DSSC - HPL Benchmark

Presta Alberto

December 2018

Introduction

The aim of this exercise is to compile the hpl benchmark against mkl libraries and to tune it in order to get closer theoretical peak performance of a node of the Ulysses cluster. In the second part of the exercise we will try to use an optimized version provided by Intel.

HPL

First of all, What is "Benchmark"? It is the act of running a computer program (or other operation) in order to assess the relative performance of an object. We know that there are different type of benchmarks, each of them specialized in evaluating different aspects concerning the performance of an object. *HPL* is one of them: It is a High-Performance Benchmark implementation which solves a uniformly random system of n linear equations in double precision (64 bits) and reports time and floating-point execution rate. Obviously HPL is focused on evaluating a CPU computational speed: indeed HPL is no longer indicated to measure overall performance of a HPC system, mainly because it is only able to stress the floating point unit and not the overall infrastructure. During the exercise we will load and use the MKL library provided by Intel, which contains high optimized mathematical routines including BLAS. We have said below that we are interested in getting closed to the theoretical peak performance; but what is it? It is the performance, in terms of number of floating point addiction and multiplication, that a HPC structure can achieve in a cycle of time of the machine. We calculate it with the following formula:

$$T.p.p = (Cores\_per\_Socket) \cdot Sockets \cdot \frac{Flops}{Cycle} \cdot Clockrate$$

For Ulisse's node we have that $T.p.p = 480.0$ GFlops.

Benchmarking

First of all we have installed HPL tools and we have loaded MKL library and OPENMPI. Then, in order to fill the dat file, we have used http://www.advancedclustering.com/act_kb/tune-hpl-dat-file/: our purpose is to find the right combination in terms of N (size of the problem), Nb (block size), P and Q (grid of the MPI processes) in order to achieve at least the 75% of the peak performance.

We will start with these data:

```
HPL.out      output file name (if any)
6            device out (6=stdout,7=stderr,file)
1            # of problems sizes (N)
64512        Ns
1            # of NBs
256          NBs
1            # of process grids (P x Q)
4            Ps
5            Qs
```

...With the following results:

N	NB	P	Q	Time	Gflops	%
64512	256	4	5	427.75	4.185e+02	93.4
64512	256	4	5	440.19	4.066e+02	90.75
64512	256	4	5	421.69	4.245e+02	94.4
64512	256	4	5	421.8	4.251e+02	94.8
64512	256	4	5	436.16	4.104e+02	91.6

We observe that, if we swap P and Q it doesn't change anything, so we have tried to change the number of blocks with the following results:

N	NB	P	Q	Time	Gflops	%
64512	750	4	5	462.23	3.872e+02	86.42
64512	192	4	5	431.61	4.147e+02	92.56
64512	1000	4	5	500.09	3.579e+02	79.8

Intel Linpack

In this part of the exercise we simply tried to run the highly optimized version of HPL which Intel provides precompiled. We retrieved the benchmarks from shared files and we compiled it. the parameters are the following one[1]:

```
Number of tests: 2
Number of equations to solve (problem size) : 64512 65000
Leading dimension of array                  : 64512 65000
Number of trials to run                     : 1     1
Data alignment value (in Kbytes)            : 1     1
```

...and we achieved the following results:

Size	time	GFLops	%
64512	406.448	440.398	98.30
65000	412.933	443.3935	98.9

As we expected the results are much more better that the previous ones; indeed we are able to achieve almost the 100% of the theoretical peak performance.

[1] Nb=256

Multithread Xhpl

Using the command $OMP\_NUM\_THREADS$ we should be able to set correctly the number of threads: Unfortunately this command doesn't as soon as we observed that the performance decreases when we decrease the number of procs (this should not happens). We supposed that the program does not recognize the command and for this reasons the number of threads doesn't change. Below there are our results, changing only the number of procs (and obviously the number P and Q):

Size	−np	−th	P	Q	GFlops
64512	10	2	5	2	2.273e+02
64512	5	4	5	1	1.076e+02
8096	2	10	2	1	4.469e+01
8096	1	20	1	1	2.608e+01

We observed that even if We have tried to change the number of threads, the performance decreases a lot; this should not happen because if you have multiple threads the job should be split between them and therefore the performance should increase.