challenge, it wasnt demanded to do any code optimizations in the HPCC, because we didnt have to provide the code we used, which you would normally have to do when you use code optimizations and because most of the other teams didnt do that as well.

6.8 Sources

[1] http://www.netlib.org/benchmark/hpl/tuning.html

[2] http://www.docstoc.com/docs/74480101/Analyze-and-optimize-the-HPL-High-Performance-Linpack-benchmark

[3] http://www.advancedclustering.com/act-kb/tune-hpl-dat-file/

[4] PARALLEL MATRIX TRANSPOSE ALGORITHMS ON DISTRIBUTED MEMORY CONCURRENT COMPUTERS, Jaeyoung Choi, Jack J. Dongarra, David W. Walker, 1993

[5] http://hpl-calculator.sourceforge.net

[6] http://www.spec.org/workshops/2007/austin/slides/Keynote_Jack_Dongarra.pdf, Folie 9

[7] http://icl.cs.utk.edu/hpcc/software/index.html

[8] HPC Challenge v1.x Benchmark Suite, SC|05 Tutorial — S13, Dr. Piotr Luszczek, Dr. David Koester, 2005

[9] https://software.intel.com/en-us/intel-sdp-home

[10] http://scicom.cmm.uchile.cl/pmwiki.php/Main/Levque-64Cores-HPCC

[11] http://www.fftw.org/doc/Upgrading-from-FFTW-version-2.html

[12] http://www2.ccs.tsukuba.ac.jp/SC/sc2008/17.pdf

6.9 HPCG

Author: Johann Weging

From the ISC'14 instructions:
HPCG is a software package that performs a fixed number of symmetric Gauss-Seidel preconditioned conjugate gradient iterations using double precision (64 bit) floating point values.

The HPCG was the third benchmark that had to be run at the competition. Because of a of some miss information previously to the competition our team didn't know about this application. And this could be called the secret application for the event.

6.9.1 Building

To configure HPCG uses different Makefiles which resides inside the *setip* directory. The final runs where done with GCC and MPICH. Listing 6.1 shows the necessary changes applied to the make file. HPCG must be linked against MPICH and the OpenMP support is disabled.

Listing 6.1: GHPC Makefile changes.

```
1 # file: hpcg-2.4/setup/Make.gcc_mvapich2
2 MPlib        = -lmpich
3 HPCG_OPTS     = -DHPCG_NOOPENMP
```

The simplest way to build HPCG is to create a build directory for specific build some where, in this case inside the HPCG directory (Listing 6.2). To configure HPCG the path to *configure* and the modified make file is given.

Listing 6.2: Building HPCG.

```
1 $ mkdir build_hpcg
2 $ cd build_hpcg
3 $ ../configure ../setup/Make.gcc_mvapich2
4 $ make
```

After the build is finished a *bin* directory is created inside the build directory with the *xhpcg* executable.

6.9.2 Benchmarking and Results

Listing 6.3 shows the input file. At the beginning there was some confusion about how to tune the input but a short chat with a member of the SCC-Council revealed, that the input file shouldn't be changed and HPCG should be run as is.

Listing 6.3: HPCG input file *hpcg.dat*.

```
1 HPCG benchmark input file
2 Sandia National Laboratories; University of Tennessee,
    ↪ Knoxville
3 104 104 104
4 900
```

To run HPCG a directory is created with the current slurm job id and the input file is copied into it. The working directory is changed to the newly create directory and the benchmark is run. The correct MPI implementation hat to be loaded (Listing 6.4).

Listing 6.4: HPCG job script.

```
1  #!/bin/sh
2  # Allocate for 64 tasks on 8 nodes.
3  #SBATCH -N 8 -n 160
4  #SBATCH --time=40
5
6  if ! (mkdir $SLURM_JOB_ID && cp hpcg.dat
     ↪ $SLURM_JOB_ID/hpcg.dat); then
7      echo "Failed copying the hpcg.dat file"
8      exit 1
9  fi
10
11 cd $SLURM_JOB_ID
12
13 . /etc/profile.d/modules.sh
14 module purge
15
16 module load gcc/4.9/4.9.0
17 module load gcc/4.9/mvapich2/1.9/1.9
18
19 mpiexec -map-by socket -bind-to hwthread
     ↪ /musorka/home/weging/hpcg/gcc_mvapich2/xhpcg
```

Table 7.1 shows the results of benchmarking runs with different problem sizes. In fact the default problem size delivers the best performance with 72.573 GFLOP/s.

