CHEOPS
Cologne High Efficient Operating Platform for Science
Application Software
(Version: 16.08.2019)

Dr. Stefan Borowski
Lech Nieroda
Dr. Lars Packschies
Volker Winkelmann
Kevin Kaatz

EMail: wiss-anwendung@uni-koeln.de
Content

Content .......................................................................................................................... i
Available software (examples) .................................................................................. 1
Gaussian ....................................................................................................................... 1
Turbomole ..................................................................................................................... 2
ORCA .......................................................................................................................... 3
Gromacs ...................................................................................................................... 4
High-level languages for numerical computations ................................................. 6
MATLAB ...................................................................................................................... 6
Sequential MATLAB batch jobs without internal multi-threading ................. 6
Sequential MATLAB batch jobs with internal multi-threading ....................... 7
Parallel MATLAB batch jobs .................................................................................. 8
Batch Jobs without MATLAB Licenses - Using the MATLAB Compiler .......... 9
NAG Toolbox for MATLAB .................................................................................... 10
Scilab ......................................................................................................................... 11
R for Statistical Computing ..................................................................................... 12
Installing additional packages .............................................................................. 12
Batch job running your R program ................................................................. 13
Efficiency and parallelization ............................................................................... 13
Multiple workers on single node ......................................................................... 14
Multiple workers on multiple nodes ................................................................. 14
Multi-threading on single node .......................................................................... 15
Low-level Software for Numerical Computations .............................................. 17
NAG Numerical Library ....................................................................................... 17
Sample batch job for the NAG Fortran 77 Library .......................................... 17
Math Kernel Library (MKL) .................................................................................. 18
Applications for bioinformatic computations .................................................... 19
RAXML ...................................................................................................................... 19
Sequential RAXML batch jobs without multi-threading .................................. 20
Parallel RAXML batch jobs with multi-threading .......................................... 20
MPI parallelized RAXML batch jobs on several computing nodes ............... 21
Hybrid parallelized RAXML batch jobs on several computing nodes .......... 21
Hybrid parallelized RAXML batch jobs on several computing nodes (extd.) ....... 23
MrBayes .................................................................................................................. 24
PhyloBayes-MPI ................................................................................................... 25

This document gives an overview of the software applications available on RRZK’s Cheops cluster. For more information about getting access to Cheops and its system software, please see the document Cheops Brief Instructions

Available software (examples)

Submitting jobs on CHEOPS is fairly straightforward given that environment modules are used. When a software module is loaded, all relevant execution path entries are automatically assigned, library paths are defined and important environment variables are set correctly.

As a consequence, the binaries and scripts you want to use with a specific application or program suite can be started without absolute paths. In addition, only your system parameters like maximum runtime (wall time), memory requirements and number of cores to use (and how they are distributed) have to be set in the batch job script.

You can list the available software modules with the command `module avail`. Please refer to the document *Cheops Brief Instructions*, Section 4 for the `module` command options.

For all calculations it is strongly recommended to use the `/scratch` file system. For Gaussian, an existing user directory in `/scratch` is required. To make sure it exists, use the following command (`-m 700 makes sure that only you have access to the data within the directory):

```
mkdir -m 700 /scratch/$USER
```

**Gaussian**

The following describes a simple and typical Gaussian job script:

```
#!/bin/bash
#SBATCH --job-name=g09_example
#SBATCH --output=g09_example_%j.output
#SBATCH --mem=10GB
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=8
#SBATCH --time=2:00:00
#SBATCH --account=UniKoeln

module load gaussian09/g09.E01
/usr/bin/time -p g09 < input.com >& output.log
```

This particular example reserves 8 cores on 1 node (nodes: number of nodes, ntasks-per-node: number of cores per node). job-name and output are arbitrary. In this case you find the variable “%j” in the output setting - the resulting filename will contain the SLURM Job-ID for your convenience.

A maximum of 10 GB of memory is set, and the runtime is expected not to exceed 2 hours. Please remember, that the amount of memory reserved for the calculation should always be at least 10% higher than the amount of memory that Gaussian uses for the calculation due
to the memory overhead — running the Gaussian binaries requires additional memory that
has to be added to the %mem link0-command. In this particular example, use 9GB.
In addition, you find the account argument. You are kindly asked to set it to the appropriate
value (please refer to Cheops Brief Instructions, Section 5.3). The default value is UniKoeln.
The module load command sets up the environment to run Gaussian jobs using version
g09 rev. E01. The RRZK provides more than one version of the program (see module avail).

Finally, Gaussian can be started just using the g09 command, as usual. Please fill in the cor-
rect input and output filenames to match your calculation setup.

To run a Gaussian calculation on more than one node set nodes and ntasks-per-node to
the appropriate values (e.g. 2 and 8 for a total of 16 cores on 2 nodes). The Gaussian input
file should contain the link0-commands “%nproclinda” set to the value of nodes (2) and
“%nprocshared” set to the value of ntasks-per-node (8).

**Turbomole**
The next example can be used to invoke a Turbomole 7.1 computation using jobex:

```bash
#!/bin/bash -l
#SBATCH --job-name=tm71_example
#SBATCH --output=tm71_example_%j.output
#SBATCH --mem=10GB
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=8
#SBATCH --ntasks=8
#SBATCH --time 2:00:00
#SBATCH --account=UniKoeln

module load turbomole/7.1

time -p jobex -c 1000 > & out.out
```

The script uses similar values as the Gaussian single node example above. Turbomole, how-
ever, requires setting ntasks additionally. The filename for Turbomole output (here: out.out) is arbitrary.
For a Turbomole calculation using 2 or more nodes, refer to the following script:

```bash
#!/bin/bash
#SBATCH --job-name=tm71_example
#SBATCH --output=tm71_example_%j.output
#SBATCH --mem=10GB
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=8
#SBATCH --ntasks=16
#SBATCH --time 2:00:00
#SBATCH --account=UniKoeln
module load turbomole/7.1

time -p jobex -c 1000 > out.out
```

