

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_Wiki)

- 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 “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)





- **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:
 $R_{\text{peak}} = 2.233 \text{ PF/s}$ (with 612 nodes)
- **Blocking HDR100 InfiniBand:**
Up to 100 Gbit/s
Islands with 64 nodes
- Power:
 $\approx 1\text{MW}$

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:
 $R_{\text{peak}} = 2.938 \text{ PF/s}$ (with parts of the system)

Network:

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

- Power:
<200kW

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**





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)

- Primary point of contact: cluster front-ends
 - `[woody|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

- By default: text mode only

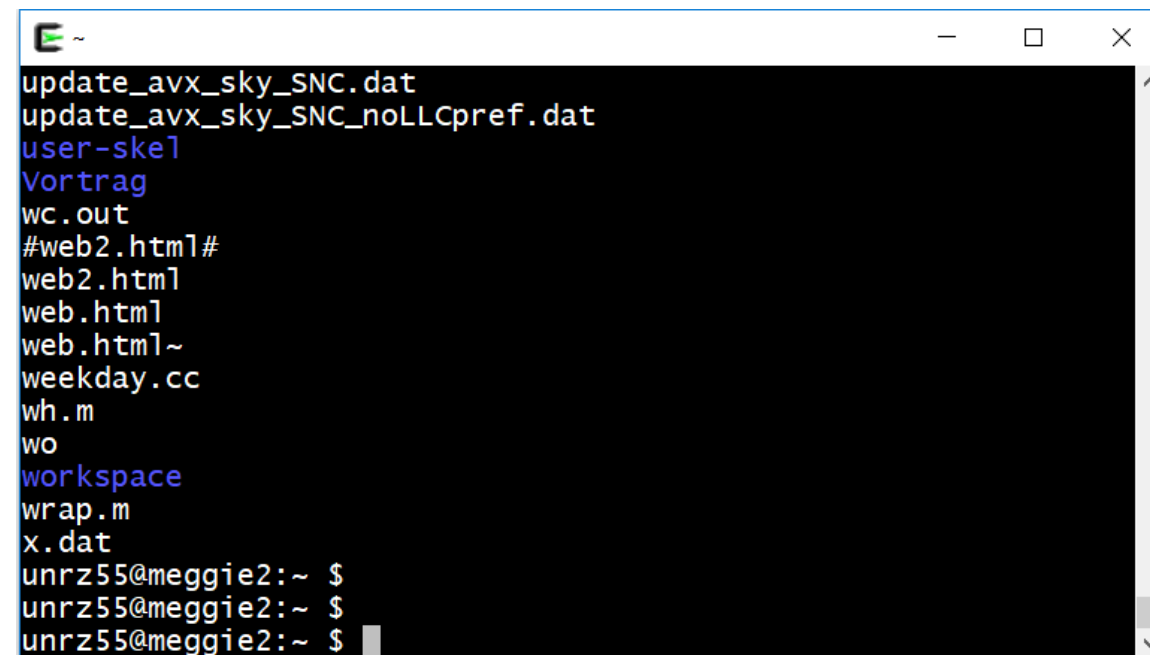
MobaXTerm:

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

```
$ 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



```
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:~ $
```

- There are no passwords anymore for accounts at NHR@FAU
- Your key is "at home"
- Connect to `cschpc` works with your key
- If you do "`ssh X`" there, no key is available
 - Use `ProxyJump` over `cschpc` from home (home key is used)
 - Generate a separate key at `cschpc` for NHR@FAU internal SSH

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 NHR@FAU**
- 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
```

- 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**, **SLURM**, 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`):

Always use
`bash -l`

```
#!/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”
`salloc`: queue job and run it “in the foreground”
`sruntime`: run job directly and wait for results

Example: SLURM batch script (Meggie, Testcl.)

```
#!/bin/bash -l
```

Job option
sentinel

```
#SBATCH --nodes=4 --ntasks-per-node=20 --time=06:00:00
```

```
#SBATCH --job-name=Sparsejob_33
```

```
#SBATCH --export=NONE
```

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

```
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 .
```

`$SLURM_*` variables
contain job-relevant
data

```
srun --mpi=pmi2 --ntasks=80 --ntasks-per-node=20 ${HOME}/bin/a.out \
```

```
-i inputfile -o outputfile
```

For MPI jobs!