Problem Size	GFLOP/s
192 192 192	71.323
160 160 160	71.2476
150 150 150	71.2369
120 120 120	71.25
104 104 104	72.573

Table 6.1: HPCG Results: Problem size on the left and the reached performance in GFLOPs on the right side.

The benchmark wasn't known before hand but like the council member said, there shouldn't be any tuning done to the input. It was just intended to build the benchmark and to one or two quick runs, so the whole process will not take to much time.

7 Applications

7.1 Quantum Espresso

Author: Marian Tietz

7.1.1 Overview

Quantum Espresso (QE) [1] is a suite of applications to do electronic-structure calculations and nano-scale material modeling. It is well supported on many platforms and offers a wide range of support for current math libraries. The project is well documented and especially the PW user guide [2] was of great help.

QE itself recommends to use ScaLAPACK, at least for large matrices ($> 1000 \times 1000$) or many processors (> 512) (Gia14). Since these criteria were unlikely to be met during the competition, it was decided to not use ScaLAPACK, also because this library was disqualified before in subsection 4.2.1.

In terms of other libraries, QE is very liberate but recommends, amongst others, GotoBLAS2, Intel MKL, ATLAS and FFTW3. The final choice, as discussed in chapter 4, was to use OpenBLAS, which is based on GotoBLAS2, and FFTW 3.3.4. MVAPICH2 was used as the MPI implementation.

Since the challenges during the competition were likely to use the Plane-Wave Self-Consistent Field tool offered by the QE suite, it made sense to do pre-testing to learn the various tuning parameters of the PW tool. This testing and tuning is described in subsection 7.1.3, followed by a section discussing the actual competition results and finishing off with a conclusion.

7.1.2 Tuning parameters

Quantum Espresso, especially the plain wave computation tool *PW* takes a few parameters which influence the parallelization and diagonalization as documented in (Gia14). QE follows a fairly deep MPI hierarchy which is pictured in Figure 7.1. When implemented in the corresponding tool there are parameters to influence how many processes are dedicated to the specified MPI level to improve performance in that area.

As shown in Figure 7.1, there are several levels to split the number of available MPI processes. These layers can be configured individually and serve a special purpose. The

[1] `http://www.quantum-espresso.org/`

[2] `http://www.quantum-espresso.org/wp-content/uploads/Doc/pw_user_guide.pdf`

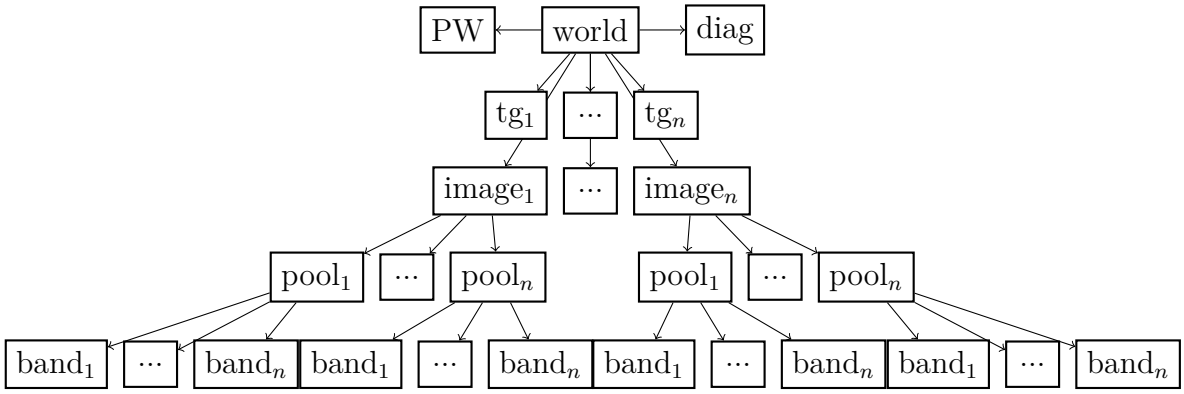

Figure 7.1: Process hierarchy of Quantum Espresso. While *world* includes all processes, each subsequent layer splits up its parent's process count equally whle *PW* and *diag* are configured independently. tg_i are groups used to parallelize FFT tasks, *diag* is a global group dedicated to linear-algebra.

purpose of each layer and how it can be configured according to (Gia14) is explained in the following.

