Exercise 6 - HPL Benchmarking

Alessia Paoletti
Matricola SM3500374

HPL is a portable implementation of the LINPACK benchmark. The LINPACK Benchmarks are a measure of a system's floating point computing power that measure how fast a computer solves a dense n by n system of linear equations Ax = b. The latest version of these benchmarks is used to build the TOP500 list, ranking the world's most powerful supercomputers. The aim is to approximate how fast a computer will perform when solving real problems.

Running the `xhpl` program with the values reported in Table 1 I have reached a performance of $4.188e+02$ GFlops, that means almost 93% of the peak perfomance, that is almost $4.48e+02$ GFlops.

1	# of problems sizes (N)
64512	Ns
256	NBs
1	# of process grids (P x Q)
4	Ps
5	Qs

Table 1: Parameters for `xhpl` execution

Running the highly optimized version of HPL which Intel provides precompiled with the same combination parameters of Table 1 I have obtained a performance of $4.405e+02$ GFlops, that means almost 98% of the peak performance. Of course, as expected, with the highly optimized version provided by Intel we obtain better results, with an increase of 5% considering the theoritical peak performance.

I have tried to run the HPL benchmark (the `xhpl` program, not the Intel version) using different combinations of number of MPI processes and number of threads. In order to change the number of MPI processes the values of P and Q (the number of process rows and columns of the process grid) need to be cahnged in such a way that $P * Q = \#MPI\_processes$ with $P \leq Q$. Doing different runnings changing both the number of MPI processes and the number of threads we obtain the results reported in Table 2. Theoretically the performance has to remain constant, but in this exercise as the number of MPI processes decreases the perfomance decreases. So it seems that the number of threads is not considered.

In order to prove that the number of threads setted is not considered at runtime I have tried to keep constant the number of MPI processes and change the number of threads. The results are reported in Table 3. We can observe that as the number of threads increases the performance remains more or less constant and does not increases as expected.

I have finally try to run the program using 40 MPI processes and using 2 nodes of Ulysses. Also in this case, as the Table 4 shows, even if we set the number of threads, it is not considered. The performance remains constant and it is almost the double of the performance respecting using one single node.

MPI processes	**Threads**	**P**	**Q**	**Performance [GFlops]**	**% Peak peformance**
20	1	4	5	4.216e+02	94.1
10	2	2	5	2.258e+02	50.4
5	4	1	5	1.239e+02	27.7
4	5	2	2	1.016e+02	22.7
2	10	1	2	5.139e+01	11.5
1	20	1	1	2.769e+01	6.2

Table 2: Perfomance obtained changing number of MPI proccesses and number of threads simultaneosly

MPI processes	**Threads**	**P**	**Q**	**Performance [GFlops]**
20	1	4	5	4.216e+02
20	2	4	5	4.138e+02
20	3	4	5	4.132e+02
20	4	4	5	4.168e+02

Table 3: Performance with 20 MPI processes

MPI processes	**Threads**	**P**	**Q**	**Performance [GFlops]**
40	1	5	8	8.075e+02
40	2	5	8	7.595e+02
40	3	5	8	7.595e+02
40	4	5	8	7.987e+02

Table 4: Performance with 40 MPI processes