HPL Benchmarking

Nandhana Sakthivel

Exercise 6

Aim

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

Introduction

LINPACK is a benchmark which consists in solving a dense system of linear equations, which are randomly generated. It is used in the TOP500 and it allows to tune its parameters to achieve the best performance. Of course, it is just one number, and so it does not reflect the overall performance of the given system. The performance achieved on this benchmark is quite always an overestimation of the performance that the same system could achieve on a real world application. In real world applications, this is not an easy task.

HPL

HPL is an implementation of the LINPACK benchmark.The HPL benchmark (the xhpl program) is ran 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 changed in such a way that,
P $\star Q = MPI processes\, with P \leq Q$

The application is run for the following combination:

- **N** - 64512
- **Nb** - 256
- **Ps** - 4
- **Qs** - 5

The performance obtained by using this **HPL Benchmark** was around **4.164e+02** which is almost **93%** of the peak performance. When we ran the same combination for the highly optimised **Intel version of the HPL** the performance was around **4.278e+02** which is **95%** of the peak performance

HPL Benchmark for different combinations

The HPL benchmark is run for different combinations of N, Nb, Ps and Qs and the following results are achieved

MPI Processes	**Threads**	**Ps**	**Qs**	**Performance**	**% Peak Performance**
20	1	4	5	414	92%
10	2	2	5	220	49%
5	4	1	5	173.9	39%
4	5	2	2	101.6	27%
2	10	1	2	51.39	11.4%
1	20	1	1	29.2	6.5%

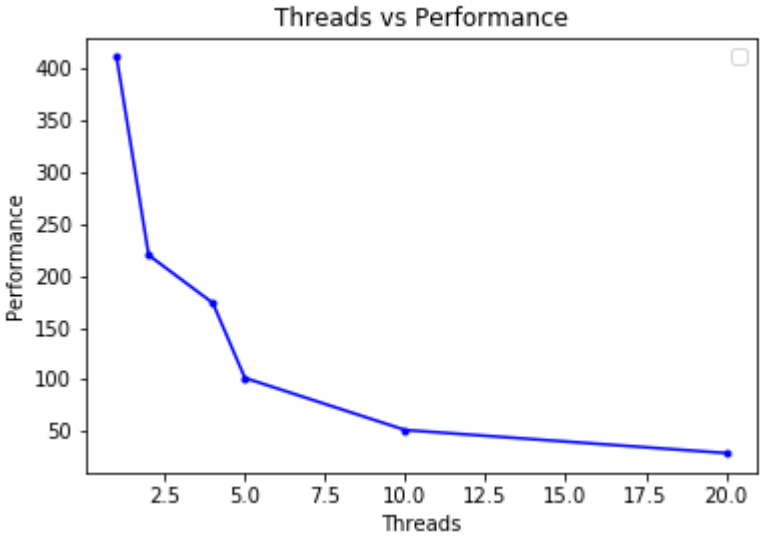

Figure 1: Threads Vs Performance

The above given graph shows that when the no of threads associated with the process increases, the performance tends to decrease. Hence, the best performance can be achieved by setting low thread count due to the communication overhead.