User Instructions for the PARADOX Cluster

1. PARADOX Cluster overview

PARADOX Cluster at Scientific Computing Laboratory of Institute of Physics Belgrade consists of 84 worker nodes (2 x quad core Xeon E5345 @ 2.33 GHz with 8GB of RAM). Its computing nodes are interconnected by the star topology Gigabit Ethernet network through three stacked high-throughput Layer 3 switches. In terms of storage resources, PARADOX provides up to 50 TB of disk space to the HP-SEE community.

2. Access to the PARADOX cluster

From your local machine, you need to use the `ssh` command to access login node of the PARADOX Cluster - **ui.ipb.ac.rs** (ssh is a program for logging into the remote machine and for executing commands on it).

```
$ ssh username@ui.ipb.ac.rs
```

If you need a graphical environment you have to use the -X option:

```
$ ssh username@ui.ipb.ac.rs –X
```

This node is used for preparing, submitting jobs to the batch system and some lightweight testing but not for the long-running computations. To log out from **ui.ipb.ac.rs**, you can use the `Ctrl-d` command, or `exit`.

3. File transfer

To transfer files between **ui.ipb.ac.rs** and your local machine, you can use the `scp` command. Create an archive with the directories you want to copy (it will be faster to transfer):

```
$ tar -cvzf archivename.tgz directoryname1 directoryname2
```

or in the case of a file:

```
$ tar -cvzf archivename.tgz filename
```

Transfer the archive to your home directory at **ui.ipb.ac.rs**:

```
$ scp archivename.tgz username@ui.ipb.ac.rs:
```

Uncompress the archive in your target directory:

```
$ tar -xvzf archivename.tgz destinationdirectory
```

4. File systems

There are two file systems available to users at **ui.ipb.ac.rs**:

- `/home`
- `/nfs`

Both file systems have directories like /`home/<USERNAME>` or /`nfs/<USERNAME>`. Only `/nfs` is shared between all nodes on the cluster and this file system should be used for cluster job submission. It is required to put all executables and data used by jobs in this directory.

Additionally, there is another local file system available on each worker node: `/scratch`. This file system should be used only for temporary storage of running jobs (environment variable: `$TMPDIR`).

5. Job submission

Job submissions, resources allocations and the jobs launching over the cluster are managed by the batch system (torque+maui scheduler). From **ui.ipb.ac.rs** jobs can be submitted using the command `qsub`. Batch system server **ce64.ipb.ac.rs** will accept submitted jobs and distribute them to PARADOX cluster worker nodes for execution.

To submit a batch job, you first have to write a shell script which contains:

- A set of directives. These directives are lines beginning with `#PBS` which describe needed resources for your job
- Lines necessary to execute your code

Then your job can be launched by submitting this script to batch system. The job will enter into a batch queue and, when resources are available, job will be launched over allocated nodes. Batch system provides monitoring of all submitted jobs.

Queue **hpsee** is available for user's job submission.

It is important to state that job submission should be performed from the shared `/nfs` file system!

Frequently used PBS commands for getting the status of the system, queues, or jobs are:

`qstat`	list information about queues and jobs
`qstat -q`	list all queues on system
`qstat -Q`	list queue limits for all queues
`qstat -a`	list all jobs on system

`qstat -au userID`	list all jobs owned by user *userID*
`qstat -s`	list all jobs with status comments
`qstat -r`	list all running jobs
`qstat -f jobID`	list all information known about specified job
`qstat -n`	in addition to the basic information, nodes allocated to a job are listed
`qstat -Qf <queue>`	list all information about specified queue
`qstat -B`	list summary information about the PBS server
`qdel jobID`	delete the batch job with *jobID*
`qalter`	alter a batch job
`qsub`	submit a job

5.1. **Sequential Job submission**

Here is a sample sequential job PBS script:

```
#!/bin/bash
#PBS -q hpsee
#PBS -l nodes=1:ppn=1
#PBS -l walltime=00:10:00
#PBS -e ${PBS_JOBID}.err
#PBS -o ${PBS_JOBID}.out

cd $PBS_O_WORKDIR
chmod +x job.sh
./job.sh
```

- `#!/bin/bash` – Specifies the shell to be used when executing the command portion of the script.
- `#PBS -q <queue>` - Directs the job to the specified queue. Queue **hpsee** should be used.
- `#PBS -o <name>` - Writes standard output to <name> (in this case it is `${PBS_JOBID}.out`) instead of `<job script>.o$PBS_JOBID`. `$PBS_JOBID` is an environment variable created by PBS that contains the PBS job identifier.
- `#PBS -e <name>` - Writes standard error to <name> (in this case it is `${PBS_JOBID}.err`) instead of `<job script>.e$PBS_JOBID`.
- `#PBS -l walltime=<time>` - Maximum wall-clock time. `<time>` is in the format `HH:MM:SS`.
- `cd $PBS_O_WORKDIR` – Change to the initial working directory.

