CHEOPS

Cologne High Efficient Operating Platform for Science

Brief Instructions

(Version: 07.10.2013)

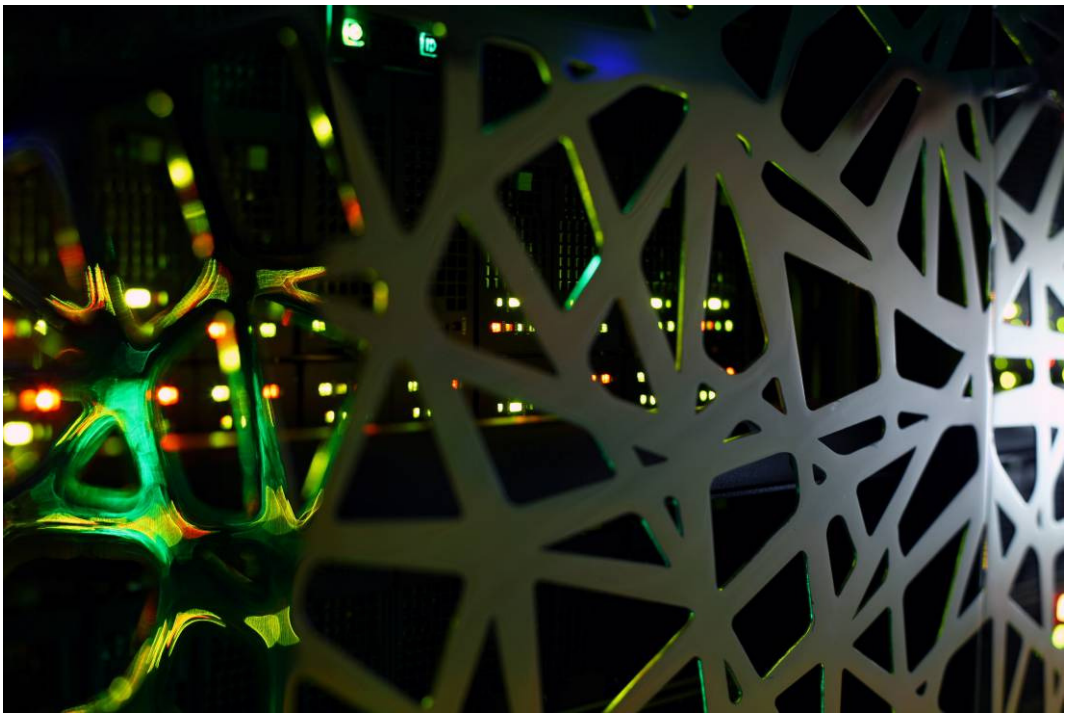

Foto: Thomas Josek/JosekDesign

Viktor Achter
Dr. Stefan Borowski
Lech Nieroda
Dr. Lars Packschies
Volker Winkelmann

HPC team: hpc-mgr@uni-koeln.de
Scientific support: wiss-anwendung@uni-koeln.de

Contents

1 Scope of this document 2
2 Contacts for support 2
3 Description of the system 2
3.1 Who may use this service? 3
3.2 How do I gain access? 3
3.2.1 How to get an Uni-account 3
3.2.2 How to get HPC authorization 4
3.3 Server addresses 4
3.4 Structure of the file system 4
3.5 What you should do on first login 5
3.6 How to archive data 5
4 Environment modules 6
4.1 Overview of important module commands 6
4.2 Usage notes 6
5 Batch system (SLURM) 7
5.1 Important differences between SLURM and TORQUE/Maui 8
5.2 Overview of important SLURM commands 9
5.3 Batch job scripts 9
5.4 Job arrays and interactive jobs 11
5.5 Resource requests 12
6 Development environment 13
6.1 Compilers 13
6.2 Intel MKL 13
6.3 NAG Numerical Library 14
6.3.1 NAG Fortran 77 Library 15
6.3.2 NAG C Library 15
6.3.3 NAG Fortran 90 Library 16
6.3.4 NAG Fortran 77 Library for SMP & multicore 16
6.3.5 NAG Fortran 77 Parallel Library (MPI version) 17
6.3.6 Examples & Documentation for the NAG Numerical Library 17
6.4 ARPACK 18
6.5 Intel MPI Library 18
6.6 Open MPI 19
6.7 MPI Runtime Environment 19
6.8 Intel Debugger 19
6.9 Allinea DDT Debugger 20
6.10 Intel Inspector XE 20
6.11 PAPI Library 20
6.12 Intel VTune Amplifier XE 21
6.13 Intel Trace Analyzer and Collector 21
6.13.1 How to filter tracing 22

1 Scope of this document

This document gives an introduction to the usage of the CHEOPS cluster, which includes the overview of the development software environment. For information on the application software please see the document “CHEOPS Application Software”.

2 Contacts for support

HPC team, hpc-mgr@uni-koeln.de:
Operation of HPC systems, parallel computing, batch system, development software

Scientific support, wiss-anwendung@uni-koeln.de:
Scientific applications, scientific computing, access to HPC systems

If you send a support request to one of these contacts, please provide all relevant information to describe the issue you encountered. First of all, error messages are crucial for analysis and should be provided with the request. Such messages are usually printed to standard error (`stderr`) or standard output (`stdout`) streams. Depending on what you are doing you will either see these messages on the screen or the streams are redirected into files. Moreover, accompanying information is very helpful to shorten analysis. For instance, if the batch system fails to run a job, you should always provide the job identifier with your report. If building of an application fails with an error, you should provide name and version of compiler and libraries used.

3 Description of the system

CHEOPS is a high performance computing cluster for science, which is provided by the Regional Computing Centre of Cologne (RRZK) together with the HPC expert Bull.

