Exercise 6 - HPL benchmark using MKL multithread library

Michela Venturini

November 26, 2018

Abstract

The aim of this exercise is to compile the hpl benchmark against the mkl libraries and to tune it in order to get performances close to the theoretical peak performance of a node of the Ulysses cluster.

1 Use HPL

HPL HPL is a software package that solves a (random) dense linear system in double precision (64 bits) arithmetic on distributed-memory computers by using LU decomposition. It is used to evauate the performance of the architecture where it runs, defined as the nuber of Flop/s. Since is difficult to obtain the precise nubero of flops, it estimates them as:

$$\frac{2n^3}{3} + 2n^2 \tag{1}$$

where n is the size of the problem.

Install HPL Firts we install HPL and we create a Makefile in the top directory by using those found in hpl.2-2/setup. We modify it in the following way:

- set appropriately the TOPDIR variable
- comment out the MPI variable because we will use the mpi wrapper compiler (mpicc)
- define the compiler and the linker as mpicc wrapper

Set the parameters After loading mkl and openmpi modules, we compiled using `arch=mkl`; After this passage the executable `xhpl` is created. To run it in the proper way, we modify the parameters of the input file:

- N (size of the problem)
- Nb(block size)
- P and Q (grid of the MPI processes)

in such a way that allows us to get at least 75% of peak performances of Ulysses' node (448 GFlops) running the program on 20 cores as 20 MPI processes on a single node. The values i actually used are:

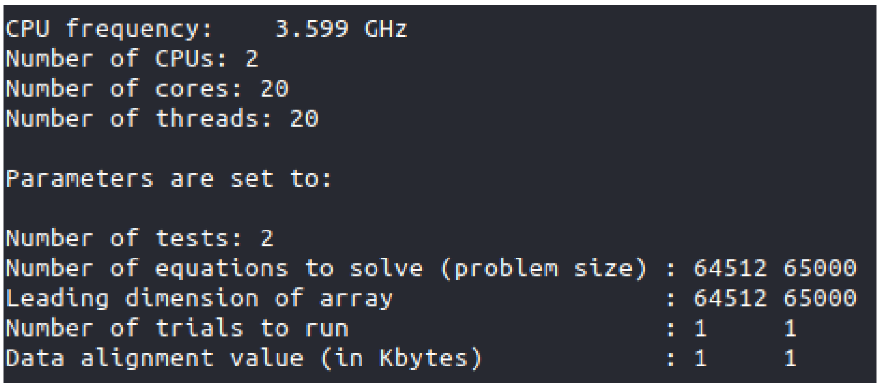
1. N = 64512 which should be as big as possible in order to use almost 80% of the memory of the node.
   To calculate N i use:
   $$N = \frac{\sqrt{\frac{Mem_{node} inGB}{8B}}}{100} * 92 \tag{2}$$
   The memory for each node is 40 GB, that divided per 8 B gives the number of floats of a matrix that occupy all the memory. At this point using square root we find the length of the border of the matrix and we take 92% of this value.
2. Nb = 256 that shuold be as small as possible and determine the granularity of the alorithm
3. P = 4 and Q = 5 that determine the grid of the processes and how the matrix is divided through the processes; the two values should be equal or Q slightly bigger than P.

Execution We execute `xhpl` by using a single node and 20 processors. The performance we obtain are 426 Gflops that is 96% of the peak performances that is a very good result.

2 Comparison with the highly optimized version of HPL provided by Intel

To compare performances we use the highly optimized executable of HPL by Intel fount in the folder `$MKLROOT/benchmarks/linpack`. We modify the input file `lininput_xeon64` by using the dimensions that allow the best performancein the same conditions. The summary input is shown in figure 1.

The results are shown in the figure 2 and compared with the performance obtained before the Intel optimized hpl benchmark achieve better results. This is due to the optimizations of the Intel benchmark that are highly optimized for Intel processors. Anyway the performance obtained are not so different from the previous.

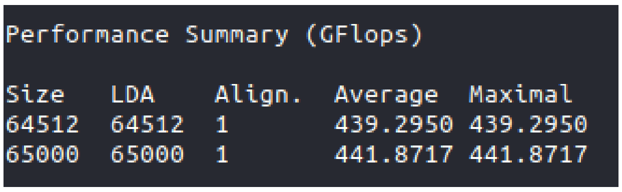


```
CPU frequency:    3.599 GHz
Number of CPUs: 2
Number of cores: 20
Number of threads: 20

Parameters are set to:

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
```

Figure 1: Input parameters



```
Performance Summary (GFlops)

Size   LDA    Align.  Average  Maximal
64512  64512  1       439.2950 439.2950
65000  65000  1       441.8717 441.8717
```

Figure 2: Output of the test

3 Tuning HPL executable

At this point we a set of HPL.dat files to run the hpl benchmark using different combinations of MPI processes and threads. To modify the number of threads we the environmental variable `OMP_NUM_TREADS`. The tests executed are summarized in the table 2.

MPI processes	Threads	P	Q	GFlops	% of theoretical
20	1	4	5	4,161e+02	92,9
10	2	2	5	2,277e+02	50,6
5	4	1	5	1,178e+02	26,1
4	5	2	2	1.017e+02	22,7
2	10	1	2	5.135e+01	11,5
1	20	1	1	2.708e+01	6,0

Table 1: Performance of HPL

The results are not what we expect: decreasing the number of MPI processes and increasing the number of threads the performance decreases faster - half themselves as the number of processes. On the contrary I was expecting that the performance would remain constant. This is due to the fact that the environmental variable `OMP_NUM_TREADS` does not affects the number of threads; the table below is another demonstration of this fact.

Nodes	MPI processes	Threads	P	Q	GFlops	% of theoretical
1 (20 ppn)	20	1	4	5	4,161e+02	92,9
1 (20 ppn)	20	2	4	5	4.121e+02	92,0
2 (20 ppn)	40	1	5	8	8.119e+02	90,6
2 (20 ppn)	40	2	5	8	8.004e+02	89,3
2 (20 ppn)	40	4	5	8	7.958e+02	88,8
2 (20 ppn)	40	8	5	8	7.949e+02	88,7

Table 2: Effectiveness of OMP_NUM_THREADS