9 Bedienungshinweise zum HPC-Hilfssystem SLURM

Die Berechnung der komplexen Schaumstoffzellmodelle als Einheitszellen mit periodischen Randbedingungen übersteigt ist mit üblichen Computersystemen nur eingeschränkt möglich. Um die Berechnungszeiten zu reduzieren, wurden besonders aufwendige Simulationen unter Berücksichtigung innerer Kontaktbedingungen am Zentrum für Informationsdienste und Hochleistungsrechnen (ZIH) der TU Dresden auf dem Bull HPC-Cluster „Taurus“ durchgeführt. Für die Lastverteilung kommt auf diesem Hochleistungsrechnen (engl. High Performance Computing) (HPC)-System die Hilfssoftware Slurm, ein sogenanntes Batch-System zum Einsatz, das die Berechnungsabfragen auf die vorhandenen Ressourcen verteilt und diese so möglichst effizient und gleichmäßig auslastet. Das Betriebssystem des Clusters ist UNIX, weshalb auch einige grundsätzliche LINUX-Befehle für die Bedienung erforderlich sind.

Um Zugang zu diesen Ressourcen zu erlangen, kann mit einem gültigen ZIH-Login über die Website `https://hpcprojekte.zih.tu-dresden.de/` ein Antrag gestellt werden. Projekte unterhalb von 40 000 CPUh gelten als Kleinstprojekte und werden unkompliziert berücksichtigt. Nach der Freischaltung erhält man eine Mail mit einem Account-Namen. Um sich mit den Systemen vertraut zu machen ist das HPC-Kompendium des ZIH sehr gut geeignet.

`https://doc.zih.tu-dresden.de/hpc-wiki/bin/view/Compendium`

Ein Zugang zu Taurus erfolgt von einem Windows-PC vorteilhaft über einen SecureShell Client (SSH), wie beispielsweise MobaXterm oder Putty. Unter der Adresse `taurus.hrsk.tu-dresden.de` erreicht man mit diesem SSH dann Taurus, um Rechnungen zu starten oder Daten zu transferieren. Das eigene Home-Verzeichnis ist das Verzeichnis ist das Startverzeichnis bei einem Verbindungsaufbau und befindet sich unter `/home/<zih-login>`. Im Projektverzeichnis `/projects/<mein-account>` können Eingangsdaten und Ergebnisse gespeichert werden. Als Arbeitsverzeichnis für Simulationen, bzw. Rechnungen steht der Ordner `/scratch/<mein-account>` zur Verfügung. Dieses Scratch-Verzeichnis wird regelmäßig automatisch bereinigt, weshalb er nicht zu längerfristigen Dateiablage genutzt werden sollte. Die Abaqus Input-Datei `<mein-job.inp>` Um sich in der Verzeichnisstruktur auf dem LINUX-System zu bewegen und Programme zu starten wird in der Regel ein Kommandozeilenfenster genutzt – hier werden Kommandos einzeln eingegeben. Wichtige Kommandos zur Navigation sind die Folgenden.

`pwd`	Pfad erfragen – Wo bin ich?
`cd <Unterverzeichnis>`	In ein Unterverzeichnis wechseln
`cd ..`	In das übergeordnete Verzeichnis wechseln
`top -u <zih-login>`	Welche Prozesse laufen für einen User?
`CTRL+C`	Prozess im laufenden Fenster abbrechen
`squeue -u <zih-login>`	Warteschlange für einen User anzeigen
`scontrol -d show job <job-id>`	Status zu einem laufenden Job
`scancel <job-id>`	Laufenden Job abbrechen
`. /etc/profile`	Nutzerprofil im aktiven Kommandofenster laden

Um einen Job mit ABAQUS zu starten sind eine Reihe von Befehlen nötig, die die erforderliche Anzahl an CPUs, Speicherplatz und Rechenzeit anfordern und Pfade und Dateinamen vorgeben. Aufgrund der vielen Einzelbefehle empfiehlt es sich hierfür eine

Datei anzulegen, die die Befehle enthält und diese automatisiert nacheinander an das Betriebssystem Linux und das Hilfssystem Slurm sendet. Solche Stapelverarbeitungsdateien heißen auch Batch- oder Bash-Dateien. Zur Erstellung einer Bash-Datei kann der sogenannte „Slurmgenerator“ genutzt werden, der über die folgende Website verfügbar ist.

