7th International Conference & Exposition on Petroleum Geophysics

"HYDERABAD 2008"

P - 395

Benchmarking a Seismic Data Processing HPC Cluster using HPL – A Case Study

Polash Ranjan Bora*,

GEOPIC, ONGC, Dehradun, e-mail: polashbora@rediffmail.com

Summary

With the advent of various computing servers and configurations in the rapidly changing market, benchmarking has become one of the most important aspects in selection, evaluation, installation and commissioning. Industry standard benchmarking is the need of the hour. Cluster computing has become the system of choice for executing large-scale seismic data processing applications. This is due to their low cost, high performance, off-the-shelf availability of hardware components, and freely accessible software tools that can be used for developing applications. This work is the case study of a benchmarking effort for a Seismic Data Processing HPC Cluster system using Linpack.

Introduction

The Linpack benchmark is an industry-standard benchmark created in 1979 by Jack Dongarra. The Linpack benchmark solves linear equations and uses the speed of the system under test at that task as a measure of the system's floating-point performance. Linpack reports its results in billions of floating point operations per second, or Gflops.

We used the HPL version of the Linpack Benchmark. HPL is a portable implementation of the High Performance Computing Linpack benchmark that generates, solves, checks, and times the solution process of a random dense linear system of equations. The package uses 64-bit floating point arithmetic and portable routines for linear algebra operations and message passing.

Three software components are required to run Linpack.

1. MPICH: It is a freely available, portable implementation of MPI (Message Passing Interface), a standard for message-passing for distributed-memory applications used in parallel computing. MPICH is a Free Software and is available for most flavors of Unix and Microsoft Windows.

The original implementation of MPICH is called MPICH1 and it implements the MPI-1.1 standard. As of 2006, the latest implementation is called MPICH2 and it implements the MPI-2.0 standard, but does not yet support data translations between different hardware architectures.

The version of MPICH we are using is MVAPICH2 (MPI-2 over OpenFabrics/Gen2-IB, OpenFabrics/Gen2-iWARP, uDAPL, VAPI and TCP/IP)

This is an MPI-2 implementation which includes all MPI-1 features. It is based on MPICH2 and MVICH. The latest release is MVAPICH2 1.0-beta (includes MPICH2 1.0.5p4). It is available under BSD licensing.

2. BLAS (Basic Linear Algebra Subprograms): Many numerically intensive applications make use of special libraries to perform common operations like:

- Linear algebra operators (e.g., dot products, matrix-vector multiplication)
- Fast Fourier transforms
- Linear solvers

To maximize application performance (and throughput), we want these libraries to be highly optimized for each computer architecture.

One commonly used numerical library is BLAS which:

- Contains routines that provide standard building blocks for performing basic vector and matrix operations
- Commonly used in scientific, engineering and graphics software
- High-profile since it is used with the Linpack benchmark, used to rank the fastest supercomputers in the world (Top 500 list)

We are using GotoBLAS which is an implementation of the BLAS library developed by Kazushige Goto. GotoBLAS has the following features:

- No major context switching
- Functions separated based on performance impact
- Non-performance bits written in C
- Crucial performance kernels written in assembly
- Actual assembler code is really small
- Easy to improve and debug

The General Matrix Multiply (GEMM) is a subroutine in the BLAS which performs matrix multiplication that is the multiplication of two matrices. This includes:

- SGEMM for single precision,
- DGEMM for double-precision,
- CGEMM for complex single precision, and
- ZGEMM for complex double precision.

DGEMM is one of the most widely used BLAS functions and is used in the Linpack Benchmark.

3. HPL (High Performance Linpack): It is a software package that solves a (random) dense linear system in double precision (64 bits) arithmetic on distributed-memory computers. It can thus be regarded as a portable as well as freely available implementation of the High Performance Computing Linpack Benchmark.