- world: Corresponds to `MPI_COMM_WORLD` and includes all available processes.
- images (`-nimage`): An image corresponds to a self-consistent or linear-response calculation, loosely coupled to other images.
- pools (`-npools`): Each pool takes care of a group of k-points to compute these in parallel. The number of k-point groups depends on the input data and the number of pools cannot exceed the number of k-points.
- bands (`-nbands`): The pools can be further partitioned to band groups where each band group is taking care of a group of Kohn-Sham wave-functions.
- PW: Plane-wave computations are distributed over all available processes. This behavior cannot be influenced.
- tasks (`-ntg`): In order to allow good parallelization of the 3D FFT when the number of processors exceeds the number of FFT planes, FFTs on Kohn-Sham states (the number can be determined by the output of PW) are redistributed to task groups so that each group can process several wave functions at the same time.
- linear-algebra group (`-ndiag`): Independent of other parallelization levels, this level is responsible for diagonalization of arrays whose dimension is the number of Kohn-Sham states. Arrays of data are distributed across a square grid of processes, therefore the parameter is given in the format n^2. It defaults to the biggest possible size n where $n^2 <$ number of total processes.

An example call of the PW utility could look like this:

```
$ mpirun -np 1024 pw.x -nimage 4 -npools 4 -ntg 2 -ndiag 121
```

In this example the number of images is 4, meaning that each image contains $1024/4 = 256$ processes. These 256 processes are further divided into 4 pools, resulting in 64 processes per pool. Plane wave computations are done using the processors of all pools but 4 k-point groups can now be computed in parallel. (O'G13) The pools are further divided in two task groups for each pool, having 32 processes for each task group.

It is noteworthy that there is low communication between images and their pools but the processes in pools communicate heavily. This means that, for optimal performance, pool processes should be on the same CPU, mainboard or at least be connected over another low-latency medium.

7.1.3 Pre-competition benchmarks

To test the performance of the setup there are a few official benchmarks with references for a number of different systems one can compare to. In this section the AUSURF112 benchmark is used since it is medium sized, therefore finishes fairly quickly but can still be run in parallel.

QE was built with the following commands:

```
module load gcc/4.9/mvapch2/1.9/1.9
module load gcc/4.9/mvapich2/1.9/fftw/3.3.4
export MPI_LIBS="-L /opt/gcc/4.9/mvapich2/1.9/lib/"
configure --enable-parallel
make pwall
```

Several parameters and their practical effectiveness were tested. The results are presented in Figure 7.2. It can be seen that hyper-threading (np $>$ 160) does not have a positive impact on the performance. Small diagonalization grids lead to worse performance while at a certain point performance is not improved with bigger process grids.

7.1.4 Competition results

During the competition 3 tests were assigned to be computed by the teams. Each test had different characteristics so that different tuning parameters had to be applied. Since these measurements were taken under the pressure of the competition, not every promising parameter combination could be tested.

The first test did not allow for k-point parallelization since it used gamma points, therefore the number of pools was irrelevant. With a relatively high number of Kohn-Sham states (1144) and nothing else to specially parallelize the major impact made increasing the diagonalization grid size. An excerpt of the results acquired during the competition is shown in Figure 7.4.