**ORCA**

ORCA is an MPI parallel quantum chemistry program for calculating electronic structures, molecular dynamics and spectroscopic properties with DFT, coupled cluster and multi-reference methods. Binaries of ORCA can be downloaded and used free of charge by academic institutions after registration. We provide version `orca/4.1.2` using our Open MPI version `openmpi/2.1.5_mpirun`. To use ORCA on CHEOPS, you have to register on the [ORCA forum](#) and send us a private message with your request to the user RRZK-ORCA. We will then assign your account to the group `orcauser` which enables execution of the ORCA binaries. A core based job example running ORCA on 16 cores from multiple nodes could look like:

```bash
#!/bin/bash
#SBATCH --ntasks=16
#SBATCH --mem-per-cpu=4000mb
#SBATCH --time=24:00:00
#SBATCH --output=%x-%j.out
#SBATCH --account=UniKoeln

module load orca/4.1.2

export workdir=/scratch/${USER}/${SLURM_JOB_ID}
mkdir -p -m 700 $workdir

cp myorca.inp $workdir
cd $workdir
${ORCADIR}/orca myorca.inp
cd -
cp -p $workdir/*/.{gbw,loc,prop,xyz,trj,opt} .
```
Each task is hosted by 1 core and runs 1 MPI process. The batch system allocates the cores from any nodes with idle cores. Memory is allocated per core (option --mem-per-cpu meaning memory per core). The ORCA resource requests in the input file myorca.inp

```
%pal
nprocs 16 end
%maxcore 3900
```

should correspond to the SLURM resource requests: The number of processes nprocs is equal to the number of requested tasks. The estimation of maximum memory used per process maxcore in MB should be below the requested memory per core. With the input file copied, the launcher orca is executed in a job specific working directory in /scratch for faster access to temporary files (e.g. integral tables) used by ORCA. Relevant output files are transferred back to the submission directory after the run has finished.

https://orcaforum.kofo.mpg.de/app.php/portal

**Gromacs**

Gromacs is available in version gromacs/2018.6 compiled in single precision. This should be OK for most calculations. Gromacs features hybrid parallelization with both MPI and OpenMP. On a small number of cores, running Gromacs in pure MPI mode is more efficient. The following core based job example runs Gromacs with MPI on 16 cores:

```bash
#!/bin/bash
#SBATCH --ntasks=16
#SBATCH --mem-per-cpu=512mb
#SBATCH --time=24:00:00
#SBATCH --output=%x-%j.out
#SBATCH --account=UniKoeln

module load gromacs/2018.6

gmx_mpi grompp -f grompp.mdp -c conf.gro -t traj.cpt -p topol.top -o topol.tpr

srun -n $SLURM_NTASKS gmx_mpi mdrun -deffnm md_test
```

Each task is hosted by 1 core and runs 1 MPI process. The batch system allocates the cores from any nodes with idle cores. Memory is allocated per core (option --mem-per-cpu meaning memory per core). With the variable SLURM_NTASKS, you tell the MPI launcher srun to start as many MPI processes as tasks.
Alternatively, a node based job requests the same amount of resources referring to nodes:

```bash
#!/bin/bash
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=8
#SBATCH --ntasks=16
#SBATCH --mem=4gb
#SBATCH --time=24:00:00
#SBATCH --output=%x-%j.out
#SBATCH --account=UniKoeln

module load gromacs/2018.6

gmx_mpi grompp -f grompp.mdp -c conf.gro -t traj.cpt \ 
  -p topol.top -o topol.tpr

srun -n $SLURM_NTASKS gmx_mpi mdrun -deffnm md_test
```

The batch system allocates 2 nodes with 8 cores each. The 8 cores host 8 tasks running 8 MPI processes on each of the 2 nodes. Memory is allocated per node. While core based MPI jobs are scheduled earlier, node based MPI jobs run more efficiently with less wall time. Starting Gromacs in hybrid mode (MPI processes with multiple OpenMP threads each) is only worth when employing many nodes exclusively in huge jobs. Here, the number of tasks per node gives the number of MPI processes per node as before. However, a task now requires more than one core for the OpenMP threads to run on (option `--cpu-per-task` meaning cores per task). Hybrid runs on our INCA nodes (with two sockets) usually use one or two MPI processes per node. Correspondingly, each task should take all cores of the node or half of them. A job exclusively running on 64 INCA nodes with 12 cores could look like:

```bash
#!/bin/bash
#SBATCH --nodes=64
#SBATCH --ntasks-per-node=2
#SBATCH --ntasks=128
#SBATCH --cpus-per-task=6
#SBATCH --mem=1gb
#SBATCH --time=24:00:00
#SBATCH --output=%x-%j.out
#SBATCH --account=UniKoeln

module load gromacs/2018.6

gmx_mpi grompp -f grompp.mdp -c conf.gro -t traj.cpt \ 
  -p topol.top -o topol.tpr

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
srun -n $SLURM_NTASKS gmx_mpi mdrun -deffnm md_test
```

The nodes are completely occupied by 2 tasks with 6 cores each to run 2 MPI processes with 6 OpenMP threads each. The variable `SLURM_CPUS_PER_TASK` (meaning cores per task) gives the number of OpenMP threads `OMP_NUM_THREADS` to use.

http://www.gromacs.org
http://manual.gromacs.org/documentation
High-level languages for numerical computations

On CHEOPS, RRZK provides several software packages with high-level languages for numerical computations, e.g. MATLAB, Scilab and R. All packages can be started within batch jobs, all batch scripts are similar to shell scripts which might already be used on users' local workstations. For slight differences see the comments on each product.