The HPL algorithm can be summarized by the following: Two-dimensional block-cyclic data distribution - Right-looking variant of the LU factorization with row partial pivoting featuring multiple look-ahead depths - Recursive panel factorization with pivot search and column broadcast combined - Various virtual panel broadcast topologies - bandwidth reducing swap-broadcast algorithm - backward substitution with look-ahead of depth =1.

The HPL package provides a testing and timing program to quantify the accuracy of the obtained solution as well as the time it took to compute it. The best performance achievable by this software on the system depends on a large variety of factors. Nonetheless, with some restrictive assumptions on the interconnection network, the algorithm and its implementation are scalable in the sense that their parallel efficiency is maintained constant with respect to the per processor memory usage.

Implementation

We undertook this study for 25 nodes in a HPC Cluster. The configuration of each node of the HPC cluster under study is as under:

Hardware details	
Processor	Dual Core Intel Xeon 5160
Processor Frequency	3 GHz
No. of processors	4
L2 Cache	2 MB/core
RAM	12 GB, 667 MHz
Hard drive	2 x 146 GB, 15K RPM
Network	Gigabit Ethernet
Operating System	
Name	Red Hat Enterprise Linux 4 Advanced Server Update 3
Filesystem	IBM General Parallel File System
Kernel	2.6.9-34

Table 1. Configuration of the nodes

The three software components were downloaded from the sources as given below:

1. MVAPICH2-0.9.8
Source: http://nowlab.cse.ohio-state.edu/projects/mpi-iba/
File: mvapich2-0.9.8p3.tar.gz

2. GotoBLAS
Source: http://www.tacc.utexas.edu/resources/software/
File: GotoBLAS-1.10.tar.gz

3. HPL
Source: http://www.netlib.org/benchmark/hpl/
File: hpl.tgz

For all software components, we compiled the components from source. We then created our work directory which was available to all the systems in the HPC cluster. In our case it was an IBM General Parallel Filesystem which made things simpler. To simplify the description below, we will refer to the working directory as $LIN_DIR. Three directoties were created as $LIN_DIR/mpi, $LIN_DIR/GotoBLAS and $LIN_DIR/hpl,

To build MVAPICH2, we did the following:
Unpacked the MVAPICH2-0.9.8.tar.gz archive in $ LIN_DIR/mpi. Created the binaries and libraries using make.mvapich2.tcp. Created the .mpd.conf which will contain a secret password in the variable called mpd_secret_word. This file must have permissions 600. Created a host file (eg. myhosts). We will call this file $HFILE. We put computing hostnames sequentially in the file. Rsh was enabled to run in all the nodes without password. The $LIN_DIR/mpi/bin directory was added to the PATH variable. To test mpi we adopted the following procedure.

To start mpi daemon

```
% mpdboot --rsh=rsh -n 10 -f myhosts
```

To check the status

```
%mpdtrace
node1
node2
…..
```

To kill mpi daemon

```
%mpdallexit
```

To build GotoBLAS, we did the following:
Unpacked the GotoBLAS-1.10.tar.gz archive in $LIN_DIR/GotoBLAS directory.

Edited the Makefile.rule file and made the following changes:

```
BINARY64=1
SMP=1
MAX_THREADS=4 (number of cores/node)
```

Ran quickbuild.64bit which created the necessary files.
Copied the libgoto_core2p-r1.10.a file to $LIN_DIR/hpl.

