Running Your Code – TG11

Moving from the desktop to HPC: When submitting jobs on Nautilus, we submit them to the queue via the batch system or interactively. Below you will find several examples of how to transfer files from your desktop to Nautilus, a simple timing script, how to compile your code using various compilers, sample batch scripts, commands we use to submit jobs, and a cheat sheet of commands we use to view jobs in the queue. Also, you will see examples of how to run jobs on your laptop versus running jobs on Nautilus.

Transferring files from desktop to HPC

There are several ways you can transfer data from one machine to another. Different methods are recommended depending on the size of the data you need to transfer. For more information on Data Transfers please refer to http://www.nics.tennessee.edu/user-support/general-support/data-transfer. One quick method to transfer your data from your laptop to Nautilus is as follows:

scp /Users/yashemamack/Downloads/tg11-tutorial/samplecode/ macky@login.nautilus.nics.utk.edu:/gpfs/medusa/macky*

The machine you are transferring your data to will prompt you for you token passcode. Sample output of a successful transfer is shown below.

```
decisions                                     100%  20KB  20.4KB/s   00:00
decisions.c                                   100%  290    0.3KB/s   00:00
```

Timing Script

The timing script provides the beginning and ending CPU time, and the total elapsed CPU time that gives an accurate measure for how long your program ran. The code in ***black*** is identical to the sample Hello World program on page 2. The code in ***blue*** is what will need to be added.

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

int main (int argc, char *argv[])
{
        time_t  t0, t1; /* time_t is defined on <time.h> and <sys/types.h> as long */
        clock_t c0, c1; /* clock_t is defined on <time.h> and <sys/types.h> as int */

        long count;
        double a, b, c;

         t0 = time(NULL);
```

```
        c0 = clock();

        printf ("\tBegin (CPU):          %d\n", (int) c0);

        printf ("\t\tHello World! \n");

        t1 = time(NULL);
        c1 = clock();

        printf ("\tEnd (CPU);            %d\n", (int) c1);
        printf ("\tElapsed CPU time:      %f\n", (float) (c1 - c0)/CLOCKS_PER_SEC);

    return 0;
    }
```

Example 1. Hello0.c

From Laptop

- Compile: **gcc –fopenmp hello0.c –o hello0**
- Verify the file was created: **ls hello0**
- Run: **./hello0**
  (Output will be displayed on your screen)

Output:

```
Begin (CPU):          1830
        Hello World!
End (CPU);            1908
Elapsed CPU time:      0.000078
```

From Nautilus

- Compile: **icc -openmp hello0.c -o hello0**
- Verify that the file was created: **ls hello0**
- Create a batch script (Example located below): **vi/vim/emacs/pico hello0.pbs**

Sample batch script

```
#PBS -S /bin/bash
#PBS -N hello0
#PBS -j oe
#PBS -l ncpus=8,mem=32GB,walltime=00:10:00
```

cd $PBS_O_WORKDIR
export OMP_NUM_THREADS=8
./hello0
ja (shows the jobid, job name, etc.) Check man pages for further details

Submitting Jobs

To submit a job to PBS, use the qsub command. The most common way to use qsub is to simply pass it the name of a 'batch script' that contains all of the information about running your job: *qsub hello0.pbs.*

Within your batch script you can add what project you want to charge against (Example is provided below). If you do not include the project within your batch script, your default project will be charged (The project that you have been added to).

- Run: **qsub hello0.pbs**
  A jobid will be created. Ex. 123456.nid00016

Command Cheat sheet

checkjob : allow you to view the details of a job in the queue
showstart <jobid>: provides an estimate of when a job will start
showq : gives a different view of jobs in the queue and show jobs in the states (running, eligible, blocked)
qstat –a | grep <jobid>: allow you to check the status of submitted jobs
xtnodestat: allow you to see which jobs are currently running and which cabinets, nodes, and processors they are running on

- Once job is completed do the following: **ls –l**
  Find the 'hello' program with the jobid at the end (hello0.o123456)
- Type: **more hello0.o123456**

Output:

Hello World!
Job Id: 44312.nemo.nics.utk.edu
Job_Name = hello0

While creating your batch scripts *ncpus* (number of cpus), by default, are allocated in multiples of 8 and memory in multiples of 32. For more information on job accounting refer to http://www.nics.tennessee.edu/computing-resources/nautilus/job_accounting.

If you decide not to set the memory limit within your batch script, a memory limit will be allocated for you based on the number of cpus you requested. For example, if you set your #PBS –l ncpus=16 and run your job the following message will appear:

> Warning:
> You job script does not set a memory limit, which is
> required on Nautilus. Based on the rest of your job
> request, a memory limit of 64000MB has been added.
> Please contact help@xsede.org if you need assistance.
> 12345.nemo.nics.utk.edu
>
> As you can see above, your job was still submitted to the queue, and provided you with a jobid (12345.nemo.nics.utk.edu).

For more advanced PBS options and batch script information please refer to: http://www.nics.tennessee.edu/computing-resources/nautilus/Batch_Scripts.

Example 2. hello1.c

From Laptop

Instructions:

- Compile: **gcc –fopenmp hello1.c –o hello1**
- Verify that file was created: **ls hello1**
- Run: **./hello1**

Output:

> Hello world.
> Hello from a thread!
> Hello from a thread!
> I am sequential now.

From Nautilus (Intel Compiler)

- Compile: **icc -openmp hello1.c -o hello1**
- Verify that the file was created: **ls hello1**
- Create a batch script (Example located below): **vi/vim/emacs/pico hello1.pbs**

Sample batch script

> #PBS -S /bin/bash
> #PBS -A <project> *ex. TG – TERA1234*
> #PBS -N hello1