Foto: Thomas Josek/JosekDesign

The complete cluster is in production since the end of 2010. With more than 100 TFlop/s peak and 85.9 TFlop/s Linpack performance, it was ranked to be the 89th fastest supercomputer worldwide in the Top500 list from November 2010 (www.top500.org).
CHEOPS is an InfiniBand coupled HPC cluster with dual socket INCA compute nodes:

- 210 x 2 Nehalem EP quad-core processors (Xeon X5550, 2.66 GHz), 24 GB RAM
- 5 x 2 Nehalem EP quad-core processors (Xeon X5550, 2.66 GHz), 96 GB RAM
- 432 x 2 Westmere hexa-core processors (Xeon X5650, 2.66 GHz), 24 GB RAM
- 170 x 2 Westmere hexa-core processors (Xeon X5650, 2.66 GHz), 48 GB RAM

and quad socket MESCA compute nodes:

- 24 x 4 Nehalem EX octo-core processors (Xeon X7560, 2.27 GHz), 512 GB RAM

The node interconnect is based on InfiniBand QDR, which is a broadband bus technology with exceptionally small latency. It was specifically developed for HPC clusters and delivers a gross bandwidth of 40 Gb/s and latencies of less than 5 µs. The INCA compute nodes provide two Intel Nehalem EP quad-core processors (Xeon X5550) and Intel Westmere hexa-core processors (Xeon X5650), respectively. The MESCA nodes are equipped with four Intel Nehalem EX octo-core processors (Xeon X7560). Intel's Nehalem (and Westmere) architecture stands for high memory bandwidth and a fast interconnect between cores as well as processors, which perfectly matches the demands of HPC. Soon, the MESCA nodes will be merged to 6 large nodes with 128 cores and 2 TB RAM to fulfill the most memory demanding jobs.
CHEOPS uses Lustre as parallel file system. Lustre is based on two DDN racks delivering a gross capacity of 500 TB (400 TB net). To provide the required performance, 4 Object Storage Servers (OSS) deliver the data to the compute nodes. The metadata is stored on an EMC box, which is exposed through 2 Meta Data Servers (MDS).

3.1 Who may use this service?

Scientists from NRW may use CHEOPS for scientific, non-commercial purposes.

3.2 How do I gain access?

To gain access to CHEOPS, you need an Uni-account and HPC authorization. The Uni-account enables members of the University of Cologne to use common services offered by the RRZK. The HPC authorization additionally enables access to the HPC systems. If you need support in filling the forms, you may ask for help at the RRZK dispatch or scientific support.

3.2.1 How to get an Uni-account

As a member of the University of Cologne you should already have an Uni-account. Members from other universities of the federal state Nordrhein-Westfalen and students working in projects may apply for such an account using the Uni-account form.
With your Uni-account there comes an according email account username@uni-koeln.de. Please retrieve emails from this account to receive important messages (e.g. maintenance of the cluster, expiration of your account). If you are using a different email account, you should set an according forward on the mail portal.
Please note that if you are already a user of HPC systems at the RRZK, you will only need to download the HPC authorization form and apply for either a test account or a full account including a project description.

3.2.2 How to get HPC authorization

With the HPC authorization you also get an account on CHEOPS.
If you only need a small amount of computing time (e.g. your project is in an early phase and you want to test your application), you may apply for a test account. It will be limited to a certain amount of computing time (e.g. 1000 CPU hours). For a test account there is no need for a project description.
If you need more computing time, you may apply for a full account which requires a detailed project description. Of course you can apply for a full account right away, if you are already set to go and have a clear description of your project.
In both cases please use the HPC authorization form.

Links:
http://rrzk.uni-koeln.de/hpc-account.html
https://mailportal.uni-koeln.de

3.3 Server addresses

Function	Address	Protocols	Status
Login Node	cheops.rrz.uni-koeln.de	ssh	OK
NFS Gateway	cheops-nfs.rrz.uni-koeln.de		WIP
CIFS Gateway	cheops-cifs.rrz.uni-koeln.de		WIP
Monitoring/Ganglia	https://cheops-mgm.rrz.uni-koeln.de/ganglia/	https	WIP

All Nodes are accessible within the university network (UKLAN). If you are outside this network, you may login through the server **`dialog.rrz.uni-koeln.de`** and from there access CHEOPS via Secure Shell (SSH) with trusted X11 forwarding:

```
ssh -Y userid@dialog.rrz.uni-koeln.de
ssh -Y userid@cheops.rrz.uni-koeln.de
```

Alternatively, you may connect to UKLAN via a Virtual Private Network (VPN) client. To do this, follow the instructions for VPN access on the RRZK webpage. You will then be virtually integrated into the UKLAN and will be able to directly connect to all services including monitoring.

Links:
http://rrzk.uni-koeln.de/vpn.html

3.4 Structure of the file system

As our users are used to, the parallel storage based on Lustre is organized into three file systems with differing bandwidth. The following table gives an overview:

File system/ location	Capacity	Defined capacity for users	Speed	Backup/archive
`/home`	56 TiB (62 TB)	100 GB (100,000 files	Low	Daily

		soft quota, 150,000 files hard quota)		
/projects	139 TiB (153 TB)	As requested	Medium - low	Archive must be requested (elsewise none)
/scratch	167 TiB (184 TB)	No limits	High	None (data will be deleted after 30 days)

NOTE: Parallel file systems are not generally known to be the most stable files systems. Please make sure that important data in **/projects** is archived (see Section 3.6).

As can be seen in the table, the number of files in the home directory is limited by a separate quota. The reason for this is that the amount of files has severe impact on the performance of a parallel file system as well as on backups. If you are getting in trouble because of this, please contact the RRZK scientific support or HPC team.

3.5 What you should do on first login

If you access CHEOPS for the first time via SSH, a key pair is generated for the access to the compute nodes. Please do **not** provide a password for the keys. Accessing the compute nodes has to work without a password. After this, you are set to go!

3.6 How to archive data

To archive data in your project directory, you first need a TSM registration for archiving. With the registration you are provided a **nodename** referring to your archive space. Then, you can access this archive space with the TSM client on the login node **cheops.rrz.uni-koeln.de**. Before archiving of data you should create a personal configuration file **dsm.opt** with the following content:

```
servername      adsm4n
virtualnode     nodename
subdir          yes
```

The server for archiving is named **adsm4n**. As virtual node you should provide your personal **nodename**. For simple data management subdirectories are to be included. The TSM client can be launched with the command line user interface **dsmc** or the graphical user interface **dsmj**. For archiving you should use the command line client:

```
dsmc archive /projects/project/tobearchived/ \
-des=archivename -optfile=/path_to_dsm.opt/dsm.opt
```

The archive name **archivename** will help you finding data to be retrieved from a specific archive. For retrieving data you should use the graphical client

```
dsmj -optfile=/path_to_dsm.opt/dsm.opt
```

Within the graphical user interface you can browse the content of your archives for data to be retrieved.