To build HPL, we did the following:
Unpacked the hpl.gz archive in $LIN_DIR/hpl. Created the Make.em64t file, as shown below.

```
--------------------------------------------------------------------------
SHELL        = /bin/sh
CD           = cd
CP           = cp
LN_S         = ln -s
MKDIR        = mkdir
RM           = /bin/rm -f
TOUCH        = touch
ARCH         = em64t
TOPdir       = $LIN_DIR/hpl
INCdir       = $(TOPdir)/include
BINdir       = $(TOPdir)/bin/$(ARCH)
LIBdir       = $(TOPdir)/lib/$(ARCH)
HPLlib       = $(LIBdir)/libhpl.a
MPdir        = $LIN_DIR /mpi
MPinc        = -I$(MPdir)/include
MPlib        = -L$(MPdir)/lib $(MPdir)/lib/libmpich.a
LAdir        = $LIN_DIR /hpl
LAinc        =
LAlib        = $(LAdir)/libgoto_core2p-r1.10.a
HPL_INCLUDES = -I$(INCdir) -I$(INCdir)/$(ARCH) $(LAinc) $(MPinc)
HPL_LIBS     = $(HPLlib) $(LAlib) $(MPlib)
HPL_DEFS     = $(F2CDEFS) $(HPL_OPTS) $(HPL_INCLUDES)
CC           = $(MPdir)/bin/mpicc
CCNOOPT      = $(HPL_DEFS)
CCFLAGS      = $(HPL_DEFS) -fomit-frame-pointer -O3 -funroll-loops -
               Bstatic
LINKER       = $(MPdir)/bin/mpicc
LINKFLAGS    = $(CCFLAGS) -lm -B static
ARCHIVER     = ar
ARFLAGS      = r
RANLIB       = echo
--------------------------------------------------------------------------
```

List 1. Makefile for HPL compilation

Ran make arch=em64t which will create the xhpl executable as $LIN_DIR/hpl/bin/em64t/xhpl. A file HPL.dat will also be available here. This is the input file for xhpl. We will refer to $LIN_DIR/hpl/bin/em64t as $HPL_BIN

Running HPL

We ran the benchmark using MPI so that this runs in parallel in all the 25 nodes. The input to the benchmark program is the HPL.dat file which contains the key parameters as shown below:

```
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)
3            # of problems sizes (N)
170000 180000  Ns
2            # of NBs
90 130       NBs
1            PMAP process mapping (0=Row-,1=Column-major)
3            # of process grids (P x Q)
5  6 25      Ps
5  4  1      Qs
16.0         threshold
3            # of panel fact
0 1 2        PFACTs (0=left, 1=Crout, 2=Right)
2            # of recursive stopping criterium
2 4          NBMINs (>= 1)
1            # of panels in recursion
2            NDIVs
3            # of recursive panel fact.
0 1 2        RFACTs (0=left, 1=Crout, 2=Right)
1            # of broadcast
0            BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1            # of lookahead depth
0            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)
```

List 2. HPL.dat

The system load has to be controlled by the parameters in the HPL.dat file. The appropriate parameters can increase the problem size such that we get the best performance. It is important to not overload the system and avoid disk swapping. A brief description of the parameters and their tuning is as under.

NB is the block size for job distribution. NB should be a number between 80 and 256.

N is the problem size. N should be sqrt(Total mem in GB * 128) *1024*(% memory to use).

PxQ is the processor grid. PxQ should be equal to the total number of nodes. If we have 25 nodes PxQ can be 5x5, 25x1 etc.

Since N can be theoretically computed, we need to run some preliminary test to determine best PxQ and NB.We have done a test with block sizes NB = 90, 100, 110, 120, 130, 140, 150, 160; problem size N = 7000, so that we get fast results and processor grids with 5x5, 6x4 and 25x1.

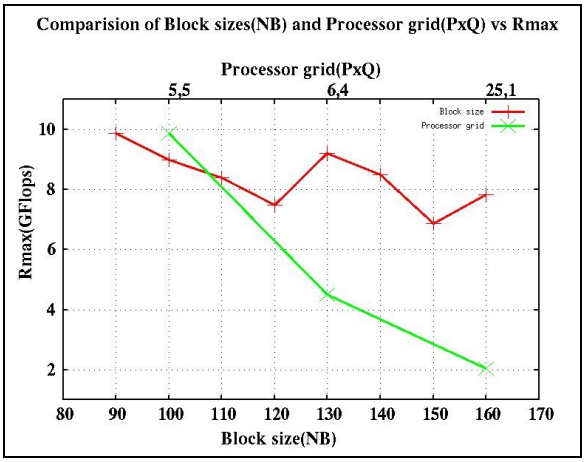

Figure 1. Choosing the best NB and PxQ

As shown in the Figure 1. above it is seen that we get best results with NB=90,130 and PxQ=5x5. Since we have a total memory of 12x25 GB our N = sqrt(12GBx25x128)x1024x90% = 180000 approximately. This is reflected in the parameters given in List 2.

