



MuCoSim Introduction

Thomas Gruber and Katrin Nusser (HPC @ Uni Erlangen)

Thomas.Gruber@fau.de



BEFORE WE START - SSH-SHELL

- <https://tinyurl.com/yau5vh2g>
- Download „Portable edition“
- Save somewhere (e.g. Desktop)
- No installation required
- **Windows only!**

Info in the web

- General info on HPC workflow and tools: HPC Wiki ⓘ

[https://hpc-wiki.info/hpc/HPC Wiki](https://hpc-wiki.info/hpc/HPC%20Wiki)

- NHR docs: ⓘ

<https://hpc.fau.de/systems-services/systems-documentation-instructions/>

On later slides these icons contain links to more content

MuCoSim Introduction

- Introduction to NHR@FAU cluster systems
- How to log into NHR@FAU cluster systems?
- How do I get a job?

NHR@FAU “Woody” cluster + TinyEth

main workhorse for *throughput*

■ **Woody:** [↗](#)

- all nodes with 4 cores and high clock frequency (3.4/3.5 GHz) Intel Xeon E3-12xx v? processors
- at least 400 GB local HDD
- and Gbit only
 - 40x Intel Sandybridge, 8 GB
 - 72x Intel Haswell, 8 GB
 - 64x Intel Skylake, 32 GB
 - 112 Kabylake, 32 GB



■ **TinyEth:** [↗](#)

- 20 nodes (480 cores)
 - › 12 cores @ 2.66 GHz
 - › 48 GB
 - › 30/190/420 GB local HDD
- **Single cores can be requested**
- **Retires end of 2021**

NHR@FAU “Emmy” cluster

main *parallel* workhorse



- **543** compute **nodes** (10.880 cores)
 - **2** Intel Xeon E5-2660v2 (**Ivy Bridge**)
2.2 GHz (**10 cores**)
 - **20 cores/node** + SMT cores
- **64 GB** main memory
- No local disks
- 16 accelerator nodes same CPUs
 - 6 nodes with 2 x NVIDIA K20 GPGPUs
 - 4 nodes with 1x K20
 - 4 nodes with 1x NVIDIA V100

- Full **QDR Infiniband** fat tree
 - 40 GBit/s and < 2 μ s latency
- Power consumption: ~160 KW
(backdoor heat exchanger)
- **#210@Top500**
(Nov 2013)

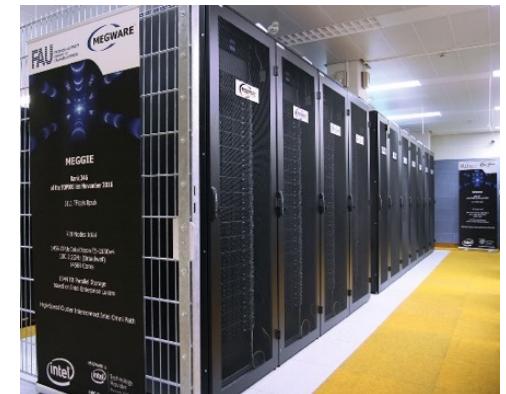
$R_{\max} = 191 \text{ TF/s}$



NHR@FAU “Meggie” cluster for scalable parallel jobs



- **728 Compute nodes (14.560 cores)**
 - 2 Intel Xeon E5-2630 v4 (Broadwell) 2.2 GHz (10 cores)
 - 20 cores/node **No SMT**
 - 64 GB main memory
- No local disks
- Peak Performance:
 $R_{\text{peak}} = 0.5 \text{ PF/s}$
- #346@Top500 (Nov. 2016)
 $R_{\text{max}} = 0.48 \text{ PF/s}$
- Intel OmniPath network:
Up to 100 Gbit/s
- Price: € 2,5 Mio.
- Power:
120-200 kW
(depending on workload)



NHR@FAU “Fritz” cluster

for scalable parallel jobs



- **944 Compute nodes (33.984 cores)**
 - 2x Intel Xeon Platinum 8360Y (Ice Lake) 2.4 GHz (**18 cores**)
 - **36 cores/node** **No SMT**
 - **256 GB** main memory
- No local disks
- Peak Performance:
 $R_{\text{peak}} = \text{XXXX PF/s}$
- Blocking **HDR100** InfiniBand:
Up to 100 Gbit/s
Islands with 64 nodes
- Power:
Unknown

Not open for general usage yet!