NOTE: The purpose of an archive is different from that of a backup. A backup saves frequently changing data. However, an archive saves data that does not change anymore (e.g. results). Therefore, you should not archive your whole project directory but finished subdirectories. You can retrieve this data on any computer with a TSM client (e.g. workstation or laptop) for further processing.

Links:
http://rrzk.uni-koeln.de/tsm.html
http://publib.boulder.ibm.com/infocenter/tsminfo/v6/index.jsp

4 Environment modules

CHEOPS provides a wide range of development software (compilers, libraries, debuggers, profilers, etc.) as well as specific scientific applications. Many of the available programs require certain environment variables to be set or changed, e.g. **`PATH`**, **`LD_LIBRARY_PATH`**, **`MANPATH`**. The environment modules package is employed to access or switch between various, sometimes conflicting versions of software. It provides the means to change the environment dynamically by loading, switching or unloading specific software modules. The module installation is customized to automatically resolve dependencies between modules. Furthermore, a quick usage primer is displayed upon loading.

4.1 Overview of important module commands

Command	Function
`module avail`	Shows all available modules
`module whatis [`*`module`*`]`	Shows a short description of all or a specific module
`module display|show `*`module`	Shows the environment changes the module would have applied if loaded
`module add|load `*`module`	Loads the module ***`module`***, i.e. sets all necessary variables for the corresponding software and loads required modules
`module list`	Shows all currently loaded modules
`module rm|unload `*`module`	Removes the module ***`module`***, i.e. removes all environment settings, which were introduced by loading the module, and removes dependent modules
`module purge`	Removes all loaded modules

4.2 Usage notes

A module can either be identified by its name only or its name together with a specific version, for example **`intel`** or **`intel/13.1`**, respectively. When only the name is specified, the default version as shown by **`module avail`** is loaded.

- Loading a module changes the environment of the shell in which it is loaded
- If other already loaded modules conflict with the module to load, an according error message is displayed and no further changes are made

- If other modules are required by the module to load, these are loaded recursively and an according message is displayed
- Removing a module reverts the changes in the environment of the shell
- If other modules depend on the module to remove, these are removed recursively and an according message is displayed

Links:
http://modules.sourceforge.net

5 Batch system (SLURM)

CHEOPS uses the Simple Linux Utility for Resource Management (SLURM) to control the usage of the compute nodes and to schedule and execute jobs.
Interaction with the system is carried out by SLURM commands, e.g. **`sbatch`**, **`squeue`**, **`sinfo`**, **`scancel`**. Resource requirements can be added directly as command line options or through appropriate **`#SBATCH`** directives in the job scripts. Users can submit jobs from the login node **`cheops`**.
The batch system offers several partitions, also called queues, which have various resource limits and requirements. Jobs submitted without a partition specified are automatically routed to a partition which corresponds to the given resource request; should this fail due to exceeded limits, the job is immediately rejected at submission time with a corresponding error message. Multi-node jobs are routed to the partitions **`mpi`** or **`mpimesca`**. The **`mpi`** partition is for jobs with common memory requests and assigns to INCA nodes. The **`mpimesca`** partition is for jobs with huge memory requests and assigns to MESCA nodes. By default the **`mpimesca`** partition is only enabled for job submission but not started for job execution. Please ask for execution of such a job with an email to the RRZK HPC team. The development partition **`devel`** is for short test jobs – it offers a high response time albeit at the price of strict limits. It must be explicitly selected with the **`--partition`** option. Here the current partition based resource limits and requirements for all partitions:

Partition	Resource	Minimum	Maximum
`mpi`	Number of nodes	2	128
	Number of cores	9	1536
	Memory per node	--	94gb
	Total wall time	--	360 h
`mpimesca`	Number of nodes	2	16
	Number of cores	33	512
	Memory per node	95gb	504gb
	Total wall time	--	360 h
`smp`	Number of nodes	1	1
	Number of cores	1	32
	Memory per node	--	504gb
	Total wall time	--	720 h
`devel`	Number of nodes	1	2
	Number of cores	1	16
	Memory per node	--	23gb
	Total wall time	--	1 h

Node based resource limits introduced by features of different node types are listed in the feature table in Section 5.5. Please use those to specify resource requests in detail.

5.1 Important differences between SLURM and TORQUE/Maui

CHEOPS has been using TORQUE/Maui as a batch system for almost 3 years. To facilitate an easier transition for current users, here an overview of important differences and new features:

- **The command set and terminology**. Unlike TORQUE/Maui where the resource manager and scheduler were two separate programs, SLURM is a modular system which combines various functionalities into a single framework. The specific commands have changed significantly and are explained later on. The names of some concepts have changed as well: queues are now called partitions, node properties are node features.
- **Resources for processes and threads**. SLURM has introduced new concepts to request resources for processes and threads for better handling of SMP, MPI and hybrid MPI jobs. Tasks are the resource required for processes including MPI processes and can be requested in total with the option **`--ntasks`** or per node with the option **`--ntasks-per-node`**. Multiple cores per task are required for threads and can be requested with the option **`--cpus-per-task`**. These settings are automatically evaluated by the SLURM launcher **`srun`**, so additional options for the MPI runtime environment are no longer necessary.
- **MPI launcher**. The common launcher **`mpirun`** from the MPI libraries has been replaced with the SLURM launcher **`srun`** (see Sections 6.7).
- **Resource memory.** The memory request is not specified in total per job anymore but per node. So, keep the memory request when scaling an MPI job to more nodes. The default unit is now Megabytes.
- **Resource containment**. The enforcement of resource limits is more thorough than in TORQUE/Maui. All processes of a job are now bound and limited to the requested cores, meaning that any oversubscription slows down the offending job itself rather than other jobs. Memory monitoring and limitation is now functional for all MPI runtime environments, thus applications that exceed their memory limit are canceled.
- **Export of environment**. In addition to the export of environment variables from the job script to the application processes known from TORQUE/Maui, SLURM also offers the export of shell limits even from the shell in which the job is submitted. Thereby, shell limits set in **`.bashrc/.cshrc`** are automatically propagated to the application processes. For interactive jobs even the environment variables from the submission shell are exported. Modules loaded at job submission are then carried over to the shell of the interactive job. This feature has been disabled for batch jobs to preserve the behavior known from TORQUE/Maui. To enable it the **`sbatch`** option **`--export=ALL`** can be used.
- **Current working directory**. The SLURM job automatically starts within the submission directory; there is no need to change the directory anymore.
- **Output files**. In contrast to TORQUE/Maui where the output files were moved to the submission location after job completion, the output file in SLURM is created there immediately. Thus, the output file can be viewed while the job is running. Per default both streams (**`stdout,stderr`**) are joined into the same file. The output file name does no longer contain the job name per default. This can be set explicitly with the appropriate option **`--output`**.

