HPL Benchmarking using MKL Multithread Library

Rabindra Khadka

DSSC, UNITS

Overview:

HPL is a portable and freely available software package for implementation of Linpack benchmark. The number of FLOPS is measured for solving a system of linear equations. The HPL.dat file was modified to tune HPL. The file contains information about the problem sizes, machine configuration and algorithm features for executable. Therefore, the objective of the exercise was to modify and find the best combinations of problem matrix size (N), block size, parallelization matrix (P and Q) so that the performance of the machine could be tuned closer to the peak performance of the machine. In the first instance we used a simple calculator provided in the following link to generate an output file as a starting point for getting best GFLOP number from the ULYSSE node <http://www.advancedclustering.com/act_kb/tune-hpl-dat-file/>

Then the highly optimized version of HPL provided by Intel was used and was ran using the best combination received from the first procedure. The results are compared in the section below.

The theoretical peak performance for Ulysses cluster can be calculated as:

GFLOPs= 10 cores * 2 sockets *2.8 GHz * 4 *2 FLOP = 448 GFLOPS/s

Results:

```
================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR11C2R4       64512   256     4     5             425.32              4.209e+02
HPL_pdgesv() start time Fri Nov 16 17:07:14 2018

HPL_pdgesv() end time   Fri Nov 16 17:14:19 2018
```

Fig 1: Run result which yielded 420.9 GFlops

The best performance was obtained with N= 65452, Nb=256, P=4 Q=5 which is 420.9 (93.95% of the theoretical peak performance).

```
Number of equations to solve (problem size) : 64512
Leading dimension of array                  : 64512
Number of trials to run                     : 1
Data alignment value (in Kbytes)            : 1

Maximum memory requested that can be used=33295676416, at the size=64512

=================== Timing linear equation system solver ===================

Size   LDA    Align. Time(s)    GFlops   Residual     Residual(norm) Check
64512  64512  1      407.539    439.2187 3.487971e-09 2.991503e-02   pass

Performance Summary (GFlops)

Size   LDA    Align.  Average  Maximal
64512  64512  1       439.2187 439.2187
```

Fig 2: Performance result received by using Intel's Linpack Benchmark.

The highly optimized version of HPL provided by Intel gave the performance result which is at 439.21 Gflops (98.03% of the theoretical peak performance).

Threads	MPI Process	P	Q	Performance (GFLOPS)	% of Theoretical Peak Performance
1	20	4	5	421.6	94.10%
2	10	2	5	231.2	51.6%
4	5	5	1	121.4	27.09%
5	4	2	2	103.8	23.16%
10	2	2	1	57.6	12.85%
20	1	1	1	32.1	7.16%

Table 1: Multiprocessors and threads combinations

The above results in table 1 was obtained by varying the number of MPI processes and the number of threads used in each instances. The product of P and Q were kept constant to the number of MPI processes. The result received showed that the relative peak performance decreased while the MPI processes were decreased and the threads parallelized were increased. This was due to the fact the code in Make.mkl file did not allow to change the threads while executing and ran with constant thread number of 1 for all the specified MPI processes.

Summary:

The estimation of best performance of our system, we should aim for the largest possible problem size that fits the memory. As a rule of thumb 80% of the total amount of memory can be used for a good guess and avoid swapping for larger problem size which can result in performance degradation. So, amount of memory available to the process is very significant.

Similarly, the other important factor that can optimize the performance is manipulation of block size. Block sizes are used for data distribution. Load balance is better managed with small number of blocks but we need to be careful that too small block sizes can hamper the performance.

Dependency of the peak performance on number of threads parallelized by the MPI processes was clearly evident even though we didn't see the effect due to thread number held at 1 by the algorithm.

Appendix (A): HPL.dat 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
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)
```

Appendix (B): Command flow in Bash.

