OpenMP Submissions via SGE to Pythagoras

Panagiotis Nastou, PhD

Department of Mathematics

School of Sciences

University of the Aegean

Georgios Petroudis, MSc

Department of Mathematics

School of Sciences

University of the Aegean

OpenMP Deployment in Pythagoras

There are two phases for the deployment of an openMP parallel environment in Pythagoras HPC. The first phase is the configuration of a parallel environment under SGE on which the OpenMP is going to operate. The second is the queue assignment where we define the set of nodes that the openMP will be available.

As for the configuration of a parallel environment, we need to log in as root. After that, the steps are as follows:

We are going to see the list of the parallel environments to make sure that openMP is not one of them

qconf -spl

Now we are ready to configure the environments. We use the qconf for the configuration, the -ap flag as add parallel and the name of the parallel environment which we want to configure, in our case openMP

qconf -ap openMP

After that a list of variables appears. The following table contains the variables and the values we set. It also contains the reason for this assignment.

Index	Configuration Fields	Value	Comments
1	pe_name	openMP	
2	slots	999	The total number of job slots that can be occupied by all parallel environment jobs running concurrently.
3	users	NONE	The user access lists that are allowed to access the parallel environment. Currently, every user of Pythagoras have access to this parallel environment.
4	xusers	NONE	The user access lists that are not allowed to access the parallel environment. Currently, there is no such list.
5	start_proc_args	NONE	These fields are optional. We can enter the startup and stop procedures of the parallel environment along with the appropriate SGE parameters. They run under the same environment setting as that the job to be started afterwards. Currently, we set NONE (a tightly integrated
6	stop_proc_args	NONE	

			PE for reliable control of resources, full process control and correct accounting).
7	allocation_rule	$pe_slots	The allocation rule helps the scheduler to decide how to distribute parallel processes among the available nodes. If, for instance, a parallel environment is built for shared memory applications only, all parallel processes have to be assigned to a single machine, no matter how much suitable machines are available. If set it to the special denominator $pe_slots is used, the full range of processes as specified with the qsub(1) -pe switch has to be allocated on a single host (no matter which value belonging to the range is finally chosen for the job to be allocated).
8	control_slaves	FALSE	It indicates whether the Grid Engine is the creator of the slave processes of a parallel application. We decided to set the FALSE value to this field since user application will create slave processes.
9	job_is_first_task	TRUE	The FALSE value indicates that the job script is not part of the parallel program, just being used to kick off the processes that do the work while TRUE means the opposite. A value of TRUE indicates that the Grid Engine job script already contains one of the tasks of the parallel application (the number of slots reserved for the job is the number of slots requested with the -pe switch), while a value of FALSE indicates that the job script (and its child processes) is not part of the parallel program.
10	urgency_slots	min	For pending jobs with a slot range PE request with different minimum and maximum, the number of slots they actually use is not determined. This field determines the method to be used by SGE to assess the number of slots such jobs might finally get.
11	accounting_summary	FALSE	The TRUE value indicates that only one record is written in the accounting file containing the accounting summary of the whole job including all slave tasks. The FALSE value indicates an individual accounting record is written for every slave task as well as for the master task. This field is valuable only if the control_slaves is set to TRUE.

2. Now is the time to add the openMP in certain queue lists.

First we can see all the queue lists available.
qconf -sql
In Pythagoras there are four queue lists: the all.q manages all cluster computation nodes, the math8Queue.q manages the nodes with 8 physical cores (16 logical cores), the math12Queue.q manages the nodes with 12 physical cores (24 logical cores) and the icsdQueue.q manages the nodes with 32 physical cores (64 logical cores).

We add the OpenMP in all of the last three queues (math8Queue.q, math12Queue.q, icsdQueue.q). We use the qconf for the configuration, with the following options: -aattr (add attribute), the value queue for which the attribute pe_list we are going to modify, the name of the parallel environment openMP and finally the name of the queue where the openMP will be added.

qconf -aattr queue pe_list openMP math8Queue.p

qconf -aattr queue pe_list openMP math12Queue.p

qconf -aattr queue pe_list openMP iscdQueue.p

openMP job submissions under SGE

Now we need a script as a wrapper for the job submission at any OpenMP program we are going to run. We present four wrappers for the user depending on the queue that will be used. We discourage any change of the wrappers beside the ones we point out. The order must be strict and as given, otherwise an error is going to be occur.

Now we present the wrappers.

openmpWrapper_all.sh

#Initialize SGE

#We use bash shell to interpret the SGE script

#!/bin/bash

#Specify the interpreting shall that SGE use to interpret the scripts below

#$ -S /bin/bash

#We define the name of the job that will be submitted for execution.

#The user can change the name of the job

#$ -N openmp_all

#Determine the path and the file name for the standard output and error stream of the job.

