Exercise 6 - HPL benchmark

The following exercise is a HPL benchmark to estimate the sustained peak performance of a given multiprocessor via the resolution a linear algebra problem (namely, a linear system of equations solved via LU decomposition) for which the parallelization works really fine and it's easy to obtain the number of floating point operations in the code.

Setup

I downloaded the latest HPC benchmark (version 2.3) tar from www.netlib.org and unpacked it as in the readme.

I then copied one of the makefiles in the "setup" folder, modified the details as in the readme, updated the `ARCH` variable to `Linux_PII`, renamed the file as "Make.Linux_PII" and ran it with `make arch=Linux_PII` after having imported the modules `openmpi, mkl`.

Configuration

To configure the HPL.dat file created by the makefile in the `bin/Linux_PII` folder, I first helped myself with http://www.advancedclustering.com/act_kb/tune-hpl-dat-file/ to produce a stub of an optimal configuration.

```
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
3            # of NBs
128 192 256          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)
##### This line (no. 32) is ignored (it serves as a separator). ######
0                               Number of additional problem sizes for
PTRANS
1200 10000 30000                values of N
0                               number of additional blocking sizes for
PTRANS
40 9 8 13 13 20 16 32 64        values of NB
```

The most important numbers here are:

- # of problem sizes: number of problems to run
- Ns: side(s) of the square matrix representing the problem (problem size)
- NBs: size of the blocks (granularity of computation) - since this isn't output by the aforementioned link, we try with a couple of numbers up to 256 (as suggested in this link https://www.netlib.org/benchmark/hpl/faqs.html)
- Ps • Qs: the process grid size - must be the number of processes that will run the benchmark (P → row size, Q → column size)

Run

After having prepared the first configuration file, I ran the program on a node on Ulysses (w/ 20 cores)

```
module load openmpi

module load mkl

mpirun -np 20 ./xhpl
```

Those are the results:

```
  N   NB    GFlops
64512 128   413.17
64512 192   425.37
64512 256   423.80
```

The best sustained performance is 425.37 GFlops (with NB=192), which accounts for **~94.95%** of the theoretical peak performance (20 cores • 2.8 GHz • 8 flop/cycle = 448.00 GFlops).