5.2 Overview of important SLURM commands

Command	Function
sbatch *script.sh*	Submits the job script ***script.sh***
squeue	Shows all current jobs
squeue –j *jobid*	Shows detailed status information on job ***jobid***
sstat *jobid*	Shows detailed runtime information on job ***jobid***
scontrol show job *jobid*	Shows detailed resource information on job ***jobid***
scancel *jobid*	Cancels the job
squeue --start –j *jobid*	Shows an approximate time slot for job ***jobid***. This estimation might change significantly until the job starts.

5.3 Batch job scripts

Job scripts are executable shell scripts, which contain resource request as well as the actual commands to be executed. Please note that modules necessary for successful compilation should be loaded accordingly in the job script during runtime. The runtime environment provides variables to refer to assigned resources: total number of tasks by **SLURM_NTASKS**, number of tasks per node by **SLURM_NTASKS_PER_NODE**, and number of cores per task by **SLURM_CPUS_PER_TASK**. For a quick start, use the following script templates:

SMP Job with multiple threads (e.g. 4 threads)

```
#!/bin/bash -l
#SBATCH --cpus-per-task=4
#SBATCH --mem=1024mb
#SBATCH --time=01:00:00
#SBATCH --account=UniKoeln

# number of nodes in $SLURM_NNODES (default: 1)
# number of tasks in $SLURM_NTASKS (default: 1)
# number of tasks per node in $SLURM_NTASKS_PER_NODE (default: 1)
# number of threads per task in $SLURM_CPUS_PER_TASK (default: 1)

module load intel/13.1

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
./a.out
```

Common MPI Job (e.g. 2 nodes, 8 MPI processes per node)

```
#!/bin/bash -l
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=8
#SBATCH --ntasks=16
#SBATCH --mem=8gb
#SBATCH --time=01:00:00
#SBATCH --account=UniKoeln

# number of nodes in $SLURM_NNODES (default: 1)
# number of tasks in $SLURM_NTASKS (default: 1)
# number of tasks per node in $SLURM_NTASKS_PER_NODE (default: 1)
# number of threads per task in $SLURM_CPUS_PER_TASK (default: 1)
```

```
module load intel/13.1 intelmpi/4.1.0

srun -n $SLURM_NTASKS ./a.out
```

Hybrid MPI Job (e.g. 2 nodes, 2 MPI processes per node, 4 threads per MPI process)

```
#!/bin/bash -l
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=2
#SBATCH --ntasks=4
#SBATCH --cpus-per-task=4
#SBATCH --mem=2gb
#SBATCH --time=01:00:00
#SBATCH --account=UniKoeln

# number of nodes in $SLURM_NNODES (default: 1)
# number of tasks in $SLURM_NTASKS (default: 1)
# number of tasks per node in $SLURM_NTASKS_PER_NODE (default: 1)
# number of threads per task in $SLURM_CPUS_PER_TASK (default: 1)

module load intel/13.1 intelmpi/4.1.0

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
srun -n $SLURM_NTASKS ./a.out
```

In Bourne shell like batch jobs the modules functionality is properly initialized by adding the **-l** option, e.g. **#!/bin/sh -l**, in C shell like jobs no addition is needed. The following table provides a more detailed description of the most common resource specifications:

#SBATCH Option	Function
--partition=*partition*	Defines the destination of the job. Only necessary for partition ***devel***. Other partitions are automatically addressed according to resource request. Default: automatic routing
--nodes=*nodes*	Defines the number of nodes for the job. Default: 1
--ntasks=*ntasks*	Defines the number of tasks for the job. Each task is mapped to a core, unless the **--cpus-per-task** option is specified. Default: 1
--ntasks-per-node=*tpn*	Defines the maximum number of tasks per node. Used mainly with MPI applications. Default: 1
--cpus-per-task=*cpt*	Defines the number of cores per task. The resource manager will allocate those cores for threads within the according task. Default: 1
--mem=*mem*	Defines the per-node memory limit of the job. The job will be canceled automatically when this limit is exceeded. The format for ***mem*** is an integer value followed by ***mb*** or ***gb***, e.g. ***10gb***. Default: **--mem=*1000mb***
--time=*walltime*	Defines the wall time limit of the job. The job will be canceled automatically when this limit is exceeded. The format for ***walltime*** is ***HH:MM:SS***. Default: **walltime=*01:00:00***

--account=*acct_string*	Defines the account string. You should provide the appropriate project/account. Default: ***UniKoeln***
--output=*filename*	Specifies file ***filename*** to collect **stdout** stream. Default: join both **stdout** and **stderr** into ***slurm-%j.out*** with job ID ***%j***
--error=*filename*	Specifies file ***filename*** to collect **stderr** stream. Default: none
--mail-type=BEGIN, END,FAIL,REQUEUE,ALL	Specifies when email is sent to the job owner. The option argument may consist of a combination of the allowed mail types.
--mail-user= *username@uni-koeln.de*	Specifies the address to which email is sent. Default: none
--array=*array_request*	Specifies a job array (see Section 5.4).