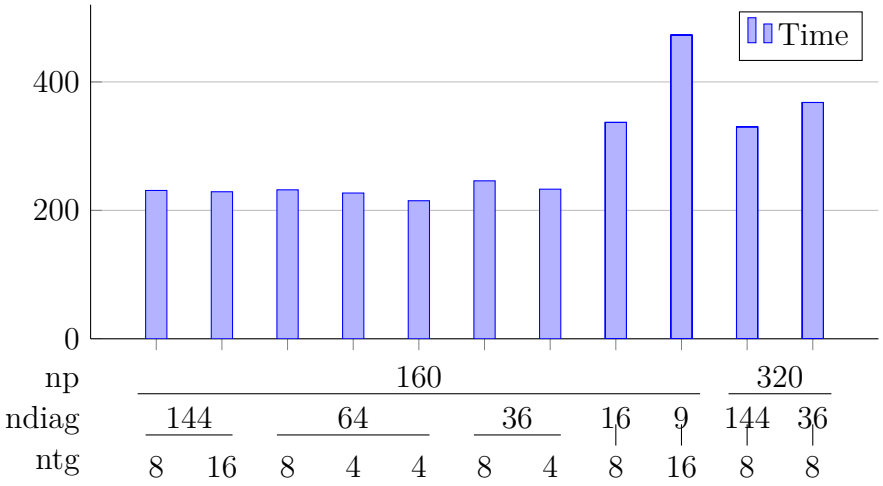

Figure 7.2: Results of AUSURF112 benchmark on the competition hardware with different parameters. The best result was achieved using a 8×8 diagonalization process grid, 4 task groups, each consisting of $160/4 = 40$ processes.

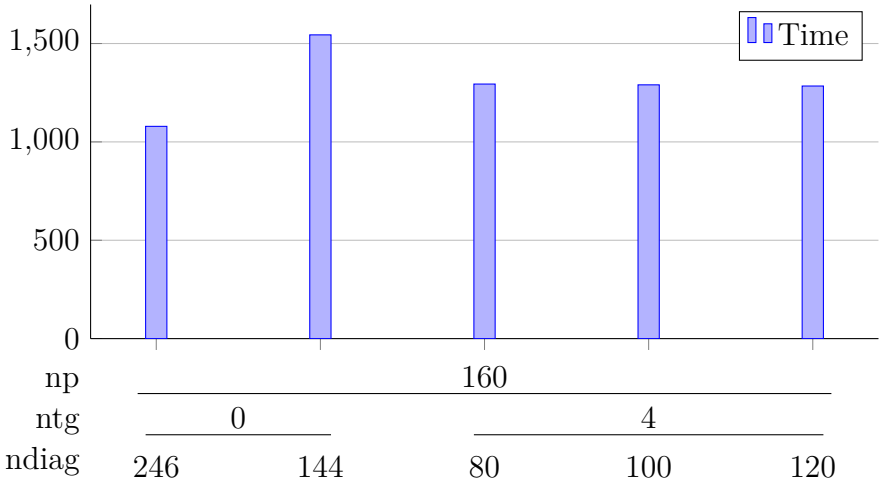

Figure 7.3: Running times of the first QE assignment during the cluster challenge. The most impact on performance was made by increasing the diagonalization grid size.

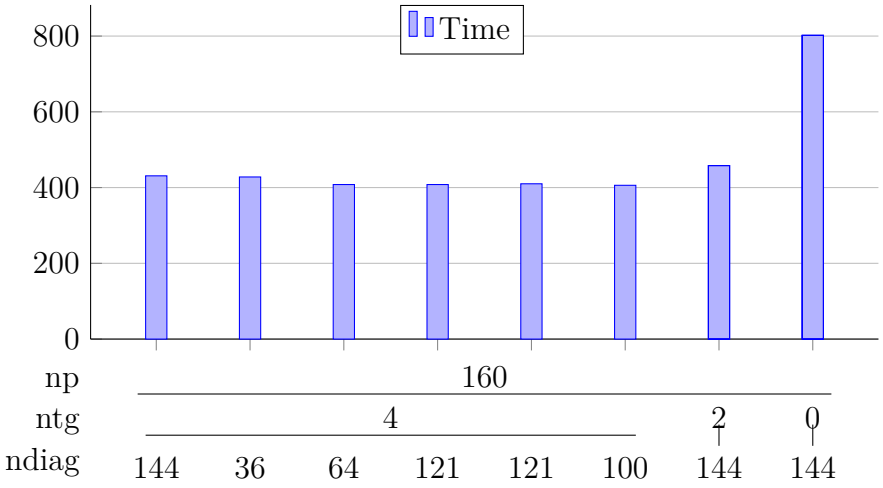

Figure 7.4: Various results of the second assignment. Grouping tasks is way more effective than in the first assignment, probably due to reduced FFT dimensionality.

The second assignment did not allow for k-point parallelization for the same reason as in the first assignment. In fact, the atomic configuration of the two assignments is the same, just the parameters changed. The most important changes are the reduction of FFT and PW dimensionality and the number of iterations which goes up from 10 to 18, leading to potentially longer runtime and higher precision of the result to compensate the lower dimensionality. Different parameter configurations and run-times are shown in Figure 7.4. The reduction of FFT dimensionality might explain why the task groups are more effective than in the test before where the FFT dimensionality exceeded the number of processes. This is also discussed by (Lar13).