- `#PBS -l nodes=1:ppn=1` – Number of nodes (`nodes`) to be reserved for exclusive use by the job and number of virtual processors per node (`ppn`) requested for this job. For sequential job one CPU on a node will be sufficient. We would have the same effect if this line was left out from the PBS script.

Assuming that you are in the `/nfs/<USERNAME>/somefolder` which contains job script in the file `job.pbs` and file `job.sh`, this job can be submitted by issuing following command:

```
$ qsub job.pbs
```

The qsub command will return a result of the type:

```
<JOB_ID>.ce64.ipb.ac.rs
```

Where `<JOB_ID>` is a unique integer used to identify the given job.

To check the status of your job use the following command:

```
$ qstat <JOB_ID>
```

This will return an output similar to:

Job id	Name	User	Time Use	S	Queue
<JOB_ID>.ce64	job.pbs	<username>	00:01:10	R	hpsee

Alternatively you can check the status of all your jobs using the following syntax of the qstat command:

```
$ qstat -u <user_name>
```

To get detailed information about your job use the following command:

```
$ qstat -f <JOB_ID>
```

When your job is finished, files to which standard output and standard error of a job was redirected will appear in your work directory.

If, for some reason, you want to cancel a job following command should be executed:

```
$ qdel <JOB_ID>
```

If qstat <JOB_ID> returns the following line:

qstat: Unknown Job Id <JOB_ID>.ce64

This most likely means that your job has finished.

5.2. **MPI Job Submission**

The cluster has several implementation of MPI, all of them installed in /opt/<MPI_VERSION> directories (for details about compilation take a look in section 6.7). Each of them has its own environment variables:

- mpich-1.2.7p1
  - MPI_MPICH_MPIEXEC=/opt/mpiexec-0.83/bin/mpiexec
  - MPI_MPICH_PATH=/opt/mpich-1.2.7p1
- mpich2-1.1.1p1
  - MPI_MPICH2_MPIEXEC=/opt/mpiexec-0.83/bin/mpiexec
  - MPI_MPICH2_PATH=/opt/mpich2-1.1.1p1
- openmpi-1.2.5
  - MPI_OPENMPI_MPIEXEC=/opt/openmpi-1.2.5/bin/mpiexec
  - MPI_OPENMPI_PATH=/opt/openmpi-1.2.5

MPI_<MPI_VERSION>_MPIEXEC defines MPI launcher for specific MPI implementation:

- MPICH and MPICH2 - mpiexec-0.83 (MPI parallel job launcher for PBS)
- OPENMPI - its own version of mpiexec

Here is a sample MPI job PBS script:

```
#!/bin/bash
#PBS -q hpsee
#PBS -l nodes=3:ppn=8
#PBS -l walltime=00:10:00
#PBS -e ${PBS_JOBID}.err
#PBS -o ${PBS_JOBID}.out

cd $PBS_O_WORKDIR
chmod +x job
cat $PBS_NODEFILE
${MPI_MPICH_MPIEXEC} ./job                    # If mpich-1.2.7p1 is used
#${MPI_MPICH2_MPIEXEC} --comm=pmi ./job       # If mpich2-1.1.1p1 is used
#${MPI_OPENMPI_MPIEXEC} ./job                 # If openmpi-1.2.5 is used
```

Depending of the MPI implementation used, appropriate environment variable (`MPI_<MPI_VERSION>_MPIEXEC`) should be used for launching of MPI jobs. MPI launcher, together with the batch system will take care of proper launching of a parallel job, i.e. no need to specify number of MPI instances to be launched or machine file in the command line. All these information launcher will obtain from the batch system.

All stated PBS directives are same as for the sequential job except the resource allocation line which is, in this case:

```
#PBS -l nodes=3:ppn=8
```

In this statement we are demanding 3 nodes with 8 cores each (3 full nodes, as PARADOX worker nodes are 8 cores machines), all together 24 MPI instances.

Job can be submitted by issuing following command:

```
$ qsub job.pbs
```

By using the qstat command we can view the resources allocated for our parallel job.

```
$ qstat -n hpsee
```

ce64.ipb.ac.rs:

```
                                                  Req'd  Req'd   Elap
Job ID          Username Queue    Jobname       SessID NDS   TSK Memory Time  S Time
--------------------- -------- -------- ---------------- ------ ----- --- ------ ----- - -----
1667065.ce64.ipb    vlada    hpsee    job.pbs         3096     3    --    --       48000 R   --
n09+n09+n09+n09+n09+n09+n09+n09+n17+n17+n17+n17+n17+n17+n17+n17+n18+n18+n18
+n18+n18+n18+n18+n18
```

Job monitoring and canceling is no different than for sequential job and it is already described in sequential job submission section (5.1).