5.4 Job arrays and interactive jobs

In addition to common jobs there are two more special job types. The job array is meant for jobs equal in script structure but using different input. It allows the submission of many job instances of a single job script. The set of instances is provides by the argument ***array_request*** following the specifying option **--array**. The format of ***array_request*** is a range or a comma delimited list of ranges, e.g. ***1-5*** or ***1,3-5***. Each instance is assigned to a new job ID which is provided by the environment variable **SLURM_JOB_ID**. The task identifier can be referenced in the job script with **SLURM_ARRAY_TASK_ID** while the job array identifier is retained in **SLURM_ARRAY_JOB_ID**. Please refer to the following example script for a job array:

Bourne shell like

```
#!/bin/bash -l
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --mem=1gb
#SBATCH --time=01:00:00
#SBATCH --account=UniKoeln
#SBATCH --array=0-2

module load intel/13.1 mkl/11.0

options=( "in.0" "in.1" "in.2" )

option=${options[SLURM_ARRAY_TASK_ID]}

echo "Job: ${SLURM_ARRAY_JOB_ID}, Task: ${SLURM_ARRAY_TASK_ID}, In-
put: ${option}, Output out.${SLURM_JOB_ID}"

./a.out ${option} > out.${SLURM_JOB_ID}
```

The total number of submitted jobs and job array instances per user is limited to 1536. Please keep in mind that job arrays instances should take a reasonable amount of wall time, i.e. longer than 1 hour. Shorter runs should be merged into one job array instance appropriately. The resource requests in the job script apply to each instance, not the entire array.

The interactive job offers flexibility for tests and experiments. By invoking **salloc** with all other necessary options, an interactive job is queued and waits for scheduling. When the job is started, the job owner is given a shell for interactive use until one of the resource limits is exceeded. For example:

```
salloc --cpus-per-task=8 --mem=1gb --time=01:00:00
--account=UniKoeln srun -n 1 --pty bash
```

will provide a bash shell on a compute node with 8 available cores and 1GB memory for one hour. X11 forwarding may be enabled by adding the option **--x11=first** to **salloc** and **srun**.

5.5 Resource requests

The batch system offers two basic procedures to allocate jobs on compute nodes:

- The user can provide all necessary resources and resource limits, e.g. the number of nodes and tasks per node, the maximum memory per node and the maximum wall time. Then, the job will automatically be allocated to nodes which satisfy these requests and waste as few cores and little memory as possible. For example

  ```
  sbatch --nodes=2 --ntasks-per-node=4 --mem=50gb script.sh
  ```

  automatically allocates the job to 2 nodes which have 4 cores and 50 GB available each. Nodes with fewer available cores and less available memory which meet the requests are preferred to reserve the remaining resources for larger jobs.
- In addition to given resources and resource limits the user can explicitly request specific groups of nodes by features. The allocation then takes only those nodes into account. The syntax is as follows

  ```
  sbatch --nodes=nnodes --ntasks-per-node=tpn --constraint
  feature script.sh
  ```

  Multiple features can be specified, together with logical operators like AND, OR, XOR and a number of nodes required with each set of groups, for example:

  ```
  sbatch --nodes=6 --ntasks-per-node=4 --constraint=
  "[inca8*4&inca12*2]" script.sh
  ```

  allocates the job with 6 nodes with 4 tasks each to 4 INCA Nehalem EP nodes and 2 INCA Westmere nodes.

Please refer to the following table for a detailed description of the available features to identify according node types with certain resource limits:

Feature	Processor type	#cores	Physical memory	Available memory
inca	Nehalem EP/Westmere	8/12	24/48/96 GiB	23/47/94gb
inca8	Nehalem EP	8	24/96 GiB	23/94gb
inca8_24gb	Nehalem EP	8	24 GiB	23gb
inca96gb	Nehalem EP	8	96 GB	94gb
inca12	Westmere	12	24/48 GiB	23/47gb
inca12_24gb	Westmere	12	24 GiB	23gb
inca48gb	Westmere	12	48 GiB	47gb
inca24gb	Nehalem EP/Westmere	8/12	24 GiB	23gb
mesca	Nehalem EX	32	512 GiB	504gb

Links:
http://www.schedmd.com/slurmdocs/documentation.html

6 Development environment

The three major parts of the development software environment are compilers, libraries and tools. Currently, the installation looks like that:

- Compilers
  - Intel compilers (version 12.0, 12.1, 13.0, 13.1)
  - GNU compilers (version 4.4.6, 4.6.2)
  - PGI compilers (version 10.1, 10.5, 11.1, 11.8)
- Libraries
  - Intel MKL (Math Kernel Library, version 10.3, 11.0)
  - Intel MPI (version 4.0.3, 4.1.0, 4.1.1)
  - NAG (Mark 09, 21, 22, 23, 24)
- Tools
  - Intel Debugger (version 12.0, 12.1, 13.0, 13.1)
  - Allinea DDT Debugger (version 4.1)
  - Intel Inspector XE (version 2013)
  - PAPI (Performance Application Programming Interface, version 5.0.1)
  - Intel VTune Amplifier XE (version 2013)
  - Intel Trace Analyzer and Collector (version 8.0.3, 8.1.0, 8.1.2)
  - Vampir (version 8.1)

If there is no preference on how to build an application, the Intel software should be used.

6.1 Compilers

The environment of the compilers is provided by environment modules. After loading one of these modules several commands invoke the compilers for according programming languages:

Compiler	Modules	C/C++	Fortran 77/90
Intel Compilers	`intel/12.0`, `intel/12.1`, `intel/13.0`, `intel/13.1`	`icc/icpc`	`ifort/ifort`
PGI Compilers	`pgi/10.1`, `pgi/10.5`, `pgi/11.1`, `pgi/11.8`	`pgcc/pgCC`	`pgf77/pgf90`
GNU Compilers	`gnu/4.4.6`, `gnu/4.6.2`	`gcc/g++`	`g77/gfortran`

Nehalem specific code can be generated by `-xhost` or `-xSSE4.2` with the Intel compilers and by `-tp nehalem-64` with the PGI compilers. Common optimizations are performed by the general option `-fast` for both Intel and PGI compilers.

