# HPL Benchmark using MKL

Gabriele Sarti

November 28, 2018

**Abstract**

In this exercise we run the HPL benchmark compiled against the MKL multithread library and tune its parameters in order to get close to theoretical peak performance of a node of SISSA's Ulysses cluster.

## 1 Introduction

HPL is a software package that solves a (random) dense linear system in double precision (64 bits) arithmetic on distributed-memory computers. It is widely regarded as a portable and freely available implementation of the HPC Linpack Benchmark [1].

MKL is a library developed by Intel containing mathematical routines optimized specifically for Intel processors through threading and vectorization. Those routines include, among others, Fast Fourier Transform, Numerical and Statistical applications [2].

The theoretical peak performance is defined as the maximum possible number of floating point operations per second (FLOPS) which could be achieved inside a system. Its value is computed as follows:

$$\text{FLOPS} = \text{NumSockets} \times \text{NumCores} \times \text{FLOP/Cycle} \times \text{ClockRate}$$

On all Ulysses' nodes [3], including the one which was used to run the HPL benchmark for this exercise, the theoretical peak performance corresponds to

$$2 \times 10 \times 8 \times (2.8 \times 10^9) = 448\text{Gflops}$$

## 2 Setup Procedure

### 2.1 HPL benchmark

First of all, we installed the HPL tool and customized a Make file following the procedure specified in the assignment description. Then, we loaded the MKL library and OpenMPI for multithread execution, and created the benchmark executable, named xhpl.

```
# Start from your home directory on Ulysses
cd ~
# Check for available modules
module avail
# Download the HPL benchmark
wget http://www.netlib.org/benchmark/hpl/hpl-2.2.tar.gz
tar -xvzf hpl-2.2.tar.gz
cd hpl-2.2/setup
cp Make.Linux_Intel64 ../.
cd ..
mv Make.Linux_Intel64 Make.mkl
# Edit accordingly to the assignment description
vim Make.mkl
# Load necessary modules
module load intel/14.0
module load mkl
module load openmpi/1.8.3/intel/14.0
# Check that mkl has been loaded correctly
echo $MKLROOT
# Should print a lot of stuff
make arch=mkl
```

Listing 1: Download HPL and create Make.mkl on Ulysses

In order to achieve good performances for HPL benchmark, we tuned the auto-generated HPL.dat file with good parameters for our system:

- We are running tests on two problem sizes, the one returned by the HPL tuning website after inserting our system specifications [4] and another bigger size that gave us interesting experimental results. The problem size is chosen as roughly 80% of the memory available for the tests. In our case, the orders of the coefficient matrix A, represented by N, are 64512 and 65000.
- We tried three possible sizes for the partitioning blocking factor NB, keeping its value between 32 and 256 blocks as HPL creators suggest [5]. In our case, the selected values were 128, 192 and 256 blocks.
- Since the total number of cores on all the sockets of a node is 20, we looked for two process grid's dimensions P and Q having 20 as product and being approximately equal [5]. We tried both the couples (5,4) and (4,5) for testing purposes.

The HPL.dat file is available in Appendix A.

## 2.2 Intel benchmark

After installing the MKL library, we moved the xlinpack_xeon64 executable and the linin-put_xeon64 specifications for Intel Xeon 64-bit benchmark to the mkl folder to facilitate subsequent scripted executions of both regular and Intel HPL.

```
# Start from the same folder as before
cd ~/hpl-2.2/bin/mkl
# Load the MKL module
module load mkl
# Move to the folder containing the Intel linpack benchmark
cd $MKLROOT/benchmarks/linpack/
# Copy the executable and execution directives in the mkl directory
cp $MKLROOT/benchmarks/linpack/xlinpack_xeon64 .
cp $MKLROOT/benchmarks/linpack/lininput_xeon64 .
```

Listing 2: Setup of the Intel benchmark

After moving the two files into the mkl folder, we modified the content of lininput_xeon64 file in order to match the problem sizes of previous HPL test, setting them at respectively 64512 and 65000. We set the trials parameters to 6 for each problem size since we found out experimentally that 12 is the maximum number of trials allowed for a single run and we are interested in both average and maximal performances.

The lininput_xeon64 file is available in Appendix B.

### 2.3 Benchmarks' execution script

After having both executable files and specifications moved to the mkl and modified following our specifications, the last setup step is to create a bash script named run_mkl.sh in order to enable interactive execution using Ulysses' queue system. Our script loads the needed libraries and executes the two benchmarks using the parameters contained in the specifications.

The run_mkl.sh script is available in Appendix C.