The third and last assignment was not finished on the machine. At this point in time the system encountered severe problems with the file system which rendered the system useless until rebooted, since this was a kernel issue.

It can be said that even though the power budget was exhausted by applications such as HPCC and HPL, nearly all runs of Quantum Espresso were at 2.6 kW, well 400 Watt below the threshold. In the future it may be possible to utilize the CPU's turbo mode on demand. During the competition this was attempted but in a non-consistent, experimental way which can be improved upon.

7.1.5 Conclusion and Outlook

Although QE is well documented and offers a great deal of information to aid in parallelizing the application, the process is still non-trivial to achieve optimal results, especially when the parameters that influence the computational complexity are not fully understood. It is still possible, however, to achieve pretty good results with the guidelines offered by QE's documentation and the analytical output of the application itself.

Since ScaLAPACK was disregarded at the beginning although it is recommended

for large input data for diagonalization in Quantum Espresso, ScaLAPACK should be considered for testing in combination with Quantum Espresso in the future. In case accelerators, such as GPUs are used, QE itself should not pose a problem since it already offers support for GPUs.

7.2 Gadget

Author: Johann Weging

From the ISC'14 Gadget instructions:
GADGET is a freely available code for cosmological N-body/SPH simulations on massively parallel computers with distributed memory. GADGET represents fluids by means of smoothed particle hydrodynamics (SPH). The code can be used for studies of isolated systems, or for simulations that include the cosmological expansion of space, both with or without periodic boundary conditions. GADGET can therefore be used to address a wide array of astrophysically interesting problems, ranging from colliding and merging galaxies, to the formation of large-scale structure in the Universe. `http://www.mpa-garching.mpg.de/gadget/` *GADGET-3 can be obtained from this location:* `http://www.prace-ri.eu/ueabs#GADGET`

7.2.1 Building

After the source code of gadget was downloaded, the file *run.c* has to be replaced by the file provided by the SCC-Council. The next step is to add a system configuration to the Makefile which contains the necessary compiler flags. Listing 7.1 shows the configuration added for GCC, FFTW and MPICH2 although the GNU Scientific Library (GSL) (gsl14) is required by Gadget.

Listing 7.1: To compile Gadget a new SYSTYPE needs to be added to the Makefile.

```
1  ifeq ($(SYSTYPE),"gcc_mvapich2")
2  CC       =  mpicc -DHAVE_HDF5 -DH5_USE_16_API -lhdf5 -lz
3  CXX      =  mpicxx
4  OPTIMIZE =  -O3 -mtune=native -march=native
5  GSL_INCL =
6  GSL_LIBS =  -lgsl
7  FFTW_INCL=
8  FFTW_LIBS=  -ldrfftw_mpi -ldfftw_mpi -ldrfftw -ldfftw
9  MPICHLIB = -lmpich
10 HDF5INCL =
11 HDF5LIB  = -lhdf5 -lz
12 endif
```

Listing 7.2 shows the make command to build Gadget. The required *Config.sh* and the *SYSTYPE* is passed to the make command.

Listing 7.2: Calling make for Gadget with configuration and the new SYSTYPE.

```
make CONFIG=Config-Large.sh SYSTYPE=gcc_mvapich2  -FFTW3
```

7.2.2 Running

The provided ICs folder has to be copied in the same directory as the *param.txt* and the path of the *param.txt* has to be passed to the Gadget executable.

```
#!/bin/sh
# Allocate for 64 tasks on 8 nodes.
#SBATCH -N 8 -n 160

mkdir $SLURM_JOB_ID
cd $SLURM_JOB_ID
mkdir output

. /etc/profile.d/modules.sh
module purge

module load gcc/4.9/4.9.0
module load gcc/4.9/gsl/1.16
module load gcc/4.9/mvapich2/1.9/1.9
module load gcc/4.9/mvapich2/1.9/fftw/2.1.5
module load gcc/4.9/mvapich2/1.9/hdf5/1.8.13

mpiexec -map-by socket -bind-to hwthread
   ↪ /musorka/home/weging/gadget3/gcc_mvapich2_fftw2/gadget
   ↪ ../param.txt
```

Sadly Gadget didn't finish because of the ZFS I/O bug.

