

# **Our new HPC-Cluster**

## **An overview**

Christian Hagen

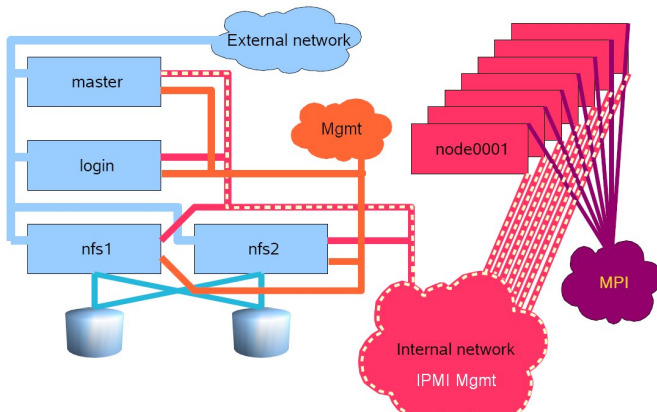
Universität Regensburg

Regensburg, 15.05.2009

# Outline

- 1 **Layout**
- 2 **Hardware**
- 3 **Software**
- 4 **Getting an account**
- 5 **Compiling**
- 6 **Queueing system**
- 7 **Parallelization**

# Layout



## Hardware overview

- **Compute Nodes:** 187  
per node:
  - 2 × Quad-Core AMD Opteron 2345 (Barcelona) 2.2GHz
  - most nodes have 16GB + some with 32GB
  - 250GB local disc space
- **Communication:** 176 nodes with Infiniband, others with GE
- **Storage:** 2 file servers (NFS) in fail-over mode,  
64TB storage capacity
- **Performance**(Infiniband nodes only): 12.3 TFlops peak,  
8.95 TFlops with LIN-  
PACK

# Storage

## Current partitioning of storage:

<code>/home</code>	1 TB	quota (2GB, 10k files), incr. backup
<code>/data</code>	8 TB	no quota, no backup, long term storage
<code>/scratch1</code>	8 TB	no quota, no backup, short term storage
<code>/scratch2</code>	8 TB	no quota, no backup, short term storage

- + special project partitions (10 TB)
- + another 29 TB in reserve

“long term” = some months – 1 year

“short term” = 1 run – 1 week

In addition, ordinary **Linux-HOME** is available on **login node** in  
`/home/dau12345`

# Software

- Compilers:
  - Intel Compiler v10.0 and v11.0
  - GNU Compiler v4.1.2
  - GNU Compiler v4.3.3 (soon)
- MPI implementations:
  - Intel MPI
  - OpenMPI
  - MVAPICH2
  - Parastation
- Math. libraries:
  - Math Kernel Library (MKL)
  - atlas
  - GNU Scientific Library (GSL)
  - PETSc

## Getting an account

First, fill out the form which you can find on the page of the Rechenzentrum. Required information are:

- Titel of project
- Description of the project
- Memory requirements and storage requirements (long and short term)
- Average runtime of jobs
- Maximum number of cores used simultaneously
- Job types: parallel and/or serial
- Programming languages: Fortran, C, C++, Java
- Compiler: GNU, Intel
- Libraries: MKL, atlas, blas
- PETSc

After submitting, printing, signing and returning it to the Rechenzentrum, you get an account.

Login: `> ssh dau12345@athene1`

# Modules

The compilers and MPI-libraries are packed in different **modules**.

```
module avail           lists all available modules
module load impi-gcc  loads a module (here, impi-gcc)
module list           lists loaded modules
module unload impi-gcc unloads the module impi-gcc
```

Available modules:

impi-gcc	mvapich2-gcc	openmpi-gcc	parastation-gcc
impi-intel	mvapich2-intel	openmpi-intel	parastation-intel
intel-10.0-icc	intel-10.0-ifort	intel-11.0	

## Math. libraries

- **Math Kernel Library (MKL):**

To set the environment variables:

```
source
```

```
/opt/intel/mkl/10.1.1.019/tools/environment/mklvarsem64t.sh
```

Then add the following libraries for linking.