## 3 Execution

To submit our script to Ulysses' queue system, we run the following command on our node

```
[user@node mkl] qsub -l nodes=1:ppn=20,walltime=2:00:00 -q regular run_mkl.sh
```

Our results for the HPL benchmark presented in Table 1 were computed on a single execution of the benchmark for each one of the parameters specified before. For the results of the Intel MKL Linpack benchmark we present in Table 2, we run the test six times for each problem size and computed an average performance for our tests.

It is interesting to note that while our average performance was obtained on node cn02-14 during the day, our maximal results were obtained by executing tests on node cn07-24 by night. This let us assume that performance reliability is highly tied with node and time of execution, and could be investigated further.

Table 1: HPL Benchmark Execution Results

| N | NB | P x Q | Run Time | Performance | % of Peak |
|---|---|---|---|---|---|
| 64512 | 128 | 4 x 5 | 445.06 s | 402.2 GFLOPs | 89.77 % |
| 64512 | 128 | 5 x 4 | 440.26 s | 406.6 GFLOPs | 90.76 % |
| 64512 | 192 | 4 x 5 | 428.18 s | 418.0 GFLOPs | 93.30 % |
| 64512 | 192 | 5 x 4 | 430.82 s | 415.5 GFLOPs | 92.75 % |
| 64512 | 256 | 4 x 5 | 429.91 s | 416.4 GFLOPs | 92.94 % |
| 64512 | 256 | 5 x 4 | 433.73 s | 412.7 GFLOPs | 92.12 % |
| 65000 | 128 | 4 x 5 | 440.21 s | 415.9 GFLOPs | 92.83 % |
| 65000 | 128 | 5 x 4 | 437.83 s | 418.2 GFLOPs | 93.34 % |
| 65000 | 192 | 4 x 5 | 432.53 s | 423.3 GFLOPs | 94.48 % |
| 65000 | 192 | 5 x 4 | 433.73 s | 422.1 GFLOPs | 94.21 % |
| 65000 | 256 | 4 x 5 | 430.99 s | 424.8 GFLOPs | 94.82 % |
| 65000 | 256 | 5 x 4 | 433.02 s | 422.8 GFLOPs | 94.38 % |

Table 2: Intel Benchmark Execution Results

| N | Avg. Time | Avg. Perf. | Avg. % of Peak | Max. Perf. | Max. % of Peak |
|---|---|---|---|---|---|
| 64512 | 410.55 s | 435.94 GFlops | 97.31 % | 438.84 GFlops | 97.96 % |
| 65000 | 417.67 s | 438.26 GFlops | 97.82 % | 442.05 GFlops | 98.67 % |

Execution's specifications are the following:

- OS: SISSALinux (Custom, Fedora-based) 6.5 (Bobici) (Kernel: 2.6.32-573.12.1.el6.x86_64)
- C compiler: gcc version 4.4.7 20120313 (Red Hat 4.4.7-4) (GCC)
- C flags: -fomit-frame-pointer -O3 -funroll-loops
- MPI: OpenMPI 1.8.3
- HPLinpack: Version 2.2
- BLAS: ATLAS 3.10.2

# 4 Result Analysis

From the results presented in the previous section it is evident how HPL is systematically outperformed by Intel benchmark using both problem sizes, achieving performance which are on average 20 GFlops below the average MKL ones. Those results are significant since this represents a loss of approximately 5% of our node peak performance.

Those results do not come to us as a surprise since, as HPL creators themselves affirm, "there is always room for performance improvements. Specific knowledge about a particular system is always a source of performance gains. Even from a generic point of view, better algorithms or more efficient formulation of the classic ones are potential winners." [5] This is indeed the case, given the fact that while HPL is as general as possible, Intel MKL Linpack benchmark is highly optimized for all Intel processors, including the CPU of our node.

Moreover, we note that both benchmarks have exceeded our expectations for this assignment, given that we had to achieve at least 75% of peak performance and we reached more than 90% in both cases, with a stunning 98.67% maximal performance for the Intel MKL one.