NHR@FAU “Alex” cluster

for GPGPU jobs (ML, DataScience, ...)



- **>50 Compute nodes**
 - 2x AMD EPYC 7713 (Milan)
2.4 GHz (64 cores)
 - 128 cores/node **No SMT**
 - **>= 512 GB** main memory
- 8 GPUs per node
 - Nvidia A100
 - Nvidia A40
- Local NVMe SSDs

- Peak Performance:
 $R_{\text{peak}} = \text{XXXX PF/s}$

Network:

- 25 GbE
- **HDR200** InfiniBand (some)

- Power:
Unknown

Not open for general usage yet!

NHR@FAU “TinyGPU” cluster

most nodes paid by individual user groups



- **7 nodes** with 2x “Nehalem” @2.66 GHz, 24 GB, HDD, **2x GTX980**
- **7 nodes** with 2x “Broadwell” @2.2 GHz, 64 GB, 980 GB SSD, **4x GTX1080**
- **10 nodes** with 2x “Broadwell” @2.2 GHz, 64 GB, 980 GB SSD, **4x GTX1080Ti**
- **4 nodes** with 2x “Skylake” @ 3.2 GHz, 96 GB, 2.9 TB SSD, **4x Tesla V100**
- **12 nodes** with 2x “Skylake” @ 3.2 GHz, 96 GB, 1.8 TB SSD, **4x RTX2080Ti**
- **8 nodes** with 2x “Cascadelake” @ 2.9 GHz, 384 GB, 3.8 TB SSD, **8x RTX3080**
- **8 nodes** with AMD Rome @ 2.0 GHz, 512 GB, 5.8 NVMe SSD, **4x A100**



NHR@FAU „Test“ cluster Playground with different systems


Accessible only after request!



Name	Description
phinally	2x „SandyBridge“ @2.7GHz (8 cores)
ivyep1	2x „IvyBridge“ @3.0GHz (10 cores)
hasep1	2x „Haswell“ @2.3 GHz (14 cores)
broadep2	2x „Broadwell“ @2.3GHz (18 cores)
skylakesp2	2x „Skylake“ @2.4GHz (20 cores)
caslakesp2	2x „Cascade Lake“ @2.5GHz (20 cores)
aurora1	2x (small) „Skylake“ and 2x NEC TSUBASA
medusa	2x „Cascade Lake“ and 4x different NVIDIA GPUs
optane1	2x „Cascade Lake“ and Intel Optane persistent memory
icx32	2x „Icelake“ @2.6GHz (32 cores)
icx36	2x „Icelake“ @2.4GHz (36 cores)
naples	2x „AMD Zen“ (Naples)
rome1	1x „AMD Zen2“ (Rome) (128 cores)
rome2	2x „AMD Zen2“ (Rome) and AMD MI100 GPU
milan1	2x „AMD Zen3“ (Milan) (128 cores)
warmup	2x „Thunder X2“ @2.2GHz (32 cores, ARMv8)

Cluster access



- Primary point of contact: cluster front-ends
 - `[woody | emmy | meggie | fritz | alex] .rrze.uni-erlangen.de`
 - `testfront.rrze.uni-erlangen.de`
 - ...
 - For TinyGPU, TinyFat and TinyEth use **woody** frontend
 - Only available from within FAU (private IP addresses)
- Access from outside FAU: dialog server 
 - **`cshpc.rrze.uni-erlangen.de`**
 - The only machine with a public IP address