For GNU compiler:

```
-lmkl_gf_lp64 -lmkl_gnu_thread -lmkl_core -lpthread -lm
```

For Intel compiler:

```
-lmkl_intel_lp64 -lmkl_sequential -lmkl_core -lpthread
```

- **ATLAS:**

The ATLAS library is located in:

```
/opt/lapack/atlas-3.8.2/lib
```

# Queueing system

Torque queueing system with Maui scheduler in **Fair-Share mode**

List of queues:

- **common**
  - Max. # of nodes = 32
  - Max. walltime = 48h
  - Max. # number of jobs per user = 10 running + 50 in queue
- **common32** (for 32GB nodes)
  - Max. # of nodes = 16
  - Max. walltime = 48h
  - Max. # number of jobs per user = 10 in total
- serial
- parallel
- special project queues

## Submit jobs

- Interactive jobs:

- `athene1> qsub -I -l nodes=4:ppn=8  
                  -l walltime=12:00:00 -l mem=12gb`
- `node001> mpiexec -n 32 ./my_mpi_job`
- `node001> exit`

- Batch jobs:

- `qsub jobscript`
- `qstat [[-f] job_id] [-u username]`
- `qdel [job_id]`
- `qalter ...[-l resource_list]... job_id`

→ more info in the man pages

## Queueing system

### Example script:

```
#!/bin/bash
#PBS -o myjob.out          specify file for stdout
#PBS -j oe                join stdout and stderr
#PBS -S /bin/bash         specify shell
#PBS -l nodes=4:ppn=8     specify # of nodes and processors
#PBS -l walltime=12:00:00 specify runtime
#PBS -l mem=12gb          specify required memory
cd ${PBS_O_WORKDIR}      change to submit directory
mpiexec -n 32 ./my_mpi_job
```