7.3 OpenFOAM

Author: Jakob Lüttgau

OpenFOAM(R) (Open Field Operation and Manipulation) provides a toolbox for CFD (Computational Fluid Dynamics). The package is free and open source and enjoys a large user base from commercial and academic organisations. Applications range from chemical reactions, heat transfer and fluid flows over to the simulation of solid dynamics and electromagnetics. It provides over 80 solver applications and more than 170 utility applications for pre- and post processing.

OpenFOAM has several dependencies 3rd party dependencies and requires cmake for compilation. The full list of dependencies includes flex, bison, cmake, zlib1g-dev, qt4-dev-tools, libqt4-dev, gnuplot, libreadline-dev, libncurses-dev, libxt-dev as well as libscotch-dev, libopenmpi-dev, libcgal-dev.

Qt is needed for GUI tools which can be used for pre and post processing. A task that was also required to solve the OpenFOAM challenge at SCC. Libscotch assists in distributing work among many nodes (e.g. sequential and parallel graph partitioning). CGAL provides a library for geometry algorithms.

The competition guidelines did reveal only little about how challenges in OpenFOAM could look like, resulting in a rather heavy weight installation on the cluster.

OpenFOAM provides good documentation once installed, but requires a lot of training time since it is very extensive and rich in features. Enough time should be reserved to work through the examples and to understand the available parameters needed to adjust a problem to the system specifications.

Installation

OpenFOAM can be installed from package management on Ubuntu Systems. The specified dependencies might not be satisfied by the Qlustars default package repositories. For the particular case installation from source is advisable anyways especially since multiple MPI backends are available.

OpenFOAM is bundled into two source packages, one for the OpenFOAM application, and another one providing third party dependencies. If in doubt build the dependencies from the third party package.

Either way, to build OpenFOAM or the dependencies it is required to adjust the bashrc located at etc/bashrc in the source package. Here it is possible to set the paths for compiler and libraries to be used for building. Problems with the executables most likely stem from a mistake made during this configuration.

OpenFOAM uses the wmake build system which come bundled with the source, it is possible to build OpenFOAM in parallel by announcing SSH nodes with corecounts using environment variables.

8 Future Work

Author: alle

The key to success - in addition to competitive hardware - is the best preparation, deep analyses and wide testing of everything that can be tested.

Since the number of combinations of hard- and software components is too large to test them all, it is most important to draw up a strategy and decide which is the favored approach. To begin structuring with the system setup, it is a good practice to test the components separately and to note down the separate power consumption and the performance rate. These two characteristics are most relevant especially in cases of surprise tasks. Having all test results can help to ease the decision-making whether to install the components - hard- or software, or to provide specific options depending on the task. Further characteristics like the robustness or flexibility should be well known before the competition starts.

Following points should be considered in more detail:

- Power consumption of
  - Head node
  - Compute nodes
  - Switches
  - HDDs or SSDs
  - Cooling
  - CPUs separately and depending on the configuration (hyper threading, turbo boost, etc.)
- Robustness of
  - Hardware (enough cooling)
  - Operating system (kernel bugs)
  - File system
- Performance of
  - Configured hardware (regarding theoretical peak performance)
  - Chosen libraries, software and corresponding tuning options
  - I/O or file system

8.1 Power Consumption

Considering the limits to the power budget and especially the last competition task it would have been helpful to try out a system configuration without a head node. The primary disadvantages are less comfort and more effort for running the system and the applications. On the other hand, it would be possible to increase performance by adding a spare node when the power budget allows it or to enable turbo-boost on certain occasions. It is noteworthy that recent SLURM versions as well as other job management systems allow for power budgeting and dynamic clock scaling. Dynamic clock scaling could potentially achieve a finer grained automatic control than could be achieved by manually clocking the nodes.

Cooling is another factor which should have been considered in the this competition. Based on the fact that the system could safely run for about half an hour without or with reduced cooling, the chances to win would significantly increase. To achieve it, the system should be extensively tested in the first place, not only in a secure environment, but also stress tested with regard to temperature differences and high load during a real competition.

8.2 File System

The decision about the file system is mainly based on the experience with the applications and on a loose prediction of what can be expected as secret applications. Since the decision is so dependent on the applications, it is necessary to find out if the published applications provide parallel I/O. The next step would be evaluating of costs and benefits of a parallel file system of which two candidates would be Lustre[1] and OrangeFS[2].