MATLAB

MATLAB is provided as a software module which, when loaded, provides all relevant execution path entries and environment variables set correctly. RRZK has obtained some licenses of the MATLAB Parallel Computing Toolbox enabling users to speed up their applications if parallel functions like parfor, matlabpool or createTask may be used to solve a problem with parallel code execution. Even sequential jobs might benefit from MATLAB’s ability to run multi-threaded functions on multi-core architectures. Since most CHEOPS nodes provide 8-12 cores, a number of MATLAB functions might show significant speed up. A list of relevant functions may be found under

http://www.mathworks.com/support/solutions/en/data/1-4PG4AN/?solution=1-4PG4AN

Sequential MATLAB batch jobs without internal multi-threading

A simple MATLAB batch job on CHEOPS using one core may look like this

```
#!/bin/bash

#SBATCH --job-name MyMatlabProg
#SBATCH --cpus-per-task=1
#SBATCH --mem=1G
#SBATCH --time=01:00:00
#SBATCH --account=UniKoeln

module load matlab

MyMatlabProgram="$HOME/matlab/example1/myprog.m"

# start Matlab with my Matlab program
time matlab -nodisplay -nodesktop -nosplash -nojvm \
    -singleCompThread -r "run $MyMatlabProgram"
```

where the variable MyMatlabProgram refers to the location of the MATLAB program to be started within the batch job. Notice the option -singleCompThread signalling that multi-threading of all internal MATLAB functions is switched off.
As already explained earlier in this document, the above script can be submitted to the batch system with the call

```
sbatch myprog.sh
```

if `myprog.sh` is the file name of the batch script.

**Sequential MATLAB batch jobs with internal multi-threading**

As mentioned earlier sequential MATLAB jobs may benefit from internal multi-threaded functions to speed up performance of a sequential program. Some MATLAB functions (see reference above) may use all cores of a compute node, so it is necessary to obtain all processors of a node.

A MATLAB batch job on CHEOPS using 8 cores of a node supporting MATLAB's internal multi-threading features may look like this

```
#!/bin/bash
#SBATCH --job-name MyMatlabProg
#SBATCH --cpus-per-task=8
#SBATCH --mem=4G
#SBATCH --time=01:00:00
#SBATCH --account=UniKoeln

module load matlab

MyMatlabProgram="$HOME/matlab/example1/myprog.m"

# start Matlab with my Matlab program and required tasks
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

time matlab -nodisplay -nodesktop -nosplash -nojvm -r "run $MyMatlabProgram"
```

where the variable `MyMatlabProgram` refers to the location of the MATLAB program to be started within the batch job. Notice that the number of required processors has been set to 8, the variable `OMP_NUM_THREADS` has been set to the number of requested cores and that the option `-singleCompThread` has been omitted to enable internal multi-threading.

Note: While some MATLAB programs seem to speed up a little bit, in most cases it is more reasonable to check whether parts of the program like for loops can be run in parallel within MATLAB. For more information on performing parallel computations on multi-core computers see

Parallel MATLAB batch jobs

RRZK has obtained several licenses of the Parallel Computing Toolbox™ which allows offloading work from one MATLAB® session to up to 12 MATLAB sessions running in parallel, called workers. When running batch jobs with functions from the Parallel Computing Toolbox™ in such a way, only one license of MATLAB itself, the Parallel Computing Toolbox™ and each further used toolbox are checked out from the license server.

Batch jobs on CHEOPS which intend to use the Parallel Computing Toolbox™ may use the following script template:

```bash
#!/bin/bash

#SBATCH --job-name MyMatlabProg
#SBATCH --cpus-per-task=8
#SBATCH --mem=8G
#SBATCH --time=01:00:00
#SBATCH --account=UniKoeln

module load matlab

MyMatlabProgram="$HOME/matlab/parallel/myparprog.m"

# start Matlab with my Matlab program
time matlab -nodisplay -nodesktop -nosplash \ 
  -r "run $MyMatlabProgram"
```

For running parallel tasks within MATLAB, MATLAB’s embedded JAVA virtual machine is used. Therefore in opposite to sequential MATLAB jobs option -nojvm must be omitted. The JVM also needs additional memory, therefore for each requested worker/processor at least 1 GB of memory is recommended; see options in the header of the above batch script.

```bash
#SBATCH --cpus-per-task=8
#SBATCH --mem=8G
```

In the case of multi-core jobs like a MATLAB job with parallel workers, the requested wall time (see option --time=...) is the expected runtime of the whole MATLAB job. You do not need to accumulate the individual runtime of each worker; essentially wall time should decrease if running the same numerical problem with more parallel workers. On the other hand memory requirements will rise the more workers are used in parallel, thus the batch option --mem=... must be increased.

For more information on performing parallel computations with MATLAB on multi-core computers see


or contact V.Winkelmann, RRZK.
Batch Jobs without MATLAB Licenses - Using the MATLAB Compiler

Especially for running many MATLAB batch jobs simultaneously on clusters, where usually an equal number of licenses for the main MATLAB programs and equivalent licenses for MATLAB toolboxes are required, it is appropriate to compile the used MATLAB program prior to submitting the MATLAB batch jobs and start the compiled versions of the specific MATLAB programs instead of its source code versions, because in that case no licences are required, neither for MATLAB itself nor for any toolbox. A MATLAB batch job will therefore never abort due to exceeded license numbers.

At first, compile your MATLAB program interactively on the Cheops frontend.

```
module load matlab gnu
cd ${MYCODE}
mcc -m -R -nodisplay -R -nodesktop -R -nosplash \${MyMatlabProgram}.m
```

where `${MYCODE}` keeps the name of the directory of the MATLAB program to be compiled, and `${MyMatlabProgram}.m` refers to the name of the specific MATLAB program.

**Note:** The reason for compiling a MATLAB program on the cluster frontend is that after usage the compiler will be blocked for half an hour for the last user and its used compute node. Since the University of Cologne only owns one compiler license, even the last user would need luck to resubmit his/her job to the same node of the cluster used before. Using the cluster frontend bypasses this "license feature" of the MATLAB compiler.

As a consequence, it might happen that, when trying to compile a MATLAB program, the compiler license is still blocked by another user who has called the compiler less than 30 minutes ago.