Secure Shell

MobaXTerm:

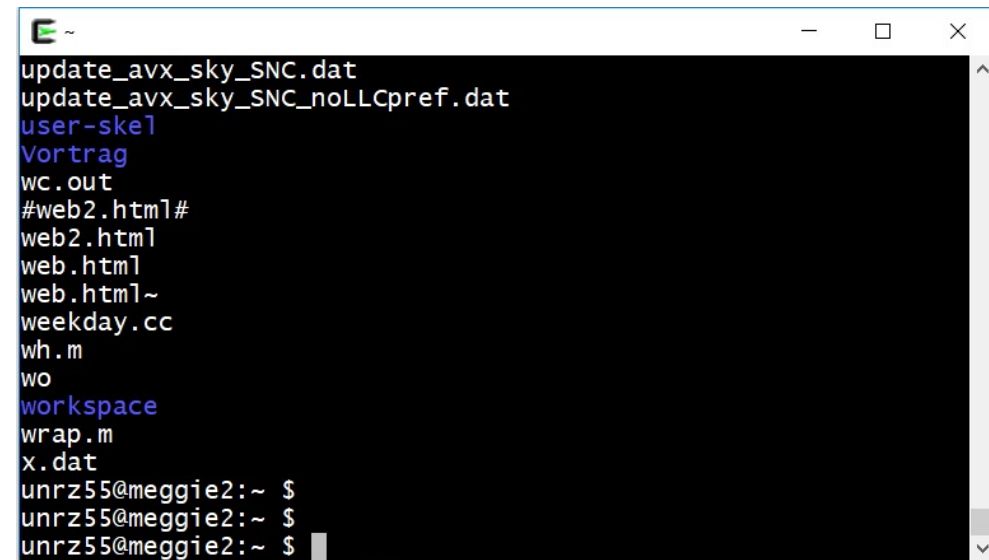
- New session
- Remote host `csahpc.rrze.uni-erlangen.de` (or `meggie.rrze.uni-erlangen.de`)
- Specify HPC user name `<hpcuser>`

- By default: text mode only

```
$ ssh <hpcuser>@meggie.rrze.uni-erlangen.de
```

- Basic knowledge of file handling, scripting, editing, etc. under Linux is required
- X11 forwarding with option `-X/-Y`
 - Requires local X server

Included in
MobaXterm



```
E ~  
update_avx_sky_SNC.dat  
update_avx_sky_SNC_noLLCpref.dat  
user-skel  
Vortrag  
wc.out  
#web2.html#  
web2.html  
web.html  
web.html~  
weekday.cc  
wh.m  
wo  
workspace  
wrap.m  
x.dat  
unrz55@meggie2:~ $  
unrz55@meggie2:~ $  
unrz55@meggie2:~ $
```

Log into frontend **meggie**.rrze.uni-erlangen.de

- Use your <hpcuser> account!
- Copy folder `~unrz139/mucosim` to your home
`cp -r ~unrz139/mucosim $HOME`

From external: Connect to **cshpc** first!

For **PTfS** Students: use **fritz**.rrze.uni-erlangen.de instead

Interactive runs on the front-ends

- The cluster front-ends are for interactive work
 - Editing, compiling, preparing input,...
 - Amount of compute time per binary is limited by system limits
 - E.g., after 1 hour of CPU time your run will be killed
 - **MPI jobs are not allowed on front ends at RRZE**
- Front-ends are shared among all users, so **be considerate!**

```
iww042@meggie1$ emacs Makefile
iww042@meggie1$ make all
iww042@meggie1$ ./scripts/preprocess.py < inputfile
iww042@meggie1$ ./bin/a.out arg1 arg2 arg3
```

Batch jobs



- All clusters have **resource manager software**
 - “**Batch system**”
 - Users can request resources for their jobs
 - Number of nodes (optionally: type of nodes, memory, ...)
 - Job runtime
 - What to run (normally a shell script)
 - Popular batch systems: PBS Pro, **Torque**ⁱ, **SLURM**ⁱ, LSF, GridEngine
 - Some setups (e.g., at RRZE) allow **interactive batch jobs**
- What you do with your node allocation is entirely up to you
- **Most queues** at RRZE have a **24 hour wall time limit**



Example: Simple SLURM batch script (Meggie, ...)