# pbstop

Place a file `.pbstoprc` in your home directory containing:

```
host=athene2
```

```
columns=8
```

```
maxnodegrid=8
```

```
show_cpu=0,1,2,3,4,5,6,7
```

```
Sitzung Bearbeiten Ansicht Lesezeichen Einstellungen Hilfe
Usage Totals: 1131/1496 Procs, 143/187 Nodes, 57/98 Jobs Running      12:18:34
Node States:   1 down      44 free
              140 job-exclusive  2 offline

visible CPUs: 0,1,2,3,4,5,6,7
                1      2      3      4      5      6      7      8
-----
node0001 Wttttt. AaaAJUU kxxssss eeee ccc UVVV HHHHYY AAkkkkk IIII
node0009  kkkkkk  NOOllll  zzzzrrrr  cccqppp  nnnn  sssq  TTTT  sss555
node0017  kkkkkk  NOOllll  zzzzrrrr  cccqppp  nnnn  sssq  TTTT  sss555
node0025  kkkkkk  NOOllll  zzzzrrrr  cccqppp  nnnn  sssq  TTTT  sss555
node0033  kkkkkk  NOOllll  zzzzrrrr  cccqppp  nnnn  sssq  TTTT  sss555
node0041  kkkkkk  NOOllll  zzzzrrrr  cccqppp  nnnn  sssq  TTTT  sss555
node0049  kkkkkk  NOOllll  zzzzrrrr  cccqppp  nnnn  sssq  TTTT  sss555
node0057  kkkkkk  NOOllll  zzzzrrrr  cccqppp  nnnn  sssq  TTTT  sss555
node0065  kkkkkk  NOOllll  zzzzrrrr  cccqppp  nnnn  sssq  TTTT  sss555
node0073  kkkkkk  NOOllll  zzzzrrrr  cccqppp  nnnn  sssq  TTTT  sss555
node0081  kkkkkk  NOOllll  zzzzrrrr  cccqppp  nnnn  sssq  TTTT  sss555
node0089  kkkkkk  NOOllll  zzzzrrrr  cccqppp  nnnn  sssq  TTTT  sss555
node0097  kkkkkk  NOOllll  zzzzrrrr  cccqppp  nnnn  sssq  TTTT  sss555
node1005  kkkkkk  NOOllll  zzzzrrrr  cccqppp  nnnn  sssq  TTTT  sss555
node1013  kkkkkk  NOOllll  zzzzrrrr  cccqppp  nnnn  sssq  TTTT  sss555
node1021  kkkkkk  NOOllll  zzzzrrrr  cccqppp  nnnn  sssq  TTTT  sss555
node1029  kkkkkk  NOOllll  zzzzrrrr  cccqppp  nnnn  sssq  TTTT  sss555
node1037  kkkkkk  NOOllll  zzzzrrrr  cccqppp  nnnn  sssq  TTTT  sss555
node1045  kkkkkk  NOOllll  zzzzrrrr  cccqppp  nnnn  sssq  TTTT  sss555
node1053  kkkkkk  NOOllll  zzzzrrrr  cccqppp  nnnn  sssq  TTTT  sss555
node1061  kkkkkk  NOOllll  zzzzrrrr  cccqppp  nnnn  sssq  TTTT  sss555
node1069  hhhhhhhh  nnnnnnnn  oooooooo  rrrrrrrr  tttttttt  uuuuuuuu  vvvvvvvv  wwwwwww
node0769  kkkkkk  NOOllll  zzzzrrrr  cccqppp  nnnn  sssq  TTTT  sss555
node0777  kkkkkk  NOOllll  zzzzrrrr  cccqppp  nnnn  sssq  TTTT  sss555
-----
Job#  Username  Queue  Jobname  Nodes  5  Elapsed/Requested
176281 krc54214  batch  jobsend_2  1  0  --/ 48:00
176282 krc54214  batch  jobsend_2  2  1  0  --/ 48:00
176283 krc54214  batch  jobsend_2  2  1  0  --/ 48:00
176284 krc54214  batch  jobsend_2  2  1  0  --/ 48:00
176285 krc54214  batch  jobsend_2  2  1  0  --/ 48:00
176286 krc54214  batch  jobsend_2  2  1  0  --/ 48:00
176287 krc54214  batch  jobsend_2  2  1  0  --/ 48:00
176288 krc54214  batch  jobsend_2  2  1  0  --/ 48:00
176289 krc54214  batch  jobsend_2  2  1  0  --/ 48:00
176290 krc54214  batch  jobsend_2  2  1  0  --/ 48:00
176291 krc54214  batch  jobsend_2  2  1  0  --/ 48:00
176292 krc54214  batch  jobsend_2  3  1  0  --/ 48:00
176293 krc54214  batch  jobsend_2  3  1  0  --/ 48:00
176294 krc54214  batch  jobsend_2  3  1  0  --/ 48:00
176295 krc54214  batch  jobsend_2  3  1  0  --/ 48:00
176296 krc54214  batch  jobsend_2  3  1  0  --/ 48:00
176297 krc54214  batch  jobsend_2  3  1  0  --/ 48:00
176298 krc54214  batch  jobsend_2  3  1  0  --/ 48:00
176299 krc54214  batch  jobsend_2  3  1  0  --/ 48:00
176300 krc54214  batch  jobsend_2  3  1  0  --/ 48:00
176301 krc54214  batch  jobsend_2  3  1  0  --/ 48:00
176302 krc54214  batch  jobsend_2  4  1  0  --/ 48:00
```

# MPI - Message Passing Interface

## Characteristics:

- Standardized distributed memory parallelism with message passing
- Process-based
- Each process has its own set of local variables
- Data is copied between local memories with messages that are sent and received via explicit calls to MPI routines (MPI\_Init, MPI\_Send, MPI\_Recv, MPI\_Gather, MPI\_Scatter, MPI\_Reduce, ..., MPI\_Finalize)
- Detailed information for all routines together with small example programs can be found in the MPI Standard (Online on [www.mpi-forum.org](http://www.mpi-forum.org) or in physics library **84/ST 230 M583**)