After compilation a new compiled program `${MyMatlabProgram}` will exist, which may be used in a subsequent batch job. A run script for your program will be created, too, but is not required, because RRZK’s module environment includes all specifications for the MATLAB runtime environment. An appropriate batch job using the compiled MATLAB program executable would look like this:

```
#!/bin/bash -l
#SBATCH --job-name MyMatlabCompilerTest
#SBATCH --mem=4G
#SBATCH --nodes 1
#SBATCH --ntasks-per-node 1
#SBATCH --ntasks 1
#SBATCH --time 0:30:00
MYCODE=...... # to be specified
MyMatlabProgram=....... # to be specified
```
NAG Toolbox for MATLAB

RRZK has licensed NAG Toolbox for MATLAB, a large and most comprehensive single numerical toolkit derived from the well-known NAG Numerical Library. The NAG Toolbox for MATLAB contains more than 1,400 functions that provide solutions to a vast range of mathematical and statistical problems and that both complements and enhances MATLAB. NAG’s collection of numerical functions is organised into more than 40 chapters, each devoted to a mathematical or statistical area, that ease the selection of required algorithms. Each function is accompanied by documentation delivered via MATLAB’s native help system, or via the web,

http://nag.com/numeric/MB/
http://nag.com/numeric/mb/calling.asp

along with advice on selection of the best algorithm and the interpretation of the results returned.

Each NAG function has an example program to demonstrate how to access it by solving a sample problem. This template can then be easily adapted to reflect the user’s specific. The validity of each function was tested by NAG on each of the machine ranges for which the Toolbox is available.

Functions from the NAG Toolbox for MATLAB might be a lot faster than similar functions from original MATLAB toolboxes, but the NAG Toolbox for MATLAB currently is only available in a non multi-threaded (sequential) version. Instead, use the MATLAB Parallel Computing Toolbox to further speeding up your MATLAB programs.

Due to a NAG campus license the use of the NAG Toolbox for MATLAB on Cheops is unrestricted, and MATLAB programs using the NAG Toolbox for MATLAB can also be developed on personal computers within the University of Cologne. For more information on obtaining the license key for the NAG Toolbox for MATLAB contact V.Winkelmann.
**Scilab**

Scilab is an interactive platform for numerical computation providing a powerful computing environment for engineering and scientific applications using a language that is mostly compatible with MATLAB. Scilab is provided as a software module which, when loaded, provides all relevant execution path entries and environment variables set correctly. Scilab is open source software with a GPL compatible licence; therefore there are no license limits on running several Scilab jobs in parallel. Since in the current version there is no Scilab feature for programming parallel tasks within a Scilab program, only sequential Scilab batch jobs are possible. On CHEOPS a batch script for a sequential Scilab job may look like this:

```bash
#!/bin/bash
#
#SBATCH --job-name raxml-sequential
#SBATCH --output=scilab-sequential-%j.out
#SBATCH --cpus-per-task=1
#SBATCH --mem=16G
#SBATCH --time=01:00:00
#SBATCH --account=UniKoeln
module load scilab

MyScilabProgram="$HOME/scilab/example1/myprog.m"

# start Scilab with my Scilab program
time scilab -f $MyScilabProgram -nwni
```

where the variable `MyScilabProgram` refers to the location of the Scilab program to be started within the batch job. As already explained earlier in this document, the above script can be submitted to the batch system with the call

```
sbatch myprog.sh
```

if `myprog.sh` is the file name of the batch script. For more information on how to use Scilab see

[http://www.scilab.org](http://www.scilab.org)
\textit{R for Statistical Computing}

R is a programming language and free software environment for statistical computing under the terms of the GNU General Public License (GPL). On CHEOPS customized R versions built with Intel compiler and Math Kernel Library (MKL) are provided by modules. For example, loading the default R module \texttt{R/3.5.1\_intel\_mkl} will provide the known R command with many packages already installed:

\begin{verbatim}
$ module load R/3.5.1\_intel\_mkl
  MODULE : loading required module intel/17.0\_gnu\_5.1.0
...
$ R
R version 3.5.1 (2018-07-02) -- "Feather Spray"
...
> installed.packages()
\end{verbatim}

If you find all packages needed by your R program, you can start right away with your batch job. Otherwise, you should install missing packages in your home directory.

\section*{Installing additional packages}

The \textit{Comprehensive R Archive Network (CRAN)} hosts hundreds of additional packages, which are not immediately generated, packed and available with our R builds. Even worse, there might be CRAN packages interfering with each other by using same function names etc. For easier usage, several important packages have been preinstalled, namely Rcpp, parallel, Rmpi, snow, snowfall, dplyr, igraph, Rgraphviz, RSiena and others. They can be loaded by invoking \texttt{library()} within an interactive R session or an R program:

\begin{verbatim}
> library(package)
\end{verbatim}

Should necessary packages still be missing, it is possible to install additional packages by building up a private R library in your home directory. To create and fill such a library, you make an R library directory in your home directory (e.g. \texttt{$HOME/R/3.5.1$}) and set the environment variable \texttt{R\_LIBS\_USER} to its path. Then start R interactively on a login node and invoke \texttt{install.packages()} to install a package from the CRAN repository for example:

\begin{verbatim}
$ mkdir -p $HOME/R/3.5.1
$ export R\_LIBS\_USER=$HOME/R/3.5.1
$ R
R version 3.5.1 (2018-07-02) -- "Feather Spray"
...
> install.packages("package", repos="https://cran.uni-muenster.de")
\end{verbatim}
A lengthy installation protocol might occur with lots of messages including downloads of other required packages. Finally, the requested package and its dependencies are installed to the desired path given by `R_LIBS_USER`. After loading the newly built package, `help()` provides an overview of its functionality:

```
> library(package)
> help("package")
```

Packages built for a specific R version are not compatible with other versions. When changing the R version used, you need to rebuild all packages of your R library.