- Most simple batch script (`job1.sh`):

```
#!/bin/bash -l
~/bin/a.out arg1 arg2 arg3
```

- Submission:

```
unrz139@meggie1$ sbatch --nodes=1 --ntasks=1 \  
--cpus-per-task=20 --time 01:00:00 \  
job1.sh  
Submitted batch job 966578
```

Resource specification

Job Identifier

sbatch: queue job and run it “in the background”
srun: run job directly and wait for results

Example: SLURM batch script (Meggie, Testcl.)

```
#!/bin/bash -l
#SBATCH --nodes=4 --ntasks-per-node=20 --time=06:00:00
#SBATCH --job-name=Sparsejob_33
#SBATCH --export=NONE
unset SLURM_EXPORT_ENV                # avoid evil login shell settings

# jobs always start in $HOME: change to a temporary job dir on $WOODYHOME
mkdir ${WOODYHOME}/${SLURM_JOB_ID}
cd ${WOODYHOME}/${SLURM_JOB_ID}
# copy input file from location where job was submitted, and run
cp ${SLURM_SUBMIT_DIR}/inputfile .
srun --mpi=pmi2 --ntasks=80 --ntasks-per-node=20 ${HOME}/bin/a.out \
-i inputfile -o outputfile

# save output
mkdir -p ${WOODYHOME}/output/${SLURM_JOB_ID}
cp outputfile ${WOODYHOME}/output/${SLURM_JOB_ID}
cd
# get rid of the temporary job dir
rm -rf ${WOODYHOME}/${SLURM_JOB_ID}
```

Job option
sentinel

Job submission options:
Nodes, cores p/ node,...

`$SLURM_*` variables
contain job-relevant
data

For MPI jobs!

Actual run of
your binary

SLURM batch job submission (Meggie, Testcluster)

```
iww042@meggie1$ sbatch job3.sh
Submitted batch job 357074
iww042@meggie1:~ $ squeue -l
Mon Jan 28 17:38:52 2019
      JOBID PARTITION     NAME     USER      STATE      TIME  TIME_LIMI  NODES NODELIST(REASON)
      357074      work Sparsejo  iww042  RUNNING    0:35   1:00:00      4  m[0101-0104]
iww042@meggie1:~ $
```

- Request additional resources (like GPUs)
 - `--gres=gpu:X` for any **X** GPUs
 - `--gres=gpu:a100:X` for **X** Nvidia A100 GPUs
 - (maybe) you have to select a different partition: `--partition=a100`
- The amount of GPUs correspond to the amount of CPU hardware threads and available memory

Interactive batch job with SLURM (Meggie, Testcluster)

- `srun` command can queue/run a job directly

```
unrz139@meggie1$ salloc --nodes=2 --ntasks-per-node=20 \  
                    --time=01:00:00  
unrz139@m0101$ echo $SLURM_SUBMIT_DIR  
/home/hpc/iww0/iww042  
unrz139@m0101$
```

- X11 forwarding currently not supported
 - Workaround
 - submit job with `sleep` command in batch script
 - use `ssh -X` to log in to node

SLURM user commands (non-exhaustive)

Command	Purpose	Options
<code>sbatch [<options>]</code> <code><script></code> <code>salloc [<options>]</code> <code><script></code>	Submit batch job	<code>--time=HH:MM:SS</code> <code>--nodes=#</code> <code>--ntasks-per-node=#</code> <code>--error=<stderr_filename></code> <code>--output=<stdout_filename></code> <code>--mail-user=<address></code> <code>--mail-type=ALL BEGIN END ...</code>
<code>squeue [<options>]</code>	Check job status	<code>-j <JobID></code> show job <code>-t RUNNING</code> show only running jobs <code>-u <USER></code> this user only
<code>scancel <JobID></code>	Delete batch job	—
<code>srun <options></code>	Run program	Many options; see man page

Submit `jobs/job1.sh` to batch system on **meggie**
(more complex `jobs/job2.sh`, can be used as reference)

Start an interactive job on a single node with 2 hours time limit and
-C hwperf option

Which host are you on?