```
# download hpl
wget http://www.netlib.org/benchmark/hpl/hpl-2.2.tar.gz
tar -xvzf hpl-2.2.tar.gz
cd hpl-2.2
cd setup/
cp Make.Linux_Intel64 ../.
cd ..
mv Make.Linux_Intel64 Make.mkl
# Edit accordingly to the github readme for ex_6
vim Make.mkl
module load intel/14.0
module load mkl
# check it exists
echo $MKLROOT
module load openmpi/1.8.3/intel/14.0
module list
# should print a lot of stuff
make arch=mkl
cd bin/mkl/
# runs a very simple test
mpirun -np 4 ./xhpl
# 0.12 Gflops is way less than what we expected
# peak performance is around 448 Gflop/node
# we need to reach at least 75% of peak for the exercise
# This is because N is only 35, so the matrix is really small
# Increase N to be around 75% of the RAM
# See http://www.advancedclustering.com/act_kb/tune-hpl-dat-file/ with
parameters:
Nodes: 1
Cores: 20
Memory per node: 40000 MB
Number of blocks: 256

# Final results are:

N = 64512
NBs = 256
P = 4
Q = 5

# Copy the text and create a HPL.dat_large in hpl2.2/bin/mkl folder pasting the text copied in it
cat > HPL.dat_large.

vim run_mkl.sh

(in the file)
#report the name of the node where I'm executing the job
/bin/hostname

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

#load the modules i need
module load mkl
module load openmpi/1.8.3/intel/14.0

# run the code:

mpirun -np 20 ./xhpl

exit
(save and close file)

mv HPL.dat HPL.dat-small-test
cp HPL.dat_large HPL.dat
#should return nothing
diff HPL.dat HPL.dat_large

qsub -l nodes=1:ppn=20,walltime=2:00:00 -q regular run_mkl.sh

# to check if everything is correct
qstat -u <your_username>
checkjob <job_name_from_qstat>
```

```
#################
#### INTEL HPL ##
#################

#start from the same place as before, ~/hpl-2.2/bin/mkl
#check if mkl is loaded, should return an empty line
echo $MKLROOT
module load mkl
# now it returns the path
echo $MKLROOT
cd $MKLROOT
cd benchmarks/linpack/
# this file contains the execution info for intel hpl
less lininput_xeon64
# get the executable file path
pwd
cd ~/hpl-2.2/bin/mkl
# copies the executable in the mkl directory (. keeps the same name)
cp
/u/shared/programs/x86_64/mkl/11.1.3/composer_xe_2013_sp1.3.174/mkl/benchmarks/linpack/xlinpack_xe
on64 .
# copies the execution info, same as before
cp
/u/shared/programs/x86_64/mkl/11.1.3/composer_xe_2013_sp1.3.174/mkl/benchmarks/linpack/lininput_xe
on64 .
# edit lininput to test new dimensions
vim lininput_xeon64
# execution results are printed in lininput_10000.o
./xlinpack_xeon64 lininput_xeon64 > lininput_10000.o
# edit the bash script to run also the intel test by adding "./xlinpack_xeon64 lininput_xeon64"
after the previous line
vim run_mkl.sh

Final files used:

Run_mlk.sh
(begin)
## Executing my HPL benchmark and the Intel one

/bin/hostname

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

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

# run the code
mpirun -np 20 ./xhpl

# now the intel one
./xlinpack_xeon64 lininpack_xeon64
Exit
(end)
Lilinput_xeon64
(begin)
```


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

/bin/hostname

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

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

# run the code
mpirun -np 20 ./xhpl

# now the intel one
./xlinpack_xeon64 lininpack_xeon64
Exit
(end)
Lilinput_xeon64
(begin)
```


```
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
1 1 # times to run a test
1 1 # alignment values (in KBytes)
(end)

<#4969832609837219 86>un the script
qsub -l nodes=1:ppn=20,walltime=2:00:00 -q regular run_mkl.sh
```

Appendix (C): Bash scripts for running code of Linpack/HPL

```
/bin/hostname
## enter in the right dir

cd /home/rkhadka/Rabin/hpl-2.2/bin/mkl
## load the modules

module load mkl
module load openmpi/1.8.3/intel/14.0

cd bin/mkl

## run the code
./xlinpack_xeon lininput_xeon64
#mpirun -np 20 ./xhpl

exit
~
~
~
~
"run_mkl.sh" 16L, 244C                                    1,1           All
```