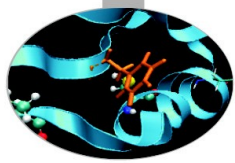
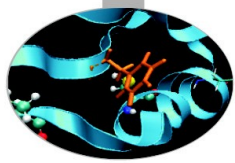
This afternoon you will learn...



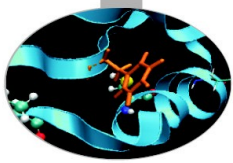
- How to set up a protein topology using GROMACS
- How to prepare your submission script
- How to submit your job to the PBS queueing system on Eurora
- Benchmarks: GPU vs. IntelPhi

Eurora job script template



```
#!/bin/bash
#PBS -l walltime=1:00:00
#PBS -l select=1:ncpus=12:mpiprocs=12:ngpus=2:mem=47GB
#PBS -o job.out
#PBS -e job.err
#PBS -q parallel
#PBS -A <account_no>
#PBS -m mail_events ==> specify email notification
                        (a=aborted,b=begin,e=end,n=no_mail)
#PBS -M user@email.com

cd $PBS_O_WORKDIR
module load autoload openmpi
module load somelibrary

mpirun ./myprogram < myinput
```

PBS commands



qsub

qsub <job_script>

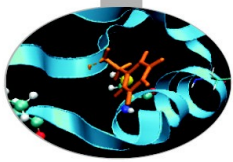
Your job will be submitted to the PBS scheduler and executed when there will be nodes available (according to your priority and the queue you requested)

qstat

qstat

Shows the list of all your scheduled jobs, along with their status (idle, running, closing, …) Also, shows you the job id required for other qstat options

PBS commands



qstat

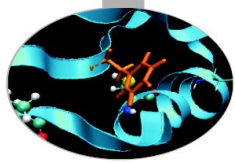
qstat -f <job_id>

Provides a long list of informations for the job requested.
In particular, if your job isn't running yet, you'll be notified about its estimated start time or, if you made an error on the job script, you will learn that the job won't ever start

qdel

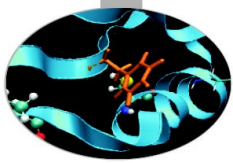
qdel <job_id>

Removes the job from the scheduled jobs by killing it



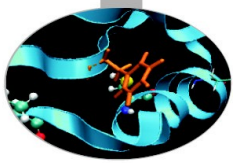
Scripts for running MD codes on PLX/Eurora

Gromacs 5.0.4, pure MPI on Eurora



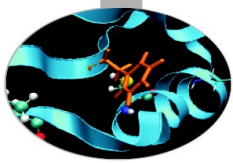
```
#!/bin/bash
#PBS -N gmx
#PBS -l select=1:ncpus=16:mpiprocs=16:mem=14GB
#PBS -q parallel
#PBS -l walltime=1:00:00
#PBS -A train_cmd22015
#PBS -W group_list=train_cmd22015

cd $PBS_O_WORKDIR                    ==> change to current dir

module load profile/advanced
module load autoload gromacs/5.0.4

export OMP_NUM_THREADS=1             ==> set nr. Of OpenMP threads per MPI proc to1

mdrun=$(which mdrun_mpi)
cmd="$mdrun -s topol.tpr -v -maxh 1.0 -nb cpu"
mpirun -np 16 $cmd
```

Gromacs 5.0.4 MPI+CUDA on Eurora



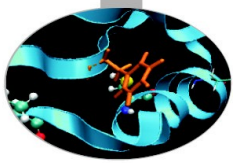
```
#!/bin/bash
#PBS -N gmx
#PBS -l select=1:ncpus=16:mpiprocs=16:mem=14GB
#PBS -q parallel
#PBS -l walltime=1:00:00
#PBS -A train_cmd22015
#PBS -W group_list=train_cmd22015

cd $PBS_O_WORKDIR                    ==> change to current dir

module load profile/advanced
module load autoload gromacs/5.0.4

export OMP_NUM_THREADS=1             ==> set nr. Of OpenMP threads to 1
#                                    ==> set total mpi tasks = 2 and bind to two GPUs

mdrun=$(which mdrun_mpi_cuda)
cmd="$mdrun -s topol.tpr -v -maxh 1.0 -gpu_id 01 "
mpirun -np 2 $cmd
```

NAMD 2.10 MPI+CUDA on Eurora



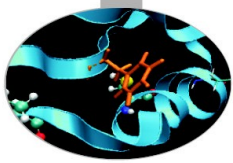
```
#!/bin/bash
#PBS -N gmx
#PBS -l select=1:ncpus=16:mpiprocs=16:mem=14GB
#PBS -q parallel
#PBS -l walltime=1:00:00
#PBS -A train_cmd22015
#PBS -W group_list=train_cmd22015
cd $PBS_O_WORKDIR                    ==> change to current dir

module load profile/advanced
module load autoload namd/2.10

namd=$(which namd2_cuda)             ==> set path to namd executable

mpirun $namd +idlepoll md.namd       ==> run CUDA version of NAMD
```