`https://doc.zih.tu-dresden.de/hpc-wiki/bin/view/Compendium/Slurmgenerator`

Allerdings werden hier für Abaqus erforderliche spezifische Befehle nicht automatisch integriert. Das folgende Bash-Skript gibt eine gute Ausgangsbasis zur Anpassung an die eigenen Anforderungen.

```
#!/bin/bash

### Abaqus JobStarter - Submit this script with: sbatch <job-starter.sh>
### Optimized for parallel jobs
#SBATCH --account=<mein-account>   # account name for slurm
#SBATCH --job-name=<mein-job>   # job name for slurm
#SBATCH --output=/projects/<mein-account>/<mein-job>-%j.out   # file for output messages
#SBATCH --error=/projects/<mein-account>/<mein-job>-%j.err   # file for error messages
#SBATCH --ntasks=<meine-cpu-anzahl>   # number of processor cores (CPUs) (i.e. tasks)
#SBATCH --gres=gpu:<meine-gpu-anzahl>   # reserve graphic processors (GPUs)
#SBATCH --mem-per-cpu=2500M   # memory per CPU core
#SBATCH --time=120:00:00   # walltime e.g. 120h
#SBATCH --mail-user=<meine-mail>   # email address
#SBATCH --mail-type=END   # send a mail when job ends
#SBATCH --mail-type=FAIL   # send a mail when job fails
#SBATCH --profile=all   # create energy and task profiling data
#SBATCH --acctg-freq=5,task=5,energy=5   # define sampling interval for profiling data (if
     needed)

### Enable debug mode
#set -x   # remove '#' to enable debug mode

### Define jobname and project directory
MYACCOUNT=<mein-account>
MYJOB=<mein-job>

### Make scratch directory for job
mkdir -p /scratch/$MYACCOUNT/$MYJOB   # make directory if does not exist

###  The files will be copied to a scratch directory (uncomment if you have large data sets).
cp $MYJOB.inp /scratch/$MYACCOUNT/$MYJOB   # copy all .inp files
cd /scratch/$MYACCOUNT/$MYJOB   # change directory to the directory where you want to start
     your job.

###  Load the module
module load abaqus/6.13   # Load Abaqus

### Set the max number of threads to use for programs using OpenMP. Should be <= ppn. Does
     nothing if the program doesn't use OpenMP.
export OMP_NUM_THREADS=$SLURM_CPUS_ON_NODE

###  Run ABAQUS
abaqus \
        job=$MYJOB \
        cpus=<meine-cpu-anzahl> gpus=<meine-gpu-anzahl> \
        -verbose 1 standard_parallel=all mp_mode=mpi \
        interactive

###  Merge the profiling files for every node in /lustre/scratch2/profiling/${USER} (if
     profiling activated)
sh5util -j ${SLURM_JOBID} -o /projects/$MYACCOUNT/$MYJOB-%j.h5

###  Transferring the results to the home directory ($HOME)
###  Uncomment if you have large data sets
mkdir -p /projects/$MYACCOUNT/$MYJOB   # make directory if it does not exist
cp -pr /scratch/$MYACCOUNT/$MYJOB /projects/$MYACCOUNT   # copy data  with subdirectories

###  View profiled data (if profiling activated)
module load hdf5/hdfview   # Load module for profiling ressource utilization
hdfview.sh /projects/$MYACCOUNT/$MYJOB-%j.h5   # load HDF viewer to visualize profiled data

### Disable debug mode
#set +x # remove '#' to disable debug mode when you enabled it before
exit 0
```

