HPC High Performance Linpackfor AMD® Opteron™ 6200 Series processors

Tags:HPC, ACML, Tools, Performance Libraries

Introduction

This paper describes a method for installing and running the High Performance Linpack (HPL) benchmark on servers using AMD "Bulldozer" processors, especially on servers using AMD Opteron™ 6200 Series processors. This is an updated version of an article which focused on earlier versions of Opteron processors. The earlier version can be viewed here http://developer.amd.com/documentation/articles/pages/HPCHighPerformanceLinpack.aspx. Please note that current versions of the ACML library should be used instead of the ACML versions referenced in any earlier papers.

The performance testing exercise discussed here uses SUSE Linux® Enterprise Server 11, Service Pack 2. This Operating System will exhibit good behavior from a NUMA / processor affinity standpoint, and features a readily available set of compiler tools.

Note that a set of HPL binary and script downloads are now available on the ACML download page. This paper will document the installation process using slightly different directories.

First, the evaluation DVD was obtained from http://download.novell.com. It was installed on thissystem:

- Two Socket server using AMD Opteron 6274 processor (thirty-two cores cores)

Before compiling HPL, you'll need to obtain some tools:

- ACML (the AMD Core Math Library). You will use ACML Version 5.1.0, and test its performance on an AMD Opteron™6200 Series processor-based server.
- MPI library - a good choice will be OpenMPI. Instructions will be provided for installing OpenMPI.
- Open64 compiler– This compiler can be obtained at http://developer.amd.com/tools/open64/Pages/default.aspx

HPC Concepts

•» Introduction to HPC

•» HPC Parallel Programming Models

•» HPC Programming Tools

•» Hardware for Parallel Computing

Hands On Exercises

•» Getting Started

•» Tutorials

More Information

•» Related Resources

•» Frequently Asked Questions

Downloading and installing ACML

ACML can be downloaded from:

http://developer.amd.com/acml

Currently ACML Version 5.1.0 is available. After browsing to the site above, find the download page.

Download from the OPEN64 group:
`acml-5-1-0-open64-64bit.tgz`

As root, install ACML using the below commands:

```
tar xzvf acml-5-1-0-open64-64bit.tgz
./install-acml-5-1-0-open64-64bit.sh
```

The install script will prompt you to accept the license agreement. The default installation locations will be in the directory `/opt/acml5.1.0`.

NOTE: If installing to the default location, you might want to change ownership of the ACML directory and it's subdirectories to that of a non-privileged user. This will allow that user to run the examples without needing root privilege. Otherwise, you might want to install into a user account subdirectory.

NOTE: It's a good idea to run the ACML examples to ensure that the package is installed correctly and that the compiler environment is correct. Just go to the example directory and type `make`. Several C and Fortran examples should build and run. That's all you need to do for ACML.

NOTE: The Gfortran version of ACML can also be used. In this case it is necessary to use the 4.6.2 version of GCC/Gfortran (or newer). Performance with Open64 is slightly better than with Gfortran.

Downloading and installing OpenMPI

Though an MPI may come with the Linux distribution, we want an MPI implementation that has great support for NUMA architectures and can be tweaked for a few other optimizations.

A good open-source choice for MPI is OpenMPI. OpenMPI can be downloaded from:

http://www.open-mpi.org (click on the "Download" link in the "Open MPI Software" menu on the left)

Excellent instructions are provided on that website. There is a general FAQ and information on how to build here:

http://www.open-mpi.org/faq/

http://www.open-mpi.org/faq/?category=building

However we'll provide some specific instructions as well. In this case, you'll install the MPI libraries to our home directory, assuming "hpcuser" is the home user. Otherwise, one would need to log in as root since by default OpenMPI will go into /usr/local. At a command prompt:

```
cd ~
wget http://www.open-mpi.org/software/ompi/v1.5/downloads/openmpi-
1.5.5.tar.bz2
tar xjfopenmpi-1.5.5.tar.bz2
mkdir /home/hpcuser/openmpi-install
cd /home/hpcuser/openmpi-install
/home/hpcuser/openmpi-1.5.5/configure --help
/home/hpcuser/openmpi-1.5.5/configure --prefix=/home/hpcuser/openmpi-install
2>&1 | tee configure.log
```

If you do not want a parallel build, remove the "-j 20":

```
make -j 20 2>&1 | tee make.log
```

```
make install 2>&1 | tee make_install.log
```

We could use more cores for building on our 32 core machines, but OpenMPI doesn't take too long to build.

Alternatively make and install can be combined with:

```
# make all install -j 20 2>&1 | tee make_all_install.log
```

Now, to use this MPI implementation, you would simply modify your `PATH`as shown below.