The main reason not to use a parallel file system, whether or not parallel I/O is expected, is the complexity of use and setup. A potential drawback of Lustre, for example, is that it requires a relatively old kernel. "The last client-side software was updated in March 2013 to work with Linux kernels up to version 3.6. "

...

8.3 GPU

The decision to leave out GPUs was, in retrospective, a reasonable one. There were many challenges to overcome and another software layer for GPU or accelerator support might have been too time consuming. Although GPU support has improved greatly and the competition is laid out to support GPUs, it is still a significant amount of additional work, especially for the first participation in this competition.

Now that experiences with a CPU-only system were made and the general proceeding of the competition is known, accelerators such as GPUs can be discussed. Software

[1] `http://lustre.opensfs.org/`
[2] `http://www.orangefs.org/`

wise there is PLASMA for a GPU/accelerator-friendly alternative to LAPACK and, for example, CUBLAS for nVIDIA graphics cards.

It is noteworthy that the performance increase of using GPUs is not universal. While accelerators outperform CPUs with a large margin on tasks that can be clearly separated and distributed, such as LAPACK/BLAS operations, applications that do not have such clear abstractions and solely rely on the CPU do not cope well with accelerators. Careful tuning and deeper knowledge of the applications is required to utilize the accelerator appropriately. It might be worth to use smaller accelerators to swap out, for example, BLAS operations.

Bibliography

[acc14] Acceleration and parallelization. http://www.gromacs.org/Documentation/Acceleration_and_parallelization, 2014. [Online; accessed 17-September-2014].

[arc] Understanding and managing hardware affinities on hierarchical platforms With Hardware Locality (hwloc). http://www.open-mpi.org/projects/hwloc/tutorials/20140522-PATC-hwloc-tutorial.pdf. [Online; accessed 17-September-2014].

[Cas14] Ralph Castain. Open MPI User's Mailing List Archives - [OMPI users] pinning processes by default. http://www.open-mpi.org/community/lists/users/2014/05/24459.php, 2014. [Online; accessed 17-September-2014].

[DMRF11] Ph.D. Dr. Mark R. Fernandez. Nodes, Sockets, Cores and FLOPS, Oh, My. http://en.community.dell.com/techcenter/high-performance-computing/w/wiki/2329, 2011. [Online; accessed 17-September-2014].

[Don13a] Wei Dong. MKL vs. ATLAS vs. OpenBLAS. http://www.wdong.org/wordpress/blog/2013/08/30/mkl-vs-atlas-vs-openblas/, 2013. Accessed: 2014-09-10.

[Don13b] Wei Dong. OpenBLAS benchmark. http://www.wdong.org/wordpress/blog/2013/09/01/openblas%E9%80%86%E8%A2%AD/, 2013. Accessed: 2014-09-10.

[gcc14a] GCC Mirros. https://gcc.gnu.org/mirrors.html, 2014. [Online; accessed 1-Oktober-2014].

[gcc14b] Gnu Compiler Collection Homepage. https://gcc.gnu.org/, 2014. [Online; accessed 1-Oktober-2014].

[Gia14] Paolo Giannozzi. Parallelization levels. http://www.quantum-espresso.org/wp-content/uploads/Doc/user_guide/node18.html, 2014. Accessed: 2014-09-22.

[gsl14] GNU Scientific Library. http://www.gnu.org/software/gsl/, 2014. [Online; accessed 1-Oktober-2014].

[hwl] hwloc(7) - Linux man page. http://linux.die.net/man/7/hwloc. [Online; accessed 17-September-2014].

[hwl14] Portable Hardware Locality (hwloc). http://www.open-mpi.org/projects/hwloc/, 2014. [Online; accessed 17-September-2014].

[icc14] Intel C++ Compiler. https://software.intel.com/en-us/c-compilers, 2014. [Online; accessed 1-Oktober-2014].

[Int] Performance Tools for Software Developers - Building Open MPI* with the Intel® compilers. https://software.intel.com/en-us/articles/performance-tools-for-software-developers-building-open-mpi-with-the-int [Online; accessed 10-September-2014].

[int13] Intel® MPI Benchmarks 4.0. https://software.intel.com/en-us/articles/intel-mpi-benchmarks, 2013. [Online; accessed 17-September-2014].

[int14a] Intel Developer Zone. https://software.intel.com, 2014. [Online; accessed 1-Oktober-2014].