```
#PBS -j oe
#PBS -l ncpus=8,mem=32GB,walltime=00:10:00

cd $PBS_O_WORKDIR
export OMP_NUM_THREADS=2
./hello1
```

- Run: **qsub hello1.pbs**
  A jobid will be created. Ex. 123456.nid00016
  Please refer to the 'Command Cheat Sheet' for commands to view your job.

- Once job is completed do the following: **ls –l**
  Find the 'hello' program with the jobid at the end (hello0.o123456)
- Type: **more hello1.o123456**

Output:

```
Hello world.
Hello from a thread!
Hello from a thread!
I am sequential now.
```

Compilers

	PGI	GNU	Intel
C	pgcc	gcc	icc
C++	pgCC	g++	icpc
Fortran	pgfortran	gfortran	ifort

To compile an OpenMP program using OpenMP directives, use the -openmp compiler option. With Fortran programs, you will also need to specify the "-fpp" Fortran precompiler option.

Examples:

C/Intel: icc -openmp helloworld.c -o helloworld

Fortran/Intel: ifort -o helloworld helloworld.f -fpp -openmp

C/GNU: gcc -fopenmp helloworld.c -o helloworld

Fortran/GNU: gfortran -fopenmp helloworld.f -o helloworld

C/PGI: pgcc -mp helloworld.c -o helloworld

Fortran/PGI: pgfortran -mp helloworld.f -o helloworld

Swapping Modules

If want to run the code using a different compiler just swap one compiler for the other and run your code with the compiler options located above. For example, on the command line type *module list* to see what compiler is loaded by default

macky@arronax:~> module list
Currently Loaded Modulefiles:
1) modules
2) torque/3.0.2
3) moab/6.0.4
4) intel/11.1.038
5) mpt/2.01
6) PE-intel
7) gold
8) hsi
9) /nics/a/proj/nics/support/modulefiles/search_tickets

If I want to switch from the Intel compiler to the GNU compiler do the following:

macky@arronax:~> module swap PE-intel PE-gnu

macky@arronax:~> module list
Currently Loaded Modulefiles:
1) modules
2) torque/3.0.2
3) moab/6.0.4
4) PE-gnu
5) mpt/2.01
6) gnu/4.3.4
7) gold
8) hsi
9) /nics/a/proj/nics/support/modulefiles/search_tickets

Example 3. hello2.c (Example 1 – page 2)

From Laptop

- Compile: **gcc -fopenmp hello2.c -o hello2**
- Verify that the file was created: **ls hello2**
- Run: **./hello2**

Output:

Hello world.
Hello from thread 0.
Hello from thread 1.
I am sequential now.

Output: *./hello | sort*

Hello from thread 0.
Hello from thread 1.
Hello world.
I am sequential now.

From Nautilus (GNU Compiler)

- Compile: **gcc -fopenmp hello2.c -o hello2-g**
- Verify that the file was created: **ls hello2-g**
- Create a batch script (Example located below): **vi/vim/emacs/pico hello2-g.pbs**

Batch Script with omplace

```
#PBS -S /bin/bash
#PBS -N hello2-g
#PBS -j oe
#PBS -l ncpus=16,mem=64GB,walltime=00:10:00