**Batch job running your R program**

To run an R program on CHEOPS, you need to submit a job script to the batch system SLURM (see CHEOPS Brief Instructions). For example, your job script `myprog.sh` for a sequential R run using one core only could look like this:

```
#!/bin/bash
#SBATCH --ntasks=1
#SBATCH --mem=1gb
#SBATCH --time=01:00:00
#SBATCH --output=%x-%j.out
#SBATCH --account=UniKoeln

module load R/3.5.1_intel_mkl
export R_LIBS_USER=$HOME/R/3.5.1

# R with my R program with command line arguments
R --vanilla -f myprog.R --args alg2 1000 1.0e-08
```

The requested task taking up to 1 GB of main memory may run up to 1 hour on the allocated core. After loading the R module needed, setting of `R_LIBS_USER` makes your additional packages available. Finally, R is executing your program `myprog.R` provided as argument to option `-f`. Option `--vanilla` takes care that no workspaces will be saved or restored unintentionally, nor will any user profiles or site profiles be loaded prior to execution. You may provide command line arguments to your R program following option `--args`. Please do not use the command processor `Rscript` in your R jobs because its environment differs from that of R.

**Efficiency and parallelization**

While R provides a wide variety of statistical methods with easily understandable code, it is not suitable for computationally intensive tasks. It can be more efficient executing those tasks in C++ and integrating them either with the R API or considerably easier with the Rcpp interface. As an introduction to Rcpp is not within the scope of this document, we refer you to its documentation.
Parallelization is another way to make the computation faster. R packages provide various methods with parallel workers, e.g. forking, socket communication or MPI. **Please do not use sockets for workers in your R program.** Such R jobs are not integrated with our batch system SLURM and would jam our Ethernet management network!

**Multiple workers on single node**

For R jobs running multiple workers on a single node, you should load the package parallel and use forking to set up the workers. Either you invoke multicore functions like `mclapply()` forking their workers in each call or you make a cluster of workers by forking:

```r
library(parallel)
...ntasks <- strttoi(Sys.getenv(c("SLURM_NTASKS")))
nworkers <- ntasks
cl = makeCluster(nworkers, type="FORK")
```

R reads the number of tasks allocated for the job from the SLURM environment variable `SLURM_NTASKS` and uses it to make the cluster of workers. To have multiple workers in your cluster, you have to increase the number of tasks requested in the job script:

```
#!/bin/bash -l
#SBATCH --nodes=1
#SBATCH --ntasks=8
#SBATCH --mem=8gb
#SBATCH --time=01:00:00
#SBATCH --output=%x-%j.out
#SBATCH --account=UniKoeln

module load R/3.5.1_intel_mkl
export R_LIBS_USER=$HOME/R/3.5.1

# R with my R program using forked workers on single node
R --vanilla -f myprog.R
```

Then the batch system will allocate an according number of cores on a single node to your job. As forking takes a complete copy of all data objects to the workers, you have to increase the requested memory per node with `--mem` as well.

**Multiple workers on multiple nodes**

For R jobs running multiple workers on multiple nodes, you have to load one of the packages parallel, snow or snowfall and make a cluster of MPI workers:

```r
library(parallel)
...ntasks <- strttoi(Sys.getenv(c("SLURM_NTASKS")))
nslaves <- ntasks-1
cl = makeCluster(nslaves, type="MPI")
```
As the MPI master occupies a task already, one worker less is available for the MPI cluster. In your job script, you now have to request more than one task and memory per worker with --mem-per-cpu. To initialize the MPI environment invoke the R command with the MPI launcher mpirun:

```bash
#!/bin/bash
#SBATCH --ntasks=64
#SBATCH --mem-per-cpu=1gb
#SBATCH --time=01:00:00
#SBATCH --output=%x-%j.out
#SBATCH --account=UniKoeln

module load R/3.5.1_intel_mkl
export R_LIBS_USER=$HOME/R/3.5.1

# R with my R program with MPI workers on multiple nodes
mpirun -quiet -np 1 R --vanilla -f myprog.R
```

To ease handling with MPI, we have provided customized R versions by modules with the suffix “parallel”. Those versions have adapted parameters in various parallel functions to use MPI clusters by default. For example, loading R/3.5.1_intel_mkl_parallel automatically initializes the MPI runtime environment. mpirun is not necessary anymore. You can invoke the R command as before:

```bash
#!/bin/bash
#SBATCH --ntasks=64
#SBATCH --mem-per-cpu=1gb
#SBATCH --time=01:00:00
#SBATCH --output=%x-%j.out
#SBATCH --account=UniKoeln

module load R/3.5.1_intel_mkl_parallel
export R_LIBS_USER=$HOME/R/3.5.1

# R with MPI runtime environment integrated
R --vanilla -f myprog.R
```

Moreover, function calls for a default cluster like `makeCluster(nworkers)` already provide an MPI cluster in your R program.

**Multi-threading on single node**

Some packages support parallelization with threads, e.g. when using threaded functions from MKL. In such case, parallelization is restricted to a single node again like with forking. However, now a single task utilizes more than one core by multiple threads. Therefore, you should request one task using multiple cores in your job script. Additionally, set the environment variable `OMP_NUM_THREADS` to the number of allocated cores per task:
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=4

export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}

Please note that you can run either thread or MPI parallel R programs. Using both in so-called hybrid runs is not supported since the MPI stack used is not thread-safe.

https://www.r-project.org
https://cran.r-project.org
https://www.r-project.org/nosvn/pandoc/Rcpp.html
Low-level Software for Numerical Computations

On CHEOPS, RRZK provides several low-level software packages for numerical computations like software libraries for programming languages, e.g. the NAG Numerical Library or the Math Kernel Library (MKL).

NAG Numerical Library

RRZK has licensed the well-known NAG Numerical Library, a programming library with a large collection of routines covering a wide range of mathematical and statistical areas including local/global optimization, linear/quadratic/integer/nonlinear programming, least squares problems, ordinary/partial differential equations and mesh generation, curve/surface fitting and interpolation, solution of dense/banded/sparse linear equations, Eigen value problems, random number generation, correlation and regression analysis, time series analysis.