5.3. **OpenMP job submission**

Here is an example of an openMP job submission script:

```
#!/bin/bash
#PBS -q hpsee
#PBS -l nodes=1:ppn=6
#PBS -l walltime=00:10:00
#PBS -e ${PBS_JOBID}.err
#PBS -o ${PBS_JOBID}.out
```

```
cd $PBS_O_WORKDIR
chmod +x job
export OMP_NUM_THREADS=6
./job
```

Executable job is compiled with OpenMP (see 6.7.3. Compiling OpenMP programs section). For the execution of the OpenMP jobs you shouldn't use more than one node, as it is specified in PBS script:

```
#PBS -l nodes=1:ppn=6
```

OpenMP is a shared memory parallel computational library and as such the processes cannot be forked among various machines. Thus, an OpenMP job on the PARADOX Cluster can at most consume 8 CPUs in parallel since the largest SMP on the cluster has 8 cores.

OMP_NUM_THREADS environment variable should be specified, especially in the case when user is not allocating whole node for its job (not using ppn=8). If this is the case and the number of threads is not specified in program, OpenMP executable will use 8 threads (worker nodes at PARADOX have 8 CPU cores) and potentially compete for CPU time with the other jobs running at the same node.

Job submitting, monitoring and canceling is the same as for other previously described types of jobs.

6. Environment

6.1. **Operating System**

Operating system on PARADOX Cluster nodes is Scientific Linux release 5.5, based Red Hat Enterprise 5 Linux.

6.2. **Available shells**

The default shell is bash. Other shells, such as ksh, csh and tcsh are also available. We strongly recommend you to use bash shell.

6.3. **Passwords**

User passwords can be changed using the `passwd` command:

```
$passwd
Changing password for user <username>.
Changing password for <username>
(current) UNIX password:
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully.
```

6.4. **Text editors**

Following text editors are available at **ui.ipb.ac.rs**

- vi
- emacs
- nano
- joe

6.5. **Available compilers**

The available compilers on the cluster are:
- Intel Compiler suite (icc, icpc, ifort)
- GNU compiler suite (gcc, g++, gfortran)

All Intel tools are installed at /opt/intel directory and in order to use them user should source provided scripts, for example:

```
$ source  /opt/intel/composerxe/bin/compilervars.sh intel64
```

Intel tools are not present at PARADOX worker nodes (only at **ui.ipb.ac.rs**) so linking with static Intel libraries (static compilation) is necessary when executables will run on batch system and they are using Intel libraries (for example, libiomp5.so used with OpenMP executables).

It is recommended to use the Intel compilers which provide the best performances.

6.5.1. **Compiler flags**

C/C++

Intel compilers: icc and icpc. Compilation options are the same, except for the the C language behavior. Icpc manages all the source files as C++ files whereas icc makes a difference between both of them.

Basic flags:

- -o exe_file : names the executable exe_file
- -c: generates the correspondent object file. Does not create an executable.
- -g : compiles in a debugging mode
- -I dir_name : specifies the path where include files are located.
- -L dir_name : specifies the path where libraries are located.
- -l<lib_name> : asks to link against the lib<libname> library

Optimizations:

- -O0, -O1, -O2, -O3: optimization levels - default : -O2
- -opt_report: generates a report which describes the optimization in stderr (-O3 required)
- -ip, -ipo: inter-procedural optimizations (mono and multi files)
- -fast: default high optimization level (-O3 -ipo -static).
- -ftz: considers all the denormalized numbers (like INF or NAN) as zeros at runtime.
- -fp-relaxed: mathematical optimization functions. Leads to a small loss of accuracy.

Preprocessor:

- -E: preprocess the files and sends the result to the standard output
- -P: preprocess the files and sends the result in file.i
- -Dname=<value>: defines the "name" variable
- -M: creates a list of dependence

Practical:

- -p: profiling with gprof (needed at the compilation)
- -mp, -mp1: IEEE arithmetic, mp1 is a compromise between time and accuracy

Fortran:

Intel compiler: ifort (Fortran compiler).

