Running the HPL benchmark

Francesco Cicala

Assignment 6

Abstract

For this assignment we compile the High Performance Linpack benchmark using the MKL multithread library. Therefore, we tune the parameters to get close to the theoretical peak performance on a 20-cores node of the SISSA's Ulysses cluster.

1. 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 some sense, the benchmark has composed to let a system to easily exploit a significant percentage of its computational resources. In real world applications, however, this is not an easy task. In fact, that is what high performance computing is all about.

HPL is an implementation of the LINPACK benchmark.

2. Installation and parameters setting

We downloaded the software from http://www.netlib.org/benchmark/hpl/hpl-2.2.tar.gz, modified some compilation settings in the Make.ulysses-mkl file and compiled it.

After some tuning on a 20 cores node, we fixed the value of of N (the linear size of the matrix) and NB (number of blocks) to 64512 and 256, and (P, Q) to (4, 5). Therefore, the HPL.dat file contained the following settings:

```
HPLinpack benchmark input file
Innovative Computing Laboratory, University of Tennessee
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
0           PMAP process mapping (0=Row-,1=Column-major)
1           # of process grids (P x Q)
4           Ps
5           Qs
16.0        threshold
1           # of panel fact
2           PFACTs (0=left, 1=Crout, 2=Right)
1           # of recursive stopping criterium
4           NBMINs (>= 1)
1           # of panels in recursion
2           NDIVs
1           # of recursive panel fact.
1           RFACTs (0=left, 1=Crout, 2=Right)
1           # of broadcast
1           BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1           # of lookahead depth
1           DEPTHs (>=0)
2           SWAP (0=bin-exch,1=long,2=mix)
64          swapping threshold
0           L1 in (0=transposed,1=no-transposed) form
0           U in (0=transposed,1=no-transposed) form
1           Equilibration (0=no,1=yes)
8           memory alignment in double (> 0)
```

3. Runs and results of xhpl

We ran the compiled HPL software with the previous settings by changing the number of cores and threads per core, getting:

Cores	Threads	GFlops
20	1	419.4
10	2	414.5
5	4	403.1
4	5	402.2
2	10	407.6
1	20	400.5

We observe that the best performances are achieved by setting a low number of threads per core. In particular, the 93.6% of the theoretical peak performance is reached by setting one thread per core.

4. Results of the Intel highly optimized HPL

We ran the Intel precompiled version of HPL with the same settings of the previous case: N = 64512, NB = 256, P = 4, Q = 5. We achieved 431.7 GFlops, corresponding to the 96.4% of the theoretical peak performance.

As expected, the Intel version of HPL performs better than the one we compiled.