Running the benchmark

To start mpi

```
% mpdboot --rsh=rsh -n 25 -f myhosts
```

To check mpi

```
% mpdtrace
```

We placed the HPL.dat file in $HOME where $HOME is home directory of the user.

```
% cp $LIN_DIR/hpl/bin/em64t/HPL.dat $HOME
```

Run hpl

```
% mpirun –machinefile $HFILE -n 25 $HPL_BIN/xhpl
```

Here $HFILE is the file containing the nodelist and $HPL_BIN is the directory where xhpl is located.

The results of this test is described below:

```
- The matrix A is randomly generated for each test.
- The following scaled residual checks will be computed:
  1) ||Ax-b||_oo / ( eps * ||A||_1  * N        )
  2) ||Ax-b||_oo / ( eps * ||A||_1  * ||x||_1  )
  3) ||Ax-b||_oo / ( eps * ||A||_oo * ||x||_oo )
- The relative machine precision (eps) is taken to be         1.110223e-16
- Computational tests pass if scaled residuals are less than          16.0
================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WC00L2L2      160000    90     5     5           11901.61              2.294e+02
--------------------------------------------------------------------------------
```

```
||Ax-b||_oo / ( eps * ||A||_1  * N      ) =        0.0038828 ..... PASSED
||Ax-b||_oo / ( eps * ||A||_1  * ||x||_1  ) =        0.0021346 ..... PASSED
||Ax-b||_oo / ( eps * ||A||_oo * ||x||_oo ) =        0.0003669 ..... PASSED
================================================================================
T/V              N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WC00L2L4      160000   90     5     5             12833.03             2.128e+02
--------------------------------------------------------------------------------
||Ax-b||_oo / ( eps * ||A||_1  * N      ) =        0.0039246 ..... PASSED
||Ax-b||_oo / ( eps * ||A||_1  * ||x||_1  ) =        0.0021576 ..... PASSED
||Ax-b||_oo / ( eps * ||A||_oo * ||x||_oo ) =        0.0003709 ..... PASSED
================================================================================
T/V              N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WC00L2C2      160000   90     5     5             12193.98             2.239e+02
--------------------------------------------------------------------------------
||Ax-b||_oo / ( eps * ||A||_1  * N      ) =        0.0038828 ..... PASSED
||Ax-b||_oo / ( eps * ||A||_1  * ||x||_1  ) =        0.0021346 ..... PASSED
||Ax-b||_oo / ( eps * ||A||_oo * ||x||_oo ) =        0.0003669 ..... PASSED
================================================================================
T/V              N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WC00L2C4      160000   90     5     5             11859.68             2.303e+02
--------------------------------------------------------------------------------
||Ax-b||_oo / ( eps * ||A||_1  * N      ) =        0.0039246 ..... PASSED
||Ax-b||_oo / ( eps * ||A||_1  * ||x||_1  ) =        0.0021576 ..... PASSED
||Ax-b||_oo / ( eps * ||A||_oo * ||x||_oo ) =        0.0003709 ..... PASSED
================================================================================
T/V              N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WC00L2R2      160000   90     5     5             12092.54             2.258e+02
--------------------------------------------------------------------------------
||Ax-b||_oo / ( eps * ||A||_1  * N      ) =        0.0038828 ..... PASSED
||Ax-b||_oo / ( eps * ||A||_1  * ||x||_1  ) =        0.0021346 ..... PASSED
||Ax-b||_oo / ( eps * ||A||_oo * ||x||_oo ) =        0.0003669 ..... PASSED
================================================================================
T/V              N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WC00L2R4      160000   90     5     5             12065.59             2.263e+02
--------------------------------------------------------------------------------
||Ax-b||_oo / ( eps * ||A||_1  * N      ) =        0.0042211 ..... PASSED
||Ax-b||_oo / ( eps * ||A||_1  * ||x||_1  ) =        0.0023206 ..... PASSED
||Ax-b||_oo / ( eps * ||A||_oo * ||x||_oo ) =        0.0003989 ..... PASSED
================================================================================
T/V              N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WC00C2L2      160000   90     5     5             12264.33             2.227e+02
--------------------------------------------------------------------------------
||Ax-b||_oo / ( eps * ||A||_1  * N      ) =        0.0038828 ..... PASSED
||Ax-b||_oo / ( eps * ||A||_1  * ||x||_1  ) =        0.0021346 ..... PASSED
||Ax-b||_oo / ( eps * ||A||_oo * ||x||_oo ) =        0.0003669 ..... PASSED
================================================================================
```