Basic flags :

- -o exe_file : names the executable exe_file
- -c: generates the correspondent object file does not create an executable.
- -g: compiles in debugging mode - R.E. 'Debugging'
- -I dir_name: specifies the path where include files are located
- -L dir_name: specifies the path where libraries are located
- -l<libname>: asks to link against the lib<libname> library

Optimizations:

- -O0, -O1, -O2, -O3 : optimization levels - default : -O2
- -opt_report : generates a report which describes the optimization in stderr (-O3 required)
- -ip, -ipo : inter-procedural optimizations (mono and multi files)
- -fast : default high optimization level (-O3 -ipo -static).
- -ftz : considers all the INF and NAN numbers as zeros
- -fp-relaxed : mathematical optimization functions. Leads to a small loss of accuracy
- -align all: fills the memory up to get a natural alignment of the data
- -pad: makes the modification of the memory positions operational

Run-time check:

- -C or -check : generates a code which ends up in 'run time error' (ex : segmentation fault)

Preprocessor:

- -E: preprocess the files and sends the result to the standard output
- -P: preprocess the files and sends the result in file.i
- -Dname=<value>: defines the "name" variable
- -M: creates a list of dependences
- -fpp: preprocess the files and compiles

Practical:

- -p : profiling with gprof (needed at the compilation)
- -mp, -mp1 : IEEE arithmetic, mpl is a compromise between time and accuracy
- -i8 : promotes integers on 64 bytes by default
- -r8 : promotes reals on 64 bytes by default
- -module <dir>: send/read the files *.mod in the dir directory
- -fp-model strict : Tells the compiler to strictly adhere to value-safe optimizations when implementing floating-point calculations and enables floating-point exception semantics. It might slow down your program.

Please refer to the 'man pages' of the compilers for more information.

GNU

Debugging:

- -Wall: Short for “warn about all,” this flag tells gfortran to generate warnings about many common sources of bugs, such as having a subroutine or function with the same name as a built-in one, or passing the same variable as an intent(in) and an intent(out) argument of the same subroutine.
- -Wextra: In conjunction with -Wall, gives warnings about even more potential problems. In particular, -Wextra warns about subroutine arguments that are never used, which is almost always a bug.
- -w: Inhibits all warning messages (Not advised)
- -Werror: Makes all warnings into errors.

6.6. **Available numerical libraries**

MKL Library

Intel MKL library is integrated in the Intel package and contains:

- BLAS
- SparseBLAS
- LAPACK
- Sparse Solver
- CBLAS
- Discrete Fourier and Fast Fourier transform (contains the FFTW interface, R.E. FFTW)

Other libraries

- LAPACK
- BLAS
- FFTW3
- SPRNG

6.7. **Parallel Programming**

6.7.1. **Available MPI Implementations:**

- mpich-1.2.7p1

Installed in `/opt/mpich-1.2.7p1` (environment variable `$MPI_MPICH_PATH`)

- mpich2-1.1.1p1

Installed in `/opt/mpich2-1.1.1p1` (environment variable `$MPI_MPICH2_PATH`)

- openmpi-1.2.5

Installed in `/opt/openmpi-1.2.5` (environment variable `$MPI_OPENMPI_PATH`)

In order to update environment variables `$PATH` and `$LD_LIBRARY_PATH` user can execute following commands:

```
$ export PATH=$MPI_<MPI_VERSION>_PATH/bin:$PATH
$ export LD_LIBRARY_PATH=$MPI_<MPI_VERSION>_PATH/lib:$LD_LIBRARY_PATH
```

6.7.2. **Compiling MPI program**

Here is an example of a MPI program:

```
#include <stdio.h>
#include <mpi.h>

   main(int argc, char **argv)
   {
      int num_procs, my_id;
      int len;
      char name[MPI_MAX_PROCESSOR_NAME];

      MPI_Init(&argc, &argv);

      /* find out process ID, and how many processes were started. */

      MPI_Comm_rank(MPI_COMM_WORLD, &my_id);
      MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
      MPI_Get_processor_name(name, &len);//

      printf("Hello, world.I'm process %d of %d on %s\n", my_id, num_procs, name);

      MPI_Finalize();
   }
```

MPI implementations are using mpicc, mpic++, mpif77 and mpif90 wrappers for compiling and linking MPI programs:

`$ mpicc -o test test.c` (Assuming that `mpicc` is on your `$PATH`)

6.7.3. **Compiling OpenMP programs**

The Intel and GNU compilers support OpenMP.
Example OpenMP program:

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[])
{
int nthreads, tid;

/* Fork a team of threads giving them their own copies of variables */
#pragma omp parallel private(nthreads, tid)
```

```
  {

  /* Obtain thread number */
  tid = omp_get_thread_num();
  printf("Hello World from thread = %d\n", tid);

  /* Only master thread does this */
  if (tid == 0)
    {
    nthreads = omp_get_num_threads();
    printf("Number of threads = %d\n", nthreads);
    }

  }  /* All threads join master thread and disband */

}
```

Intel compilers flag: -openmp

```
$ icc -openmp -o prog prog.c
```

Since Intel compiler is not available at worker nodes of cluster, OpenMP programs have to be compiled statically at **ui.ipb.ac.rs** machine before submission:

```
$ icc -openmp -static-intel -o prog prog.c
```

GNU compilers flag: -fopenmp

```
$ gcc -fopenmp -o prog prog.c
```

6.8. **Available Debuggers**

- Gnu : GDB
- Intel : IDB
- TotalView Debugger