# 5 Multiprocess and Multithreaded Execution

After having assessed the performances of the system using 20 MPI processes and a single thread, we want to test the scalability of the system with respect to the HPL Benchmark by changing respectively the number of MPI processes and the number of threads used to run it. We performed a single test for each one of the combinations, obtaining the result shown in Table 3

Table 3: HPL Benchmark with multiple combinations of processes and threads. OMP Threads represent the value assigned to the variable OMP_NUM_THREADS before the execution of the benchmark. MPI Proc is the number of MPI Processes used for the execution.

| MPI Proc | OMP Threads | N | NB | $P \times Q$ | Performance | % of Peak |
|---|---|---|---|---|---|---|
| 20 | 1 | 65000 | 192 | $4 \times 5$ | 402.9 GFLOPs | 89.93 % |
| 10 | 2 | 65000 | 192 | $2 \times 5$ | 230.6 GFLOPs | 51.47 % |
| 5 | 4 | 65000 | 192 | $1 \times 5$ | 125.3 GFLOPs | 27.97 % |
| 4 | 5 | 65000 | 192 | $2 \times 2$ | 101.7 GFLOPs | 22.70 % |
| 2 | 10 | 65000 | 192 | $1 \times 2$ | 52.1 GFLOPs | 11.63 % |
| 1 | 20 | 65000 | 192 | $1 \times 1$ | 27.2 GFLOPs | 6.07 % |

There is an evident degradation of performance when reducing the number of processes, which should not be the case given the fact that we are also increasing the number of execution threads.

We analyzed the problem by the mean of the command "top" inside the execution node which was running our benchmark, and discovered that setting the variable didn't affect the

actual number of execution threads, which stayed one for all the previous runs. In order to pinpoint the source of this problem, we inspected the file Make.mkl which was provided to us for the exercise was missing the linking of the OpenMP library through the -fgomp argument, making the setting of the threading variable provided by this specific library useless.

Knowing we used a single thread all along, we computed again the % Peak values of Table 4 with respect to the relative peak performance of the processors we used. The results are presented in table

Table 4: HPL Benchmark computed by reducing the number of MPI processes gradually and keeping the number of threads constant. % of Rel. Peak is computed by multiplying the total Peak of our execution node by the proportion of cores used (E.g. for 10 MPI Proc, we have a Rel. Peak of $448 \times \frac{1}{2} = 224$ GFLOPs)

| MPI Proc | Threads | N | NB | $P \times Q$ | Performance | % of Rel. Peak |
|---|---|---|---|---|---|---|
| 20 | 1 | 65000 | 192 | $4 \times 5$ | 402.9 GFLOPs | 89.93 % |
| 10 | 1 | 65000 | 192 | $2 \times 5$ | 230.6 GFLOPs | 102.95 % |
| 5 | 1 | 65000 | 192 | $1 \times 5$ | 125.3 GFLOPs | 111.88 % |
| 4 | 1 | 65000 | 192 | $2 \times 2$ | 101.7 GFLOPs | 113.50 % |
| 2 | 1 | 65000 | 192 | $1 \times 2$ | 52.1 GFLOPs | 116.29 % |
| 1 | 1 | 65000 | 192 | $1 \times 1$ | 27.2 GFLOPs | 121.43 % |

The performances we obtained exceeded the theoretical relative peak performance as soon as we stop using all the processors inside the node for multiprocessing, as shown in Figures 1 and 2.

Figure 1: Performance of HPL Benchmark with a single thread, reducing the number of MPI Processes gradually, with respect to relative peak performance of processors on Ulysses.