[int14b] Intel Registration Center. https://registrationcenter.intel.com/RegCenter/, 2014. [Online; accessed 1-Oktober-2014].

[Joh13] Blake Johnson. OpenBLAS comparison to MKL. https://github.com/JuliaLang/julia/issues/3965#issuecomment-22745557, 2013. Accessed: 2014-09-10.

[Lar13] Peter Larsson. Quantum espresso vs. vasp (round 2). https://www.nsc.liu.se/~pla/blog/2013/02/19/qevasp-part2/, 2013. Accessed: 2014-09-24.

[liba] Open MPI Development Mailing List Archives - [OMPI devel] hwloc: libpci vs. libpciaccess - what should OMPI do? http://www.open-mpi.org/community/lists/devel/2013/03/12206.php. [Online; accessed 17-September-2014].

[libb] Paket: libibmad-dev (1.3.11-3 und andere) . https://packages.debian.org/de/sid/libibmad-dev. [Online; accessed 17-September-2014].

[libc] Paket: libibumad-dev (1.2.3-20090314-1) [universe] . http://packages.ubuntu.com/de/lucid/libibumad-dev. [Online; accessed 17-September-2014].

[MPIa] MPICH ABI Compatibility Initiative. http://www.mpich.org/abi/. [Online; accessed 05-September-2014].

[mpib] mpirun(1) man page (version 1.8.1). https://www.open-mpi.org/doc/v1.8/man1/mpirun.1.php. [Online; accessed 17-September-2014].

[MV2a] MPICH Installer's Guide. http://www.mpich.org/static/downloads/3.0.4/mpich-3.0.4-installguide.pdf. [Online; accessed 17-September-2014].

[MV2b] MVAPICH2 2.0 Quick Start Guide. http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.0-quickstart.pdf. [Online; accessed 17-September-2014].

[MV2c] MVAPICH2 Changelog. http://mvapich.cse.ohio-state.edu/static/media/mvapich/MV2_CHANGELOG-2.0.txt. [Online; accessed 18-September-2014].

[MV2d] Using the Hydra Process Manager. https://wiki.mpich.org/mpich/index.php/Using_the_Hydra_Process_Manager. [Online; accessed 17-September-2014].

[MV213] MVAPICH2. https://wiki.calculquebec.ca/w/MVAPICH2/en, 2013. [Online; accessed 17-September-2014].

[oCSU14] MVAPICH Team Network-Based Computing Laboratory Department of Computer Science and Engineering The Ohio State University. MVAPICH2 2.0 User Guide. http://mvapich.cse.ohio-state.edu/static/media/mvapich/mvapich2-2.0-userguide.html#x1-90004.3.1, 2014. [Online; accessed 17-September-2014].

[O'G13] Christopher O'Grady. Quantum espresso. https://confluence.slac.stanford.edu/display/SUNCAT/Quantum+Espresso, 2013. Accessed: 2014-09-22.

[ope] FAQ: Building Open MPI. http://www.open-mpi.org/faq/?category=building. [Online; accessed 17-September-2014].

[ope14] FAQ: General run-time tuning. http://www.open-mpi.org/faq/?category=tuning, 2014. [Online; accessed 17-September-2014].

[pci] Conventional PCI. http://en.wikipedia.org/wiki/Conventional_PCI. [Online; accessed 17-September-2014].

[Rei09] James Reinders. Mixing MPI and OpenMP, hugging hardware and dealing with it. https://software.intel.com/en-us/blogs/2009/11/16/mixing-mpi-and-openmp-hugging-hardware-and-dealing-with-it, 2009. [Online; accessed 17-September-2014].

[Squ12] Jeff Squyres. Open MPI v1.5 processor affinity options. http://blogs.cisco.com/performance/open-mpi-v1-5-processor-affinity-options/, 2012. [Online; accessed 17-September-2014].

[Squ13a] Jeff Squyres. Open MPI: Binding to core by default . http://blogs.cisco.com/performance/open-mpi-binding-to-core-by-default/, 2013. [Online; accessed 17-September-2014].

[Squ13b] Jeff Squyres. Process and memory affinity: why do you care? . http://blogs.cisco.com/performance/process-and-memory-affinity-why-do-you-care/, 2013. [Online; accessed 17-September-2014].

List of Figures

List of Tables

Listings