NAMD 2.10 Intel Phi (offload mode) on Eurora



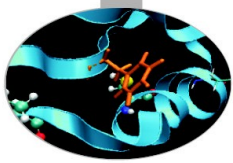
```
#!/bin/bash
#PBS -N gmx
#PBS -l select=1:ncpus=16:mpiprocs=16:mem=14GB
#PBS -q parallel
#PBS -l walltime=1:00:00
#PBS -A train_cmd22015
#PBS -W group_list=train_cmd22015
cd $PBS_O_WORKDIR                    ==> change to current dir

module load profile/advanced
module load autoload namd/2.10

namd=$(which namd2_cuda)             ==> set path to namd executable

mpirun $namd +idlepoll md.namd       ==> run CUDA version of NAMD
```

Amber-14 on Eurora (pure MPI version)



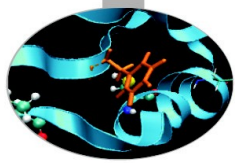
```
#!/bin/bash
#PBS -N gmx
#PBS -l select=1:ncpus=16:mpiprocs=16:mem=14GB
#PBS -q parallel
#PBS -l walltime=1:00:00
#PBS -A train_cmd22015
#PBS -W group_list=train_cmd22015

cd $PBS_O_WORKDIR                    ==> change to current dir

Module load profile/advanced
module load autoload amber/14

cmd="pmemd.MPI -O -i mdin -o mdout -p prmtop -c inpcrd -r restrt -x mdcrd"
mpirun -np 16 $cmd
```

Amber 14 – MPI+CUDA version



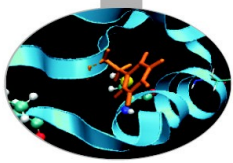
```
#!/bin/bash
#PBS -N gmx
#PBS -l select=1:ncpus=16:mpiprocs=16:mem=14GB
#PBS -q parallel
#PBS -l walltime=1:00:00
#PBS -A train_cmd22015
#PBS -W group_list=train_cmd22015

cd $PBS_O_WORKDIR
module load autoload amber/14

# for best performance use 1 mpi task/1 gpu.  In this example we have 1*2 gpus = 2 MPI tasks.

cmd="pmemd.cuda.MPI -O -i mdin -o mdout -p prmtop -c inpcrd -r restrt -x mdcrd"

mpirun -np 2 $cmd
```

Amber 14 – Intel Phi
(offload mode)



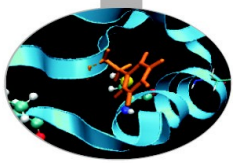
```
#!/bin/bash
#PBS -N gmx
#PBS -l select=1:ncpus=16:mpiprocs=16:nmics=1:mem=14GB
#PBS -q parallel
#PBS -l walltime=1:00:00
#PBS -A train_cmd22015
#PBS -W group_list=train_cmd22015

cd $PBS_O_WORKDIR
module load autoload amber/14
exe=$AMBER_HOME/bin/pmemd.mic_offload.MPI
source $INTEL_HOME/bin/compilervars.sh intel64

 mpirun -np 16 $exe -O -i mdin.CPU -o mdout-offload -p prmtop -c inpcrd
```

Amber 14 – Intel Phi
(offload mode)



```
#!/bin/bash
#PBS -N gmx
#PBS -l select=1:ncpus=16:mpiprocs=16:mem=14GB
#PBS -q parallel
#PBS -l walltime=1:00:00
#PBS -A train_cmd22015
#PBS -W group_list=train_cmd22015

cd $PBS_O_WORKDIR
module load autoload amber/14
exe=$AMBER_HOME/bin/pmemd.mic_offload.MPI
source $INTEL_HOME/bin/compilervars.sh intel64

export MIC_ENV_PREFIX=PHI
export OMP_NUM_THREADS=1

mpirun -n 7 $exe -O -i mdin.CPU -o mdout-offload -p prmtop -c inpcrd \
 : -n 1 -env PHI_KMP_PLACE_THREADS="30c,4t,0O" -env PHI_KMP_AFFINITY="scatter" -env
PHI_OMP_NUM_THREADS=30 -env MIC_OMP_STACKSIZE=4M \
 $exe -O -i mdin.CPU -o mdout-offload -p prmtop -c inpcrd \
 : -n 1 -env PHI_KMP_PLACE_THREADS="30c,4t,0O" -env PHI_KMP_AFFINITY="scatter" -env
PHI_OMP_NUM_THREADS=30 -env MIC_OMP_STACKSIZE=4M \
 $exe -O -i mdin.CPU -o mdout-offload -p prmtop -c inpcrd  \
 : -n 7 $exe -O -i mdin.CPU -o mdout-offload -p prmtop -c inpcrd
```

Amber 14 – Intel Phi
(offload mode)



In the MIC offload version of PMEMD only the middle two MPI processes are responsible for offloading work to the MIC coprocessor, e.g. if 8 MPI processes are specified, threads 4 and 5 are responsible for offloading to the MIC coprocessor. These two MPI processes simultaneously spawn OMP threads on the MIC coprocessor to execute the offloaded chunks of work. By partitioning the execution command to reflect the decomposition strategy, the number of OMP threads can be manually set. Partitioning of an MPI execution command is done via the use of ":" which is demonstrated in the example runscript above.