Links:
http://software.intel.com/en-us/articles/intel-c-composer-xe-documentation
http://software.intel.com/en-us/articles/intel-fortran-composer-xe-documentation
http://www.pgroup.com/resources/docs.htm

6.2 Intel MKL

The MKL (Math Kernel Library) includes the functionality of a lot of other libraries (BLAS, LAPACK, FFT, Sparse BLAS) addressing mathematical operations and algorithms often needed

in scientific computing. Both shared memory and distributed memory is supported by multi-threaded routines and implementations using message passing (ScaLAPACK, PBLAS, BLACS), respectively. Again, MKL is made available by the according environment modules (`mkl/10.3, mkl/11.0`). In C/C++ the MKL header `mkl.h` has to be included. The common linkage scheme for sequential processing is compiler specific because of the Fortran runtime libraries needed:

Compiler	Linkage in C/C++	Linkage Fortran 77/90

Intel Compilers	`-lmkl_intel_lp64`
	`-lmkl_sequential`
	`-lmkl_core -lpthread`
		`-lmkl_intel_lp64`
		`-lmkl_sequential`
		`-lmkl_core -lpthread`

PGI Compilers	`-lmkl_intel_lp64`
	`-lmkl_sequential`
	`-lmkl_core -lpthread`
		`-lmkl_intel_lp64`
		`-lmkl_sequential`
		`-lmkl_core -lpthread`

GNU Compilers	`-lmkl_intel_lp64`
	`-lmkl_sequential`
	`-lmkl_core -lpthread`
	`-lgfortran`
		`-lmkl_gf_lp64`
		`-lmkl_sequential`
		`-lmkl_core -lpthread`

The common linkage scheme for multi-threaded processing is more complicated due to the compiler specific threading:

Compiler	Linkage in C/C++	Linkage Fortran 77/90

Intel Compilers	`-lmkl_intel_lp64`
	`-lmkl_intel_thread`
	`-lmkl_core`
	`-liomp5 -lpthread`
		`-lmkl_intel_lp64`
		`-lmkl_intel_thread`
		`-lmkl_core`
		`-liomp5 -lpthread`

PGI Compilers	`-mp`
	`-lmkl_intel_lp64`
	`-lmkl_pgi_thread`
	`-lmkl_core`
	`-pgf90libs`
		`-mp`
		`-lmkl_intel_lp64`
		`-lmkl_pgi_thread`
		`-lmkl_core`
		`-pgf90libs`

GNU Compilers	`-lmkl_intel_lp64`
	`-lmkl_gnu_thread`
	`-lmkl_core`
	`-liomp5 -lpthread`
	`-lgfortran`
		`-lmkl_gf_lp64`
		`-lmkl_gnu_thread`
		`-lmkl_core`
		`-liomp5 -lpthread`

The number of threads can be set by the environment variable `MKL_NUM_THREADS` or by the more general OpenMP variable `OMP_NUM_THREADS`.

Links:
http://software.intel.com/en-us/articles/intel-math-kernel-library-documentation
http://software.intel.com/en-us/articles/intel-mkl-link-line-advisor

6.3 NAG Numerical Library

The NAG Numerical Library consists of numerical, symbolic, statistical and simulation routines for the solution of problems in a wide range of applications in such areas as science, engineering, financial analysis and research. The NAG Numerical Library is written in various languages where the following shared library versions are made available on CHEOPS by according environment modules. Some of the Fortran libraries may also be linked with programs written in C language. Please note that the order of the listed modules is dependent.

6.3.1 NAG Fortran 77 Library

Compiler	Modules	Linkage Fortran 77/90	Linkage in C/C++
Intel	`intel` `nag/f_22`	`-lnag_nag`	`-lnag_nag`

Intel &
NAG MKL
	`intel`
	`nag/f_22`
		`-lnag_mkl`
		`-lmkl_intel_lp64`
		`-lmkl_sequential`
		`-lmkl_lapack`
		`-lmkl_core`
			`-lnag_mkl`
			`-lmkl_intel_lp64`
			`-lmkl_sequential`
			`-lmkl_lapack`
			`-lmkl_core`

Intel &
Intel MKL
	`intel`
	`nag/f_22`
	`mkl`
		`-lnag_mkl`
		`-lmkl_intel_lp64`
		`-lmkl_sequential`
		`-lmkl_lapack`
		`-lmkl_core`
			`-lnag_mkl`
			`-lmkl_intel_lp64`
			`-lmkl_sequential`
			`-lmkl_lapack`
			`-lmkl_core`

PGI	`pgi` `nag/f_22`	`-lnag_nag`	`-lnag_nag`
PGI & NAG ACML	`pgi` `nag/f_22`	`-lnag_acml -lacml`	`-lnag_acml -lacml`
GNU	`gnu` `nag/f_21`	`-lnag_nag`	`-lnag_nag`
GNU & NAG ACML	`gnu` `nag/f_21`	`-lnag_acml -lacml`	`-lnag_acml -lacml`

Header files to be included for C/C++ can be found in the documentation of the specific routine (see NAG online documentation).

6.3.2 NAG C Library

Compiler	Modules	Linkage in C/C++
Intel	`intel` `nag/c_09`	`-lnagc_nag`
Intel & NAG ACML	`intel` `nag/c_09`	`-lnagc_acml -lacml`
PGI	`pgi` `nag/c_09`	`-lnagc_nag`
PGI & NAG ACML	`pgi` `nag/c_09`	`-lnagc_acml -lacml`
GNU	`gnu` `nag/c_09`	`-lnagc_nag`
GNU & NAG ACML	`gnu` `nag/c_09`	`-lnagc_acml -lacml`

Header files to be included can be found in the documentation of the specific routine (see NAG online documentation).

6.3.3 NAG Fortran 90 Library

Compiler	Modules	Linkage Fortran 90
Intel & NAG MKL	**intel** **nag/f90_04**	**-lnagf190_mkl** **-lmkl_lapack64 -lmkl**
Intel & Intel MKL	**intel** **nag/f90_04** **mkl**	**-lnagf190_mkl** **-lmkl_intel_lp64 -lmkl_sequential** **-lmkl_core**

Fortran modules to be used can be found in the documentation of the specific routine (see NAG online documentation).