#$ -o ./omp_all.o

#$ -e ./omp_all.e

#We merge the error and the output file in one file

#$ -j y

#Execute the job script from the current working directory

#$ -cwd

#Execute the job script in the all.q queue

#Note that if this script is used for a machine beside Aristarxos or Pythagoras then

#the all.q name must change to the queue name that this machine use

#$ -q all.q

```
#Allow command to be binary file instead of script
#$ -b y
#The job is going to start in the parallel environment of openMP using 4 cores
#The user can change this numbers depending the needs. The upper value is 64 because
#there is no node in alll.q with more than 64 logical cores
#$ -pe openMP 4
#We specifically pass environment virable OMP_NUM_THREADS to the job
#This variable is all the threads are going to be open by SGE and it is fixed to 8
#The user can change this value depending the needs. The lower bound to this is the value
#we set in -pe openMP, in our fixed case 4. We note that the upper bound depends on the power
#of the machine one uses and the difficulty of the program running
#$ -v OMP_NUM_THREADS=8

#The section in which the script run the .out file along this all its arguments
./$@
```

At the other three wrappers we are not going to put comments as the comments above apply for the wrappers that follow.

openmpWrapper_icsdQueue.sh

```
#!/bin/bash
#$ -S /bin/bash
#$ -N openmp_icsd
#$ -o ./omp_icsd.o
#$ -e ./omp_icsd.e
#$ -j y
#$ -cwd
#$ -q icsdQueue.q
#$ -b y
#$ -pe openMP 4
#$ -v OMP_NUM_THREADS=8

./$@
```

openmpWrapper_math8Queue.sh

```
#!/bin/bash
#$ -S /bin/bash
#$ -N openmp_math8
#$ -o ./omp_math8.o
#$ -e ./omp_math8.e
#$ -j y
#$ -cwd
#$ -q math8Queue.q
#$ -b y
#$ -pe openMP 4
#$ -v OMP_NUM_THREADS=8

./$@
```

openmpWrapper_math12Queue.sh

```
#!/bin/bash
#$ -S /bin/bash
#$ -N openmp_math8
#$ -o ./omp_math8.o
#$ -e ./omp_math8.e
#$ -j y
#$ -cwd
#$ -q math12Queue.q
#$ -b y
#$ -pe openMP 4
#$ -v OMP_NUM_THREADS=8

./$@
```

After we write those wrappers we need to make the executables. We do that for every wrapper with the following command:

#chmod +x openmpWrapper_all.sh

#chmod +x openmpWrapper_icsdQueue.sh

#chmod +x openmpWrapper_math8Queue.sh

#chmod +x openmpWrapper_math12Queue.sh

Now the script is ready for submission.

Running pi.c on Pythagoras

We use the pi.c program written in C and uses OpenMP. You can find the code in the appendix section. At first we need to compile the pi.c with the gcc compiler. In addition we need to use the option for the OpenMP -fopenmp. Finally, we can use to -o output flag to name the output name and diraction. We assume that the wrapper and the C program are in the same directory, let it be myOpenMPProject.

$cd myOpenMPProject

$gcc -fopenmp -o pi.out pi.c

At this point the pi.out appear in the same directory. A look in the pi.c code shows us that for the execution we need an argument.

If run this on our machine or in the Front End all we have to do is to give the command

$./pi.out arg1

where arg1 is the argument needed for the pi.out execution.

If it is to submit our program in any computation node of Pythagoras we use the wrapper we made. Let, say, we want to use the math12Queue.q queue (if we want to use a different queue we just use a different wrapper). Then, we give the command

$qsub ./openmpWrapper_math12Queue.sh ./pi.out arg1

ATTENTION: It is necessary to use ./openmpWrapper_math12Queue.sh instead of just openmpWrapper_math12Queue.sh because then the qsub will not know where this script is. The same goes to ./pi.out. After the submission we can check the job status with the command

$qstat -f

We can see that the selected queue is indeed the math12Queue.q. After the job completion, the file omp_math12.o appears in the current directory. That file contains the results. To access the results we give the command

$cat omp_math12.o

APPENDIX

----------pi.c----------

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#define pi 3.14159265358979323846264 3

double g(double x)
{
return (4.0/(1.0+x*x));
}

int main(int argc, char* argv[])
{
int i, N;
double dx, piApprox=0.0, x, piDiff, elapsed;
N=atoi(argv[1]);
dx=1.0/((double) N);
elapsed=omp_get_wtime();
#pragma omp parallel for private(i,x) reduction(+:piApprox)
for(i=0; i<N; i++){
x = (((double) i)+0.5)*dx;
piApprox += g(x);
}
piApprox *= dx;
piDiff=fabs(pi - piApprox);
elapsed=omp_get_wtime()-elapsed;
printf("Number of Launched Threads=%d\n", omp_get_max_threads());
printf("Elapsed Time= %f s\n", elapsed);
printf("Estimated Value = %24.16f\n", piApprox);
printf("Divergence= %24.16f\n", piDiff);
```

```
return 0;
}
```

----------end of pi.c----------