cd $PBS_O_WORKDIR
export OMP_NUM_THREADS=16
omplace -nt 16 -c 0-15 ./hello2-g
sleep 10
pidstat -t -p ALL | grep hello2-g > myhello2-g_out
```

- Run: **qsub hello2-g.pbs**
- Once job is completed do the following: **ls –l**
  Find the 'hello' program with the jobid at the end (hello0.o123456)
- Type: **more hello2-g.o123456**

Output: (First run)

Hello world.
Hello from thread 0.
Hello from thread 1.
Hello from thread 2.
Hello from thread 3.
Hello from thread 4.
Hello from thread 5.
Hello from thread 6.
Hello from thread 7.
Hello from thread 8.

Hello from thread 14.
Hello from thread 15.
Hello from thread 12.
Hello from thread 10.
Hello from thread 13.
Hello from thread 9.
Hello from thread 11.
I am sequential now.

Output: (Second run) with 'ja' command in batch script

Hello world.
Hello from thread 0.
Hello from thread 1.
Hello from thread 2.
Hello from thread 3.
Hello from thread 4.
Hello from thread 5.
Hello from thread 6.
Hello from thread 7.
Hello from thread 10.
Hello from thread 13.
Hello from thread 12.
Hello from thread 8.
Hello from thread 11.
Hello from thread 9.
Hello from thread 14.
Hello from thread 15.
I am sequential now.
Job Id: 45332.nemo.nics.utk.edu
Job_Name = hello2-g
resources_used.cput = 00:00:59
resources_used.mem = 62376kb
resources_used.vmem = 120032kb
resources_used.walltime = 00:01:09

Example 4. hello3.c, loop0.c, loop1.c, loop2.c (Example 1 – pages 3-5)

From Laptop

- Compile: **gcc -fopenmp hello3.c -o hello3**
- Verify that the file was created: **ls hello3**
- Run: **./hello3**

Output: hello3.c

Hello from thread 3.
Hello from thread 0.
Hello from thread 6.
Hello from thread 4.
Hello from thread 5.
Hello from thread 1.
Hello from thread 2.
Hello from thread 7.
Thread 3 sleeping.
Thread 0 sleeping.
Thread 6 sleeping.
Thread 4 sleeping.
Thread 5 sleeping.
Thread 1 is sick of this program.

Output: loop0.c (no parallelization)

Greetings from i = 0.
Greetings from i = 1.
Greetings from i = 2.
Greetings from i = 3.
Greetings from i = 4.
Greetings from i = 5.
Greetings from i = 6.
Greetings from i = 7.
Greetings from i = 8.
Greetings from i = 9.

Output: loop1.c (incorrect parallelization)

Greetings from i = 0.
Greetings from i = 0.
Greetings from i = 1.
Greetings from i = 2.
Greetings from i = 3.
Greetings from i = 4.
Greetings from i = 5.
Greetings from i = 6.
Greetings from i = 7.
Greetings from i = 8.
Greetings from i = 9.

Output: loop2.c (correct parallelization)

Greetings from i = 0.

Greetings from i = 5.
Greetings from i = 1.
Greetings from i = 6.
Greetings from i = 7.
Greetings from i = 2.
Greetings from i = 8.
Greetings from i = 3.
Greetings from i = 9.
Greetings from i = 4.

From Nautilus (PGI Compiler)

- Compile: **pgcc -mp hello3.c -o hello3-p**
- Verify that the file was created: **ls hello3-p**
- Create a batch script (Example located below): **vi/vim/emacs/pico hello3-p.pbs**

Batch Script with dplace

```
#PBS -S /bin/bash
#PBS -N hello3-p
#PBS -j oe
#PBS -l ncpus=24,mem=96GB,walltime=00:10:00

cd $PBS_O_WORKDIR
export KMP_AFFINITY=disable
export OMP_NUM_THREADS=8
dplace -c 0-7 ./hello3-p
```

*dplace **-c** 0-7 ./hello3-p:* the ***-c*** option allows you to specify which cpu or range of cpus you want your processes to run on. For other options refer to the man pages.

- Run: **qsub hello3-p.pbs**
- Once job is completed do the following: **ls –l**
- Type: **more hello3-p.o123456**

Output:

```
Hello from thread 5.
Thread 5 sleeping.
Hello from thread 0.
Thread 0 sleeping.
Hello from thread 3.
Thread 3 sleeping.
```

Hello from thread 7.
Thread 7 sleeping.
Hello from thread 6.
Thread 6 sleeping.
Hello from thread 4.
Thread 4 sleeping.
Hello from thread 1.
Thread 1 is sick of this program.
Hello from thread 2.
Thread 2 is really excited!!!
Thread 2 sleeping.

Example 5. sections0.c, sections1.c – Controlling threads (Example 4a – page 10)

Output: time ./sections0

This is function 1.
This is function 2.
This is function 2.
This is function 2.

real 0m7.107s
user 0m0.000s
sys 0m0.002s

Output: time ./sections1

This is function 1.
This is function 2.
This is function 2.
This is function 2.

real 0m4.089s
user 0m0.001s
sys 0m0.003s

Example 6. hello4.c, hello4-serial, hello4-parallel (Example 5c – page 13)

Output: ./hello4

Hello world.
Hello from thread 0.
Hello from thread 1.
I am sequential now.

Output: ./hello4-serial

Hello world.
Hello from thread 0.
I am sequential now.

Output: ./hello4-parallel

Hello world.
Hello from thread 0.
Hello from thread 1.
I am sequential now.

Example 7. version.c - OpenMP Version (Example 5d – page 14)

- Compile: **gcc –fopenmp version.c –o version**

  Output: ./version

  Compiled with OpenMP, version dated: 200505

- Compile: **gcc –o version1 version.c**

  Output: ./version1

  Not compiled with OpenMP.

Interactive jobs

To submit an interactive job you must use the *qsub –I* command on the command line. The *–I* initiates the interactive session. You can also use other variable such as *–l* which allows you to specify how many cpus and how much memory you want to use, as well as, how long you want to run your job. Ex. *qsub –I –l ncpus=8, memory=32GB, walltime=00:10:00.*