6.3.4 NAG Fortran 77 Library for SMP & multicore

Compiler	Modules	Linkage Fortran 77/90
Intel & NAG MKL	**intel** **nag/fs_21**	**-fpp -openmp -auto** **-lnagsmp** **-lmkl_lapack64 -lmkl** **-lguide -lpthread**
Intel & Intel MKL	**intel** **nag/fs_21** **mkl**	**-fpp -openmp -auto** **-lnagsmp** **-lmkl_intel_lp64** **-lmkl_intel_thread -lmkl_core** **-liomp5 -lpthread**
PGI & ACML	**pgi** **nag/fs_21**	**-mp -pgf90libs** **-lnagsmp** **-lacml**

The Fortran 90 runtime libraries for the PGI version (provided by option **-pgf90libs**) are only needed when linking with the Fortran 77 compiler **pgf77**. The number of threads to be used is constrained by the OpenMP environment variable **OMP_NUM_THREADS** and must not exceed the number of cores requested in the job script

```
#SBATCH --cpus-per-task=cpt
```

A common job using the NAG Library for SMP & multicore and automatically determining the number of threads would look like

```
#!/bin/bash -l
#SBATCH --cpus-per-task=4
#SBATCH --mem=8gb
#SBATCH --time=02:00:00
#SBATCH --account UniKoeln

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

module load intel nag/fs_21
ifort -o myprog -fpp -openmp -auto myprog.f \
      -lnagsmp -lmkl_lapack64 -lmkl -lguide -lpthread
./myprog
```

6.3.5 NAG Fortran 77 Parallel Library (MPI version)

Compiler	Modules	Linkage Fortran 77/90
Intel & NAG MKL	**intel** **intelmpi** **nag/fd_21**	**-lnagmpi** **-lnagmpisup** **-lmkl_blacs_intelmpi_lp64** **-lmkl_intel_lp64** **-lmkl_sequential** **-lmkl_core**
Intel & Intel MKL	**intel** **intelmpi** **nag/fd_21** **mkl**	**-lnagmpi** **-lnagmpisup** **-lmkl_blacs_intelmpi_lp64** **-lmkl_intel_lp64** **-lmkl_sequential** **-lmkl_core**

For linking with NAG Parallel Library and Intel MPI, the compiler script **mpiifort** must be used for both compiling and linking. Resulting executables are started within a batch job by

```
srun –n $SLURM_NTASKS ./a.out
```

after loading the required modules where **$SLURM_NTASKS** specifies the number of MPI processes to be used.

6.3.6 Examples & Documentation for the NAG Numerical Library

Each NAG numerical library version provides sample scripts, which allow the user to extract an appropriate sample program for a specific NAG routine, to compile and link the program to the required library and to finally start the resulting executable.
After loading the required module one can find the sample scripts in the directory **$NAGHOME/script** where **$NAGHOME** is a shell variable set by the module. In the mentioned directory one will find several sample scripts for linking the NAG library with or without MKL/ACML library, static or shared. Usage of the shared version is recommended, e.g.

```
module intel nag/f_22
$NAGHOME/scripts/nag_example_shar e04ucf
```

where **e04ucf** is the routine a check should be made for.
For more information on the NAG Numerical Library and the description of its routines see the NAG online documentation.

Links:
http://www.nag.com/numeric/numerical_libraries.asp

6.4 ARPACK

The Arnoldi package ARPACK is a library for solving large eigenvalue problems. It computes a few eigenvalues and corresponding eigenvectors of large sparse matrices by the Implicitly Restarted Arnoldi Method (IRAM). PARPACK is an MPI parallel version of ARPACK. Both libraries feature a reverse communication interface, which needs only the action of the given matrix on a vector. Again, ARPACK is made available by the according environment modules (**arpack/arpack96**, **arpack/parpack96**). The linkage scheme for sequential ARPACK is compiler specific because of the Fortran runtime libraries needed:

Compiler	Linkage in C/C++	Linkage Fortran 77/90
Intel Compilers	**-larpack_lp64 -lifcore -lgfortran**	**-larpack_lp64**
PGI Compilers	**-larpack_lp64 -pgf90libs**	**-larpack_lp64**
GNU Compilers	**-larpack_lp64 -lgfortran**	**-larpack_lp64**

Following these options the MKL options have to be provided since ARPACK was built with the MKL (see Section 6.2). Finally, the linkage scheme for the MPI parallel PARPACK only needs the additional library in front of the ARPACK linkage:

Compiler	Linkage in C/C++	Linkage Fortran 77/90
Intel Compilers	**-lparpack_lp64 -larpack_lp64 -lifcore -lgfortran**	**-lparpack lp64 -larpack_lp64**
PGI Compilers	**-lparpack_lp64 -larpack_lp64 -pgf90libs**	**-lparpack lp64 -larpack_lp64**
GNU Compilers	**-lparpack lp64 -larpack_lp64 -lgfortran**	**-lparpack_lp64 -larpack_lp64**

and should be invoked with the according compiler scripts from the MPI library used. Resulting executables are again started within a batch job by

```
srun -n $SLURM_NTASKS ./a.out
```

with the number of processes provided by the SLURM variable **SLURM_NTASKS**.

Links:
http://www.caam.rice.edu/software/ARPACK

6.5 Intel MPI Library

Intel MPI provides a full implementation of the MPI-2 standard. Again, Intel MPI is made available by according environment modules (**intelmpi/4.0.3**, **intelmpi/4.1.0**, **intelmpi/4.1.1**). The compiler scripts including header files and linking libraries have compiler specific names:

Compiler	C/C++	Fortran 77/90
Intel Compilers	**mpiicc/mpiicpc**	**mpiifort/mpiifort**
PGI Compilers	**mpipgcc/mpipgCC**	**mpipgf77/mpipgf90**
GNU Compilers	**mpigcc/mpigxx**	**mpif77/mpif90**

Integers of 8 bytes size are supported by the option **-ilp64**, thread safety is provided by the option **-mt_mpi**. Please note that Intel MPI does not support transfers of long doubles from PGI compilers.

6.6 Open MPI

Open MPI is an open source implementation of the MPI-2 standard. It is made available by according environment modules (**openmpi/1.6.5**, **openmpi/1.6.5_ilp64**, **openmpi/1.6.5_mt**, **openmpi/1.6.5_ilp64_mt**). Both support of 8 bytes integers (suffix **ilp64**) and thread safety (suffix **mt**) are provided by specific modules pointing to according builds. The compiler scripts including header files and linking libraries have the same naming for all compilers:

Compiler	C/C++	Fortran 77/90
All Compilers	**mpicc/mpicxx**	**mpif77/mpif90**

Links:
http://www.open-mpi.org

6.7 MPI Runtime Environment

With the new batch system SLURM, MPI applications can only be executed within jobs. The batch system provides its own launcher **srun**, which tightly integrates the application processes. Therefore, only the number of MPI processes has to be specified

```
srun -n $SLURM_NTASKS ./a.out
```

and is provided by SLURM as the number of tasks. Interactive invocation of this command is only possible within interactive jobs. The SLURM runtime environment also provides pinning and affinity of application processes according to the assigned resources. Therefore, options or environment variables from the MPI libraries changing the runtime environment like pinning are ignored and should not be used anymore (see Section 5.1).