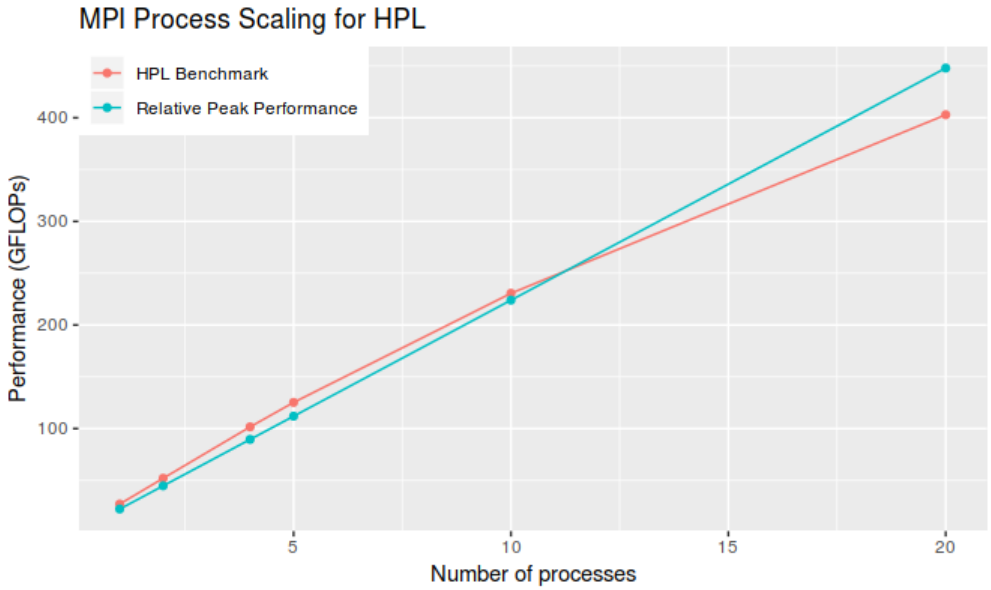
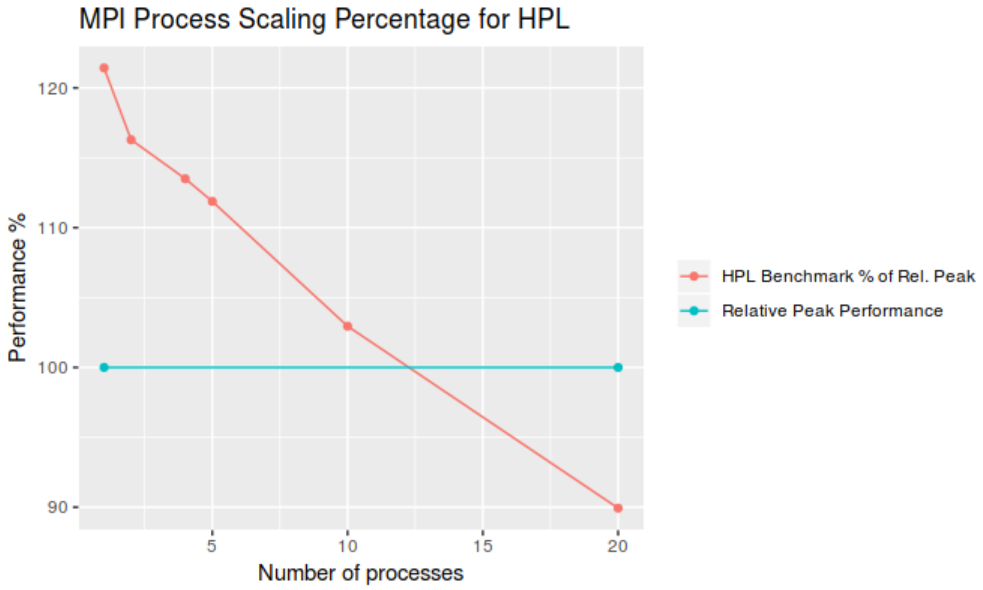


Figure 2: Percentage of relative peak performance of HPL Benchmark with a single thread, reducing the number of MPI Processes gradually, with respect to relative peak performance of processors.




This phenomenon is caused by the optimization performed inside Intel processors, which allows them to run at a higher frequency if only a part of them is used for a specific task.

# 6 Conclusions

In this exercise we showed how it is possible to obtain high performance inside a Ulysses node with respect to its theoretical peak performance, using both regular and Intel HPL Benchmarks. Due to problems in the setting up of the assignment we weren't able to observe the scaling using multiple threads, but we observed an interesting phenomenon of overclocking when only a fraction of the processors was used.

# References

[1] Dongarra, J., Luszczek, P., Petitet, A. "The LINPACK Benchmark: Past, Present, and Future," Concurrency: Practice and Experience, 15, pp. 803-820, 2003.

[2] `http://www.its.hku.hk/services/research/hpc/software/mkl`

[3] This information was given to us during the courses.

[4] `http://www.advancedclustering.com/act_kb/tune-hpl-dat-file/`

[5] `http://www.netlib.org/benchmark/hpl/faqs.html`

# Appendices

## A HPL.dat

```
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)
65000        Ns
3            # of NBs
128 192 256  NBs
0            PMAP process mapping (0=Row-,1=Column-major)
2            # of process grids (P x Q)
4 5          Ps
5 4          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
```

# B lininput_xeon64

```
Sample Intel(R) Optimized LINPACK Benchmark data file (lininput_xeon64)
Intel(R) Optimized LINPACK Benchmark data
2                    # number of tests
64512 65000 # problem sizes
64512 65000 # leading dimensions
6 6 # times to run a test
1 1 # alignment values (in KBytes)
```

# C run_mkl.sh

```
## Executing my HPL benchmark and the Intel one

# Print execution node name
/bin/hostname

# Enter in the right directory
cd hpl-2.2/bin/mkl

# Load the modules I need for execution
module load mkl
module load openmpi/1.8.3/intel/14.0

# Run the HPL benchmark
mpirun -np 20 ./xhpl

# Run the Intel benchmark
./xlinpack_xeon64 lininput_xeon64

exit
```