(You can use the interactive session for the following hands-on, so don't close it!)

Open a separate shell to **meggie** for compilation

PTfS Students: Similar on **fritz**

MuCoSim Introduction

Software Environment



Pre-installed software packages

- **Linux** standard distro **packages**
- “Full” installation available on cluster front-ends
 - Easy to add additional packages
- Node installation: usually stripped down
 - Not easy to add new software due to space constraints
 - On some nodes you cannot even compile your code (no headers)
 - Compile on front-ends

The module system



- Software provided locally by RRZE
 - Compilers, libraries, commercial and open software
 - Installed on central server and available on all cluster nodes

- Software has to be loaded in the user's environment to become usable
 - Command: **module**
 - All module commands **affect the current shell** only!

The `module` command

Show available modules: `module avail`

```
$ module avail
```

```
----- /apps/modules/data/applications -----  
amber-gpu/14p13-at15p06-gnu-intelmpi5.1-cuda7.5 gromacs/4.6.6-mkl-IVB  
amber-gpu/16p04-at16p10-gnu-intelmpi5.1-cuda7.5 gromacs/5.0.4-mkl-IVB  
amber/12p21-at12p38-intel16.0-intelmpi5.1      gromacs/5.1.1-mkl-IVB_d  
----- /apps/modules/data/development -----  
cuda/7.5                intel64/16.0up04                intelmpi/5.1up03-intel  
cuda/8.0                intel64/17.0up05 (default)     llvm-clang/3.8.1  
cuda/9.0                intel64/18.0up02                opencl/intel-cpuonly-5.2.0.10002  
cuda/9.1                intel64/18.0up03                openmpi/1.08.8-gcc  
$
```

The `module` command

Load a module: `module load <modulename>`

```
$ module load intel64
```

```
$ icc -V
```

```
Intel(R) C Intel(R) 64 Compiler for applications running on Intel(R) 64, Version 17.0.5.239 Build 20170817
```

```
Copyright (C) 1985-2017 Intel Corporation. All rights reserved.
```

At RRZE, the Intel compilers are named `intel64/<version>`
Without `<version>`, the default is loaded

Display loaded modules: `module list`

```
$ module list
```

```
Currently Loaded Modulefiles:
```

```
1) python/3.5-anaconda      2) intelmpi/2017up04-intel  3) mkl/2017up05           4) intel64/17.0up05
```

Fritz:

- Intel compilers are named `intel/<version>`
- Modules commonly do not load other modules (no `mkl` or `intelmpi` module when loading `intel`)

Module command summary

Command	What it does
<code>module avail</code>	List available modules
<code>module whatis</code>	Shows over-verbose listing of all modules
<code>module list</code>	Shows which modules are currently loaded
<code>module load <pkg></code>	Loads module <i>pkg</i> (<i>default</i>), i.e., adjusts environment
<code>module load <pkg>/<version></code>	Loads specific version of <i>pkg</i> instead of default
<code>module unload <pkg></code>	Undoes what the load command did
<code>module help <pkg></code>	Shows a detailed description of <i>pkg</i>
<code>module show <pkg></code>	Shows what environment variables <i>pkg</i> modifies and how

How many Intel MPI modules are available?

What is the latest installed GCC version?

Is it possible to load IntelMPI and OpenMPI at the same time? Why?

What's the `JAVA_ROOT` for java module?