6.8 Intel Debugger

The Intel Debugger is part of the Intel compilers and becomes available after loading an according module. Executables to be analyzed are instrumented as usual by

```
icc|icpc|ifort -g -O0
```

including debugging information and disabling any optimization. Then, the debugger is launched with the command line user interface **idbc** or the graphical user interface **idb**. Within these environments the application is executed and analyzed. Single source lines or even assembly lines can be resolved within the analysis.

Links:
http://software.intel.com/en-us/articles/intel-c-composer-xe-documentation

6.9 Allinea DDT Debugger

DDT is a debugger from Allinea for parallel applications (both multi-threaded and MPI). As DDT is integrated with the Intel MPI and Open MPI libraries, the according environment modules (**ddt/4.1**) require an Intel MPI or Open MPI module. When using DDT with the Intel MPI library, components of the Intel Trace Analyzer and Collector are needed for the Intel Message Checker plugin and the according Trace Analyzer and Collector module is required as well. Executables to be analyzed are instrumented by

```
mpiicc|mpipgcc|mpigcc|… -g -O0
```

including debugging information and disabling any optimization. Then, the debugger is launched with its graphical user interface **ddt**. Within this environment applications are executed via batch system by default. However, changing to interactive execution is possible e.g. within an interactive batch job environment on the local compute node. In the analysis single source lines can be resolved for different threads and processes, respectively.

Links:
http://www.allinea.com/products/ddt-support

6.10 Intel Inspector XE

The Inspector is the new memory debugger for multi-threaded applications following the Thread Checker. As it also relies on the Intel compilers, the according environment module (**inspxe/2013**) requires an Intel compiler module. Executables to be analyzed are instrumented by

```
icc|icpc|ifort -g -O0
```

Then, the Inspector can be launched with the command line user interface **inspxe-cl** or the graphical user interface **inspxe-gui**. Within these environments the application is executed and analyzed. Issues in memory access (memory leaks, race conditions, etc.) are pointed out. In the analysis single source lines from the relating code sequences can be resolved for different threads and processes, respectively.

Links:
http://software.intel.com/en-us/articles/intel-inspector-xe-documentation

6.11 PAPI Library

The PAPI library collects information from the performance counters in the processor hardware. After loading the according module (**papi/5.0.1**) the utilization of PAPI by High Level Functions is straightforward:

C	Fortran 90

`#include <papi.h>`
 	
`PAPI_flops(&rtime, &ptime, &flpops, &mflops );`
	`#include "f90papi.h"`
	
	`call PAPIF_flops(real_time,& proc_time, flpops, mflops, check)`

```
/* a lot of work */

PAPI_flops(&rtime, &ptime,
&flpops, &mflops );
printf("%f %f %lld %f\n", rtime,
ptime, flpops, mflops);
```

```
! a lot of work

call PAPIF_flops(real_time,&
proc_time, flpops, mflops, check)
print *, real_time, proc_time,&
flpops, mflops, check
```

Include the header via preprocessing, place according calls around the source code of interest and link with the library (link option **-lpapi**).

Links:
http://icl.cs.utk.edu/projects/papi/wiki/Main_Page

6.12 Intel VTune Amplifier XE

VTune Amplifier is a profiler, which offers time and event based profiling. Like PAPI, VTune Amplifier collects information from the performance counters in the processor hardware. Therefore, the executable does not have to be instrumented. However, VTune Amplifier needs debugging information

```
icc|icpc|ifort -g -O2
```

for a source code resolved analysis. All optimizations have to be specified explicitly since the compiler option **-g** sets no optimization by default. As VTune Amplifier relies on the Intel compilers, the according environment module (**vtune/2013**) requires an Intel compiler module. After loading this module VTune Amplifier can be launched with the command line user interface **amplxe-cl** or the graphical user interface **amplxe-gui**. Within these environments the application is executed and analyzed. Single source lines or even assembly lines can be resolved within the analysis of the performance data. Moreover, a call graph with critical path can be determined.

Links:
http://software.intel.com/en-us/articles/intel-vtune-amplifier-xe-documentation

6.13 Intel Trace Analyzer and Collector

The Trace Analyzer and Collector (TAC) is a profiler for MPI applications. Currently it offers time based profiling by linking with a profiling library. Event based profiling by linking with the PAPI library will be integrated soon. As TAC relies on the Intel MPI library, the according environment modules (**tac/8.0.3, tac/8.1.0, tac/8.1.2**) require an Intel MPI module. After loading one of these modules the whole MPI application is instrumented at compile time by linking with option **-trace**. An execution of the instrumented MPI application results in a trace file, which then can be analyzed

```
traceanalyzer ./a.out.stf
```

with the Trace Analyzer.

6.13.1 How to filter tracing

The instrumentation by compiling with the compiler option **-tcollect** enables filtering of routines to be traced. The filter rules are then defined in an appropriate configuration file for the Trace Collector during runtime. For example, a file **myconfig** containing the lines

```
SYMBOL * FOLD
SYMBOL mykernel* ON UNFOLD
```

restricts tracing to the routine **mykernel** and those routines that it calls. The asterisk after **mykernel** matches a possible underscore in compiler's naming convention. Providing this configuration file during runtime by the environment variable **VT_CONFIG**

```
env VT_CONFIG=myconfig srun -n $SLURM_NTASKS ./a.out
```

results in tracing only MPI functions called in **mykernel**. In the same way specific MPI functions can be filtered throughout the whole MPI applications by using wildcard expressions matching a subset of MPI functions. Please refer to the manual page of **VT_CONFIG** for more details on the Trace Collector configuration file.

Links:
http://software.intel.com/en-us/articles/intel-trace-analyzer-and-collector-documentation