The NAG Numerical Library contains more than 1,600 functions that are organised into chapters, each devoted to a mathematical or statistical area, that ease the selection of required algorithms. Each function is accompanied by documentation along with advice on selection of the best algorithm for your purposes and interpretation of the results returned.

The NAG Numerical Library has APIs for different programming languages and/or compilers. The following versions of the NAG Numerical Library are available on CHEOPS and corresponding modules environments exist:

- NAG Fortran 77 Library (Intel, PGI, GNU)
- NAG C Library (Intel, PGI, GNU)
- NAG Fortran 90 Library (Intel)
- NAG Fortran 77 Library for Multicore & SMP (Intel, PGI)
- NAG Fortran 77 Parallel Library (Intel MPI version)

For more information on the available algorithms start from

http://www.nag.com/support_documentation.asp

and search the documentation of the required API. For more information on the CHEOPS environment modules for the NAG Numerical Library and the linking options see the document Cheops Brief Instructions.

Sample batch job for the NAG Fortran 77 Library

The following batch job demonstrates the use of the NAG Fortran 77 Library routine G05PJF, which generates a realization of a multivariate time series from a vector autoregressive moving average (VARMA) model:
#!/bin/bash
#SBATCH --cpus-per-task=1
#SBATCH --mem=1024mb
#SBATCH --time=00:05:00
#SBATCH --account=UniKoeln
#
module load nag/f_24
#
get test program for NAG routine G05PJF
# Please note: $NAGHOME is RRZK specific!
#
cp $NAGHOME/examples/source/g05pjfe.f90 .
cp $NAGHOME/examples/data/g05pjfe.d .

ifort g05pjfe.f90 -o g05pjfe -lnag_nag -lpthread
./g05pjfe < g05pjfe.d

The corresponding sample program and its input data are copied from the local examples folder on Cheops after the environment for the NAG Fortran 90 Library is initialized, the sample Fortran program is compiled by the Intel Fortran compiler and finally the executable is started, reading its data from the provided input file.
For information on the module names and the linking options concerning the NAG Numerical Library for all supported languages and compilers see the document Cheops Brief Instructions.

**Math Kernel Library (MKL)**

The MKL (Math Kernel Library) includes the functionality of a lot of other programming libraries (BLAS, LAPACK, FFT, Sparse BLAS) addressing mathematical operations and algorithms often needed in scientific computing. For more information on MKL see the document Cheops Brief Instructions.
Applications for bioinformatic computations

On CHEOPS, RRZK provides several software packages for bioinformatic computations like phylogenetic analysis or sequence analysis. All packages can be started within batch jobs, all batch scripts are similar to shell scripts which might already be used on users' local workstations. For slight differences see the comments on each product.

RAxML

RAxML (Randomized Axelerated Maximum Likelihood) is a program for sequential and parallel Maximum Likelihood based inference of large phylogenetetic trees. It has originally been derived from fastDNAml which in turn was derived from Joe Felsentein's dnaml being part of the PHYLIP package.

An older user manual of RaxML may be found under


while the source can be investigated under

https://github.com/stamatak/standard-RAxML

RaxML exists in 4 versions, one is a pure sequential program lacking of any parallel code, another is parallelized for the Message Passing Interface (MPI), a further on is a multi-threaded version that uses pthreads to run RaxML in parallel on one cluster node, and the last one is a hybrid version of the pthread and MPI parallelized version, speeding up RaxML significantly in many cases.

RaxML is provided as a software module which, when loaded, provides all relevant execution path entries and environment variables being set correctly. See the following table for the names of the corresponding RaxML executables:

<table>
<thead>
<tr>
<th>Version</th>
<th>Program Name on Cheops</th>
</tr>
</thead>
<tbody>
<tr>
<td>single-thread</td>
<td>raxmlHPC</td>
</tr>
<tr>
<td>multi-thread, one node</td>
<td>raxmlHPC-PTHREADS</td>
</tr>
<tr>
<td>MPI</td>
<td>raxmlHPC-MPI</td>
</tr>
<tr>
<td>Hybrid, multi-threaded, multi-nodes</td>
<td>raxml-HYBRID</td>
</tr>
</tbody>
</table>
Sequential RAxML batch jobs without multi-threading

A simple (single threaded) RAxML batch job on CHEOPS using one processor core may look like this

```bash
#!/bin/bash

#SBATCH --job-name raxml-sequential
#SBATCH --output=raxml-sequential-%j.out
#SBATCH --cpus-per-task=1
#SBATCH --mem=200mb
#SBATCH --time=01:00:00
#SBATCH --account=UniKoeln

module load raxml

RAXML=raxmlHPC

DATADIR=/opt/rrzk/software/raxml/RRZK/data
INPUT=$DATADIR/Cryothecomonas.phylip
OUTPUT=Cryothecomonas.sequential

time $RAXML -f a -x 12345 -p 12345 -N 100 -m GTRGAMMA \ 
   -s $INPUT -n $OUTPUT
```

where the variable `RAXML` refers to the location of the used RaxML executable (here the single threaded version) to be started within the batch job, `INPUT` specifies the input file to be analysed and `OUTPUT` defines the base name of the generated output files.

As already explained earlier in this document, the above script can be submitted to the batch system with the call

```
sbatch raxml-sequential.sh
```

if `raxml-sequential` is the file name of the batch script.