List 3. Results of the benchmark test

The results in List 3 above can be summarized as:

Test Name	Result (GFlops)
WR00L2L2	229.4
WR00L2L4	212.8
WR00L2C2	223.9
WR00L2C4	230.3
WR00L2R2	225.8
WR00L2R4	226.3
WR00C2L2	222.7

Table 2. Results of the benchmark test

Interpreting the results:

Rpeak is the maximum theoretical peak achievable by the system. The formula to compute this is = No. of cores * CPU clock speed * Floating point issue rate.

Rpeak for the system under study is 2 FLOP/cycle (w/ SSE2) * 3 GHz * 25 nodes * 4 cores/node = 600 GFlops.

As we can see in the sample output above the system ran the fastest computation at 230.3 Gflops. This test will give total of 864 such values. We have to take the highest of these values that will be the Rmax for this system.

The goal of running Linpack is to get the Rmax as near as Rpeak as possible. CPU architecture and performance, network interconnect architecture and bandwidth, memory distribution, cache performance all play a vital role in the results of Linpack.

The configuration of the system also plays a major role in the results as well. With Ethernet interconnects, we get 50-60% of Rpeak. Myrinet or Infiniband will improve this to 75-85%.

Conclusions

The results show that the system under study is achieving an Rmax of 230.3 Gflops against a Rpeak of 600 GFlops. Thus our Rmax is almost 38% of Rpeak. This is a fair result keeping in mind the Gigabit Ethernet backbone on which the system is based. As indicated earlier with Gigabit Ethernet we can achieve an Rmax which is 50-60% of Rpeak. This also demonstrates there is scope for tuning the system. Linpack HPL is a reassurance and vital cross check for system engineers before running the number crunching parallel seismic processing programs.

Acknowledgements

The author gratefully acknowledges the facilities provided at GEOPIC, ONGC computer center for carrying out the benchmarks. He expresses his deepest gratitude to SK Das, Head-GEOPIC, RK Verma, Head-CSSW, TR Murali Mohan and P Manoharan for their invaluable support and guidance without which this work would not have been possible.

The views expressed in this work are solely of the author and does not necessarily reflect those of ONGC.

References

Pummill, J and Bui, H, HPL Student Guide, University of Arkansas

Kandadai, S.N., Tuning tips for HPL on IBM xSeries Linux Clusters, Redbooks Paper

Principled technologies test report: Linpack Performance on RHEL 5.1 and 3 AS Intel Servers

http://www.netlib.org/benchmark/hpl/faqs.html
http://www.netlib.org/benchmark/hpl/tuning.html
http://www.netlib.org/benchmark/hpl
http://www.cs.utexas.edu/users/flame/goto
http://nowlab.cse.ohio-state.edu/projects/mpi-iba/
http://www.tacc.utexas.edu/resources/software/