```
# 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}
```

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
- Attach to a running job:

```
$ srun --pty --jobid YOUR-JOBID bash -l
```

SLURM pitfalls

- Setting CPU frequency on Fritz/Alex can only be done via **srun**:

```
$ salloc ...  
$ srun --cpu-freq=<min>-<max>:performance ...
```

- Access to hardware performance counters needs to be requested

```
$ salloc --constraint hwperf  
$ srun -C hwperf
```

SLURM user commands (non-exhaustive)

Command	Purpose	Options
<code>sbatch</code> [<code><options></code>] <code><script></code> <code>salloc</code> [<code><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</code> [<code><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</code> <code><JobID></code>	Delete batch job	–
<code>srun</code> <code><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



- Software provided locally by RRZE
 - Compilers, libraries, commercial and open software
 - Installed on central server and available on all cluster nodes
 - Module availability can be different each system

- 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/compiler -----
gcc/9.4.0  gcc/10.3.0  gcc/11.2.0  gcc/12.1.0  intel/2021.4.0  intel/2022.1.0
----- /apps/modules/data/development -----
intelmpi/2021.6.0  openmpi/4.1.2-gcc11.2.0
intelmpi/2021.7.0  openmpi/4.1.2-intel2021.4.0
$
```

The module command

Load a module: **module load <modulename>**

```
$ module load intel
$ icc --version
icc (ICC) 2021.6.0 20220226
Copyright (C) 1985-2022 Intel Corporation. All rights reserved.
```

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

Display loaded modules: **module list**

```
$ module list
Currently Loaded Modulefiles:
 1) intel/2022.1.0          3) intelmpi/2021.7.0
 2) python/3.9-anaconda   4) mkl/2022.1.0
```

Older systems:

- Intel compilers are named intel64/<version>
- Modules commonly load other modules (mkl or intelmpi module when loading intel64)

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 the default `java` module?

MuCoSim Introduction (Part II)

Thomas Gruber and Katrin Nusser (HPC @ Uni Erlangen)

Thomas.Grubert@fau.de

Preparation:

Copy `~unrz139/mucosim` to your home
(WARNING: may overwrite existing files!)

Let's start with some recap:

Go to `mucosim/stream` and compile the code with latest **Intel** suite

Run code with `OMP_NUM_THREADS=4` a few times and determine min. and max. bandwidth for the `copy` kernel.

MuCoSim Introduction

COMPILERS



- We provide [GCC](#), [Intel C/C++ Compiler](#) (and PGI and others)
- On warmup also arm-clang (/opt/arm/... license up-on-request)
- Provided through **module** system
- Common compiler names:
 - GCC: **gcc** and **gfortran**
 - Intel: **icc** and **ifort**

Library names

`libtest.so` → `-ltest`

Location(s) of headers

Location(s) of libs

```
gcc -O3 -fopenmp -I<INCDIR> -L<LIBDIR> test.c -o test -llib
gfortran -O3 -fopenmp -I<INCDIR> -L<LIBDIR> test.f90 -o test -llib
```

Compiler options

Input file(s)

Output file

Compilers @ RRZE

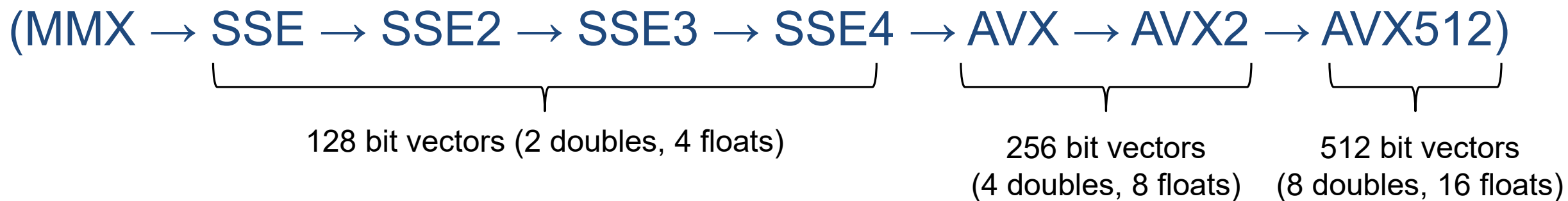
- Common compiler switches:

Meaning	GCC	Intel compilers
General optimization level	<code>-O1</code> , <code>-O2</code> , <code>-O3</code> (some vendor specific options like <code>-Ofast</code>)	
Hardware feature flags	<code>-m</code> like <code>-mavx2</code>	<code>-x</code> like <code>-xCORE-AVX2</code>
Compiler feature flag	<code>-f</code> like <code>-fopenmp</code>	<code>-q</code> like <code>-qopenmp</code>

- In many cases, the Intel compiler
 - produces „better“ code and often better performing
 - provides fallbacks for GCC flags (`-fopenmp` accepted by ICC)
- CUDA compilers only available at nodes with GPUs

Compile `compile/get_cpuflags.c` with GCC and run it

What is the widest SIMD the system supports?



Compile `compile/triad.c` with recent GCC and ICC and ...

What's the minimum runtime you can achieve on the compute node?

Single hardware thread? Do optimization flags help?

All hardware threads?

```
Remember: Copy folder to your home/workdir/...  
cp -r ~unrz139/mucosim $HOME
```