Parallel RAxML batch jobs with multi-threading

A parallel RAxML batch job on CHEOPS using several processors on one computing node may look like this:

```bash
#!/bin/bash

#SBATCH --job-name=raxml-pthreads
#SBATCH --output=raxml-pthreads-%j.out
#SBATCH --cpus-per-task=4
#SBATCH --mem=800mb
#SBATCH --time=01:00:00
#SBATCH --account=UniKoeln

module load raxml
```
RAXML=raxmlHPC-PTHREADS

DATADIR=/opt/rrzk/software/raxml/RRZK/data
INPUT=$DATADIR/Cryothecomonas.phylip
OUTPUT=Cryothecomonas.pthreads

time $RAXML -f a -x 12345 -p 12345 -N 100 -T 4 -m GTRGAMMA \ -s $INPUT -n $OUTPUT

where in the example the batch option --cpus-per-task=4 requests 4 processor cores, setting the shell variable RAXML=raxmlHPC-PTHREADS forces the script to use the multi-threaded version of RAxML, and finally the RaxML option -T 4 instructs RAxML to use 4 parallel threads during execution.

In the case of multi-core jobs just like a batch job with parallel RAxML threads, the requested wall time is the expected runtime of the whole RAxML job (see batch option --time=...). You do not need to accumulate the individual runtime of each thread; essentially wall time should decrease if running the same job with more parallel threads.

On the other hand memory requirements will rise the more threads are used in parallel, thus the batch option --mem=... must be increased.

As already explained earlier in this document, the above script can be submitted to the batch system with the call

```
sbatch raxml-pthreads.sh
```

if raxml-pthreads.sh is the file name of the batch script.

**MPI parallelized RAxML batch jobs on several computing nodes**

A MPI-parallelized RAxML version exists which can run more than one RAxML process in parallel on several computing nodes via MPI (Message Passing Interface) and which allows performing parallel bootstraps, rapid parallel bootstraps, or multiple inferences on the original alignment. However, this version of RAxML is not multi-threaded and allows only distributing single threaded RAxML tasks over the requested number of nodes. This means that all communication is done via MPI, whose latency is larger than in the pthreads case. Because RAxML also provides hybrid version merging MPI and PThreads, we suggest using that hybrid version rather than the pure MPI version of RAxML.

**Hybrid parallelized RAxML batch jobs on several computing nodes**

A hybrid parallelized RAxML version exists which can run more than one RAxML process in parallel on several computing nodes via MPI (Message Passing Interface) and allows multi-threading within the cores of each requested node, thus reducing time for communication between the RAxML tasks within a single node.

A RAxML batch job on CHEOPS using 4 computing nodes with 8 processors each may look like this:
#!/bin/bash

#SBATCH --job-name=raxml-hybrid
#SBATCH --output=raxml-hybrid-%j.out
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=8
#SBATCH --mem=22GB
#SBATCH --time=01:00:00
#SBATCH --account=UniKoeln

module load raxml

RAXML=raxmlHPC-HYBRID

DATADIR=/opt/rrzk/software/raxml/RRZK/data
INPUT=$DATADIR/Cryothecomonas.phylip
OUTPUT=Cryothecomonas.hybrid-2-1-8

time srun -n 2 \
    $RAXML -f a -x 12345 -p 12345 -N 100 -T 8 \n    -m GTRGAMMA -s $INPUT -n $OUTPUT

where in the example the batch options

#SBATCH --nodes=2
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=8

ask for 2 computing nodes with 1 RAxML process started on each node, requesting 8 processor cores for each RAxML task (thus using 16 processors in total!). Setting the shell variable RAXML=raxmlHPC-HYBRID forces the script to use the hybrid parallelized version of RAxML. To benefit from the 2 requested nodes and their processors, RAxML is started via srun, taking care that only one RAxML process is started on each node and that each process uses 8 PThreads within a node.

A note from the RAxML authors: The current MPI-version only works properly on several computing nodes if you specify the option --N in the command line, since this option has been designed to do multiple inferences or rapid/standard BS searches in parallel! For all remaining options, the usage of this type of coarse-grained parallelism does not make much sense! The MPI-version is for executing really large production runs (i.e. 100 or 1,000 bootstraps) on a LINUXcluster. You can also perform multiple inferences on larger datasets in parallel to find a best-known ML tree for your dataset. Finally, the novel rapid BS algorithm and the associated ML search have also been parallelized with MPI.

In the case of multi-core jobs just like a batch job with parallel RAxML threads, the requested wall time is the expected runtime of the whole RAxML job (see batch option --time=...).
You do not need to accumulate the individual runtime of each thread; essentially wall time should decrease if running the same job with more parallel threads. On the other hand memory requirements will rise the more threads are used in parallel, thus the batch option --mem=... must be increased. As already explained earlier in this document, the above script can be submitted to the batch system with the call

```
sbatch raxml-hybrid.sh
```

if raxml-hybrid.sh is the file name of the batch script.

### Hybrid parallelized RAxML batch jobs on several computing nodes (extd.)

The experienced user can run the hybrid parallelized version of RAxML in an advanced manner if the special hardware architecture of the computing nodes of the cluster Cheops is respected. Each Cheops computing node consists of two Intel Nehalem Quadcore CPUs (that’s 8 processors overall per node). Experiments show that if one starts one RAxML process with 4 threads on each Quadcore processor (that means 2 RAxML processes, each with 4 threads on one node, and not only one RAxML process with 8 threads as before), performance may increase up to 10%. A corresponding batch job of the example above would read:

```
#!/bin/bash

#SBATCH --job-name=raxml-hybrid
#SBATCH --output=raxml-hybrid-%j.out
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=2
#SBATCH --cpus-per-task=4
#SBATCH --mem=22GB
#SBATCH --time=01:00:00
#SBATCH --account=UniKoeln

module load raxml

RAXML=raxmlHPC-HYBRID

DATADIR=/opt/rrzk/software/raxml/RRZK/data
INPUT=$DATADIR/Cryothecomonas.phylip
OUTPUT=Cryothecomonas.hybrid

time srun -n 4 \ 
  $RAXML -f a -x 12345 -p 12345 -N 100 -T 4 \ 
  -m GTRGAMMA -s $INPUT -n $OUTPUT
```

where in the example the batch options

```
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=2
#SBATCH --cpus-per-task=4
```
MrBayes

MrBayes is a program for the Bayesian estimation of phylogeny. Bayesian inference of phylogeny is based upon a quantity called the posterior probability distribution of trees, which is the probability of a tree conditioned on the observations. The conditioning is accomplished using Bayes's theorem. The posterior probability distribution of trees is impossible to calculate analytically; instead, MrBayes uses a simulation technique called Markov chain Monte Carlo (or MCMC) to approximate the posterior probabilities of trees. More information on MrBayes may be found under

http://mrbayes.csit.fsu.edu/

On Cheops, MrBayes takes advantage of the fact that Metropolis coupling or heating is well suited for parallelization, and MrBayes uses MPI to distribute heated and cold chains among available processors. The maximum number of processors suitable for MrBayes corresponds to the product of the number of chains (nchains) and the number of simultaneous analyses (nruns) as specified in the MrBayes block at the end of the Nexus file.