## MPI - Pros and Cons

### Pros:

- Runs on both shared and distributed memory architectures
- Portable. Exists on almost all systems
- Very flexible. Thus can be used to a wide range of problems
- Very scalable programs (depending on the algorithm)
- Explicit parallelism → better performance
- Communication and computation can in principle overlap

### Cons:

- Requires more programming changes to go from serial to parallel version
- Can be hard to debug
- Complicates profiling and optimization
- Performance is limited by the communication between the processes

# Hello World - MPI

## hello\_world\_mpi.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
int main (int argc, char **argv)
{
    int rank, size;
    MPI_Init (&argc, &argv);
    MPI_Comm_rank (MPI_COMM_WORLD, &rank);
    MPI_Comm_size (MPI_COMM_WORLD, &size);
    printf ( "Hello world from process %d of %d\n." , rank, size);
    MPI_Finalize ();
    return EXIT_SUCCESS
}
```

*/\* starts MPI \*/*  
*/\* get current process id \*/*  
*/\* get number of processes \*/*

## Compiling and running the code:

```
> mpicc hello.world.mpi.c
> mpiexec -n 2 ./a.out
Hello world from process 0 of 2.
Hello world from process 1 of 2.
```

# OpenMP

## Characteristics:

- Standardized shared memory parallelism
- Thread-based
- Each thread sees same global memory and a set of local variables
- Mainly loop-parallelism via OpenMP-directives (e.g., OMP PARALLEL DO)
- Compiler translates OpenMP directives into thread-handling (Intel: -openmp, GNU: -fopenmp)
- The latest official OpenMP specifications can be found on **[www.openmp.org](http://www.openmp.org)**. Also contains a number of examples.

# OpenMP - Pros and Cons

## Pros:

- Easier to program and to debug than MPI
- Directives can be added incrementally → gradual parallelization
- Can still run the program as a “serial code“
- Serial code doesn't have to be changed significantly
- Code easier to understand → easier to maintain
- Becomes more and more important (Many-Core processors)
- Implicit messaging

## Cons:

- Can only be run on shared memory architectures
- Requires compiler that supports OpenMP
- Mostly used for loop parallelization
- Overhead is an issue if loop is too small
- Threads are executed in a nondeterministic order
- Explicit synchronization is required

# Hello world - OpenMP

## hello\_world\_openmp.c:

```
#include <omp.h>
#include <stdio.h>
int main (int argc, char *argv[])
{
    int th_id, nthreads;
    #pragma omp parallel private(th_id)
    {
        th_id = omp_get_thread_num();
        printf("Hello World from thread %d\n.", th_id);
        #pragma omp barrier

        if ( th_id == 0 ) {
            nthreads = omp_get_num_threads();
            printf("There are %d threads\n.",nthreads);
        }
    }
    return 0;
}
```

*/\* start a threaded block, each thread gets its own th\_id\*/*

*/\* get thread number \*/*

*/\* wait until all threads finished printing \*/*

*/\* only thread 0 \*/*

*/\* get number of threads \*/*

*/\* and print it \*/*

## Compiling and running the code:

```
> icc -openmp hello_world_openmp.c
> ./a.out
Hello World from thread 1.
Hello World from thread 0.
There are 2 threads.
```

## Hybrid MPI + OpenMP

### Why hybrid:

- Most modern HPC systems are **clusters of SMP nodes**.
- Elegant in concept and architecture: Using **MPI on node interconnect** and **OpenMP within each SMP node**. Good usage of shared memory system resources (memory, latency, bandwidth).
- **Avoids the extra communication overhead** with MPI within node.
- OpenMP allows **increased** and/or **dynamic load balancing**.
- **Reduction of replicated data** on MPI level  
( $Volume/node > Volume/core$ )
- Significantly **reduces the number of MPI processes**
- OpenMP threading **makes each process “faster”**, even when code is already Amdahl-limited