Alle `<Platzhalter>` in der Bash-Datei müssen durch eigene Angaben ersetzt werden. Wichtig unter Windows ist, dass zum Speichern dieses Skripts ein Editor verwendet wird, der die Datei mit UNIX-Zeilenenden (LF) speichern kann, beispielweise Notepad++. Stan-

dardendung für Bash-Dateien ist `.sh`. Für die in dieser Arbeit durchgeführten Rechnungen wurden folgende Werte verwendet.

```
<mein-account> = p_schaumstoff
<mein-job> = P42a_wet_15
<meine-cpu-anzahl> = 24
<meine-gpu-anzahl> = 1
<meine-email> = eike.dohmen@tu-dresden.de
```

Zeichen nach # gelten als auskommentiert und werden nicht interpretiert. Ausnahme sind hier die Slurm-Befehle #SBATCH, die das Hilfssystem konfigurieren und die Ressourcen reservieren. Die in dieser Bash-Datei vorgesehenen Befehle für ein Profiling, d.h. eine Überwachung des Ressourcen- und Energiebedarfs, sind für Abaqus leider bisher ohne Wirkung, da Abaqus die Parallelisierung einer Rechnung über ein eigenes Datenverarbeitungsinterface (engl. MPI - Message Passing Interface) koordiniert und Slurm so keinen Zugriff auf die CPU Daten sowie Speicherauslastung und Energieverbrauch hat.

Bei der Reservierung von Prozessoren und Speicher sind noch zwei wesentliche Grenzen zu beachten. Einerseits die Ressourcen des Clusters und deren sinnvolle Nutzung. Andererseits die verfügbaren Lizenzen für die genutzte Software. Hat ein Kern auf dem Taurus-Cluster 24 CPUs, würde eine Nutzung von 26 CPUs kaum einen Leistungszuwachs, sondern eher eine Verlangsamung verursachen, da Daten von einem Knoten zu einem anderen übertragen werden müssten. Die Kommunikation zwischen Knoten ist deutlich langsamer, als die Kommunikation in einem Knoten. Zudem gilt für GPUs, dass diese nicht alleine zu einem Knoten gehören, sondern beispielsweise immer zwei GPUs mit 16 CPUs einen Knoten bilden. Um eine sinnvolle Auslastung der Ressourcen zu erzwingen, kann es beispielsweise eine Vorgabe des verwaltenden Rechenzentrums sein, dass man unabhängig von der angeforderten Anzahl an CPUs und GPUs immer einen vollen Knoten zugeordnet bekommt. Das hieße bei Start einer zweistündigen Rechnung unter Anforderung von einer GPU und 10 CPUs, würden trotzdem 2 GPUs und 16 CPUs für die Kalkulation der Rechenzeit herangezogen, also 36 CPUh. Eine gute Planung der angeforderten Ressourcen ist daher sinnvoll. Die Verfügbaren Lizenzen müssen ebenso beachtet werden. Derzeit gibt es 20 Lizenzen, auch Tokens genannt, für Abaqus Standard (mit 19 Tokens können 24 CPUs genutzt werden) und 1 GPGPU-Lizenz für die Nutzung von einer GPU. Die Berechnung der erforderlichen Lizenzen erfolgt dabei nach der folgenden Formel.

$$\left\lceil n_{\mathrm{GPU}} + 5 \cdot n_{\mathrm{CPU}}^{0{,}422} \right\rceil \tag{13}$$

Unter der Internetseite `http://deviceanalytics.com/abaqus-token-calculator` kann die Tokenanzahl einfach durch Eingabe der CPU-Anzahl berechnet werden.

Die Nutzung von GPUs kann insbesondere bei großen Berechnungsmodellen mit deutlich über 500 000 Gleichungen vorteilhaft sein. Bei impliziten Berechnungsmodellen kann mittels einer GPU vor allem die Transformation der aufwendigen Steifigkeitsmatrix rasant erfolgen, da GPUs verhältnismäßig viel Arbeitsspeicher (bspw. 25 GB) mit sehr hohe interne Datenraten kombinieren. Der Flaschenhals bei Einsatz einer GPU ist jedoch die Kommunikation mit den CPUs, weshalb sich der Einsatz erst für aufwendige Berechnungen lohnt.