MrBayes is provided as a software module which, when loaded, provides all relevant execution path entries and environment variables being set correctly.

A simple MrBayes batch job on CHEOPS using a standard configuration with 2 analyses of 4 chains each may look like this

```
#!/bin/bash

#SBATCH --job-name MyMrBayes
#SBATCH --output MrBayes-test1.nxs-%j.out
#SBATCH --mem=10G
#SBATCH --nodes 1
#SBATCH --ntasks-per-node 8
#SBATCH --cpus-per-task 1
#SBATCH --time 1:00:00
#SBATCH --account=UniKoeln

module load mrbayes

DATADIR=/opt/rrzk/software/mrbayes/RRZK/data
INPUT=test1.nxs
OUTPUT=${INPUT}.${SLURM_JOB_ID}.log

time srun -n $SLURM_NTASKS mb $DATADIR/$INPUT > $OUTPUT
```
where the batch system is asked to provide 1 computing node for 8 tasks (8 processor cores) for this job (remember: we want to compute 2 analyses a 4 chains each); a memory limit of 10 GB over all cores is requested, and an expected runtime (walltime) of 1 hour.

If the whole number of chains, e.g. the product nchains by nruns, exceeds 8 you can increase the number of nodes in order to provide more processors to MrBayes and to speed up your program. Note that most nodes on Cheops only provide 8-12 processors; specifying more than 12 tasks per node will reduce your job’s priority due to the less availability of nodes with more than 12 processors. MrBayes cannot use more than nchains x nruns processors, therefore do not request more nodes/tasks as really needed!

In the case of multi-node jobs the requested wall time is the expected runtime of the whole MrBayes job (see batch option --time=...).

You do not need to accumulate the individual runtime of each node; essentially wall time should decrease if running the same job and having enough chains to be distributed on the requested nodes/tasks.

In the above job all output files are written to the working directory where the job was sent from. Since MrBayes creates several output files, it is recommended to use a separate working directory for each MrBayes batch job to avoid mixing output of different jobs.

Finally, the MrBayes executable mb is started via the MPI command srun which takes care that mb is started in parallel, using test1.nxs as input file while the log of MrBayes is directed into the logfile test1.nxs.<jobid>.log of the current working directory.

As already explained earlier in this document, the above script can be submitted to the batch system with the call

```
sbatch myprog.sh
```

if myprog.sh is the file name of the batch script.

---

**PhyloBayes-MPI**

PhyloBayes-MPI is a Bayesian Markov chain Monte Carlo (MCMC) sampler for phylogenetic inference exploiting a message-passing-interface system for multi-core computing. The program will perform phylogenetic reconstruction using either nucleotide, protein, or codon sequence alignments. Compared to other phylogenetic MCMC samplers, the main distinguishing feature of PhyloBayes is the use of non-parametric methods for modelling among-site variation in nucleotide or amino-acid propensities. More information on PhyloBayes may be found under

```
http://phylobayes.org
```

A run of the MCMC sampler program pb_mpi will produce a series of points drawn from the posterior distribution over the parameters of the model. Each point defines a detailed model configuration (tree topology, branch lengths, nucleotide or amino-acid profiles of the mixture, etc.). The series of points defines a chain.

On Cheops, PhyloBayes is provided as a software module. Since pb_mpi is parallelized, several processor cores or even nodes can be used for running chains in order to speed up processing. A simple PhyloBayes batch job on CHEOPS may look like this:
#!/bin/bash

#SBATCH --job-name MyPhyloBayes
#SBATCH --output PhyloBayes-brpo-%j.out
#SBATCH --mem=4G
#SBATCH --nodes 1
#SBATCH --ntasks-per-node 8
#SBATCH --time 01:00:00
#SBATCH --account=UniKoeln

module load phylobayes

DATADIR=$PBMPI_HOME/data/brpo
INPUT=brpo.ali
CHAINNAME=brpo.chain.$SLURM_JOB_ID

time srun -n $SLURM_NTASKS pb_mpi \
    -d $DATADIR/$INPUT -cat -gtr $CHAINNAME

where the batch system is asked to provide 1 computing node for 8 tasks (8 processor cores) for this job; a memory limit of 4 GB over all cores is requested, and an expected runtime (wall time) of 1 hour. In the case of multi-node jobs the requested wall time is the expected runtime of the whole PhyloBayes job (see batch option --time=...).

For multi-node jobs you do not need to accumulate the individual runtime of each node. Usually pbmpi runs until the specified wall time is exceeded and aborts thereafter. For estimating the required memory and runtime, please read the PhyloBayes manual.

In the above job all output files are written to the working directory from where the job was sent to the batch system. Since PhyloBayes creates several output files, it is recommended to use a separate working directory for each PhyloBayes batch job to avoid mixing output of different jobs.

Finally, the PhyloBayes executable pbmpi is started via the MPI command srun which takes care that pbmpi is started in parallel, using brpo.ali as input file.

As already explained earlier in this document, the above script can be submitted to the batch system with the call

```
sbatch myprog.sh
```

if myprog.sh is the file name of the batch script.