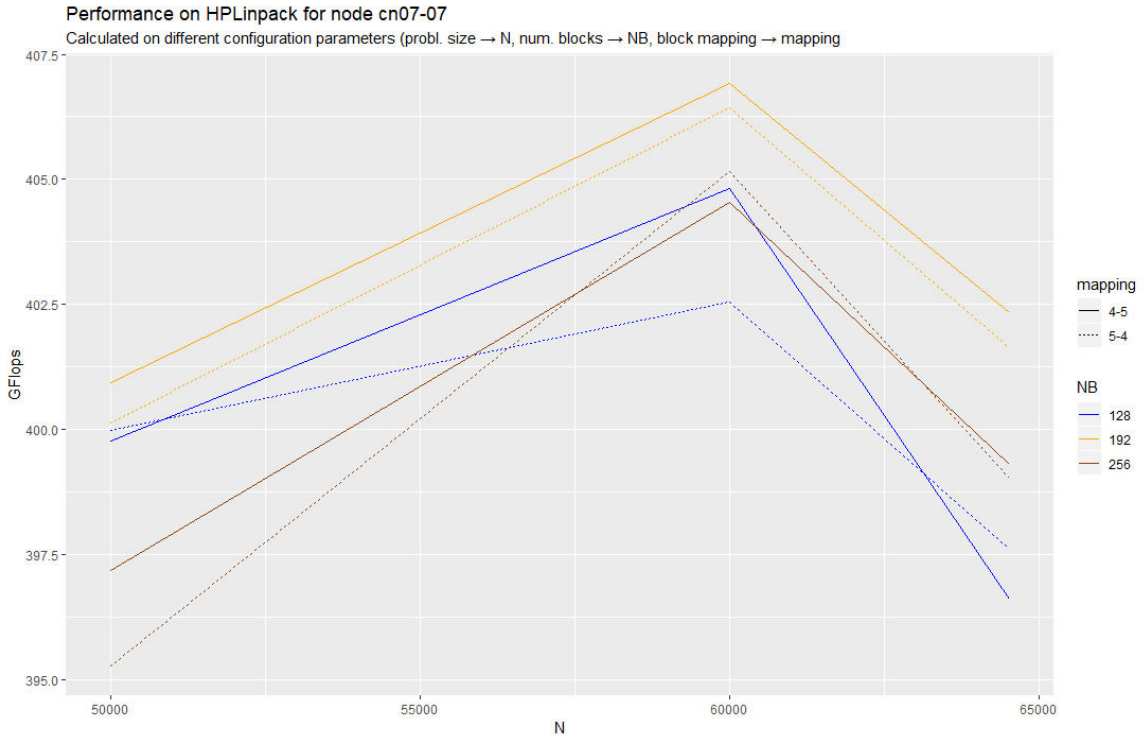
Fine-Tuning

Taking previous configuration as a starting point, I ran the program modifying HPL.dat in order to operate a *grid-search fashion* search for optimal parameters, varying values of N, NB and PxQ around the ones previously considered:

```
  N   NB    PxQ
50000 128   5-4
50000 192   5-4
50000 256   5-4
60000 128   5-4
60000 192   5-4
60000 256   5-4
64512 128   5-4
64512 192   5-4
64512 256   5-4
```

```
50000 128    4-5
50000 192    4-5
50000 256    4-5
60000 128    4-5
60000 192    4-5
60000 256    4-5
64512 128    4-5
64512 192    4-5
64512 256    4-5
```

I ran the benchmark on node cn07-07, obtaining the following results:



The solid line is realted to PxQ 4x5, the dotted line is the mapping 5x4. Blue lines is for block size of 128, orange is for block size of 192, maroon for block size of 256.

Overall, the block size suggested by the aforementioned link works best in all configurations, peaking at 406.92 GFlops for N=60.000 (lower than suggested). This accounts for 90.83% of the theoretical peak performance. The mapping 4x5 seems to work slightly better than 5x4, although for NB=256 and N=60.000, there's an inversion of the trend.

Intel Optimized LINPACK Benchmark

Next, I'm going to obtained the substained peak performance using another benchmark algorithm, this one developed by intel, which solves an analogous problem (resolution of a system of linear equations).

On Ulysses there's already a precompiled executable of this program, which I copied, along with its configurations file, on a private folder of mine.

The executable is `xlinpack_xeon64` and the configurations file is `lininput_xeon64`. Essentially, what we can toggle in the configurations is the problem size along with a couple of other variables. I configured the program to run 7 times, each time changing the problem size, which can assume those values: 50000 55000 60000 62500 64512 65000 67500.

Those are the results:

Probl. size	Performance (GFlops)
50000	429.77
55000	**431.94**
60000	426.37
62500	424.42
64512	423.81
65000	430.13
67500	423.41

With a problem size of 55.000, we obtained the best result of 431.94 GFlops, which accounts for 96.42% of the theoretical peak performance.

The result is better than the HPL benchmark: it is to be expected, since the Intel Benchmark is supposed to run in optimally on an Intel processor, while HPL is not architecture-specific.

HPL on multiple threads

Since HPL doesn't provide OMP support (no import on the makefile), we can't run the benchmark on multiple threads;

By executing the command

`OMP_NUM_THREADS=`***x*** `mpirun -np` ***y*** `./xhpl`

what we get is an execution over ***y*** processors, but on only 1 thread (no matter how many ***x*** were specified). Then, if we keep the problem size fixed, we'll have a decreasing performance (in Flops) as we decrease the number of processors (of course, if the problem size is adapted to the total memory available, that is ***y*** times the memory per core) then I expect to get roughly the same % of the theoretical peak performance, recalculated on the number of available cores, that is ***y***.

Next, the results obtained on this execution. Note that the grid configuration (PxQ) was re-adapted each time to fit ***y***, such that PxQ=***y***, and P≥Q. Problem size was fixed at 64512.

Num. processors	Performance (GFlops)	Theoretical Peak	Perf % of TPP
10	203.54	224 GFlops	90.87
5	108.62	112 " "	96.98
4	**102.72**	89.6 " "	**114.64**
2	44.43	44.8 " "	99.17
1	**26.70**	22.4 " "	**119.19**

Theoretical Peak Performance was calculated with the formula

y cores • 2.8 GHz • 8 flop/cycle

The surpassing of the T.P.P. (see **bold** lines in the table) is due to the overclock property of the Intel multiprocessors when the number of cores in use is lower than total.