To make sure OpenMPI will work correctly, and to make sure it will work on multiple nodes, make sure you add statements like the following to your `.bashrc` file:

```
export PATH=/home/hpcuser/openmpi-install/bin:${PATH}
export LD_LIBRARY_PATH=\
/home/hpcuser/openmpi-install/lib:${LD_LIBRARY_PATH}
```

After installing OpenMPI on multiple nodes, you can very quickly validate OpenMPI like this:

```
mpirun -np2 -host host1,host2 -np2 /bin/ls
```

If this fails, it may be because you did not modify `PATH`and `LD_LIBRARY_PATH`, in the `.bashrc` or `/etc/profile.local` on all nodes. In that case you may see errors like the following:

```
bash: orted: command not found
```

```
[node1:11985] [0,0,0] ORTE_ERROR_LOG: Timeout in file /home/hpcuser/openmpi-
1.5.5/orte/mca/pls/base/pls_base_orted_cmds.c at line 275
```

```
[node1:11985] ERROR: A daemon on node node2 failed to start as expected.
```

Compiling and Running HPL - The Basics

The High Performance Linpack software is obtainable from:

http://www.netlib.org

It's also bundled as part of the HPC Challenge collection of software. In that case, they've added a few extra Makefiles illustrating various recent platforms.

Basic installation instructions can be found here:http://www.netlib.org/benchmark/hpl

Following is a set of commands to get going with HPL:

```
cd /home/hpcuser
wget http://www.netlib.org/benchmark/hpl/hpl-2.0.tar.gz
tar xzvf hpl-2.0.tar.gz
cd hpl-2.0
cp setup/Make.Linux_ATHLON_FBLAS make.open64_acml
vi Make.open64_acml
```

1. Change the line with

```
ARCH            = Linux_ATHLON_FBLAS
```

To something like (this can be any name you want):

```
ARCH            = open64_acml
```

2. Change the line with

```
TOPdir        = $(HOME)/hpl
```

as required to point to the correct directory.

3. Change this line:

```
CC            = /usr/bin/gcc
```

to:

```
CC            = opencc
```

4. Change this line:

```
LINKER        = /usr/bin/g77
```

to:

```
LINKER        = openf90
```

5. For OpenMPI, change these default lines:

```
MPdir         = /usr/local/mpi
MPinc         = -I$(MPdir)/include
MPlib         = $(Mpdir)/lib/libmpich.a
```

to:

```
MPdir         = /home/hpcuser/openmpi-install
MPinc         = -I$(MPdir)/include
MPlib         =-L$(MPdir)/lib -lmpi
```

6. For ACML 5.1.0, change these lines:

```
LAdir         = $(HOME)/netlib/ARCHIVES/Linux_ATHLON
LAinc         =
LAlib         = $(LAdir)/libf77blas.a $(Ladir)/libatlas.a
```

to:

```
LAdir         = /opt/acml5.1.0/open64_64_fma4
LAinc         = -I$(LAdir)/include
LAlib         = $(LAdir)/lib/libacml.a
```

Note the use of the static ACML library. Do not use the shared object library for HPL since this may result in very poor performance.

7. Last, change the default C compiler flags

```
CCFLAGS      = $(HPL_DEFS) -fomit-frame-pointer -O3 -funroll-loops -W -Wall
```

to:

```
= $(HPL_DEFS) -O3
```

Now do the compile of HPL:

```
cd /home/hpcuser/hpl-2.0

make -j 8 arch=open64_acml
```

The example above only uses 8 cores for the build. You can use more than eight cores for the build, but HPL builds very fast even with just one processor.

NOTE: Remember for subsequent builds you will need to clean out the objects as in the comments below:

```
# When changing compiler and link options,
# clean out the objects before rebuilding:
# make clean arch=open64_acml
```

To run HPL with OpenMPI, you can use the following commands – and do not forget to set the `PATH` variable!

```
export PATH=/home/hpcuser/openmpi-install/bin:${PATH}
export LD_LIBRARY_PATH=/home/hpcuser/openmpi-install/lib
cd /home/hpcuser/hpl/bin/open64_acml
# Do this however you want; this way lets one know
# exactly what HPL.dat generated what output.
cat HPL.dat >xhpl.out
mpirun -np 4 ./xhpl>>xhpl.out
```

In the example above, `np4` means to run this distributed workload on 4 processors. Running the default `HPL.dat` should take no more than a few seconds. The default output simply goes to `stdout,` which redirects to a file.

Next, examine xhpl.out for the results. If it has run correctly, with the default HPL.dat, then you should see that it has run 864 tests. It'll say that they've passed residual checks.

To glance at the performance, simply:

```
grep WR xhpl.out
```

The far left-hand field "`T/V`" is a combination of characters that describe the possible parameter permutations. Other parameters, "`N`", "`NB`", "`P`", and "`Q`" are enumerated in separate columns. Then the performance metrics; the meaning of "`Time`" is obvious, and it is in seconds. Finally, the metric everyone is interested in is "`Gflops`" on the far right. This is an aggregated throughput metric based on the estimated number of floating point calculations for a particular set of parameters, divided by time: gigaflops per second.

To make it more realistic, modify HPL.dat as follows:

1. Change line 6 ("Ns", controlled by preceding line 5), from:

```
29 30 34 35 Ns
```

to:

```
10000 15000 20000 30000 Ns
```

2. Change lines 7 and 8 ("# of NBs", "NBs" -- block sizes and how many) from:

```
4            # of NBs
1 2 3 4       NBs
```

to:

```
3            # of NBs
100 92 104         NBs
```

To simplify what you arelooking at, reduce the possible permutations of how the workload is distributed and factorized.

3. Change lines 10-12 ["# of process grids (P X Q)"] from:

```
3
2 1 4        Ps
2 4 1        Qs
```

to:

```
1
4            Ps
8            Qs
```

The HPL.dat file should look like this:

```
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)
4          # of problems sizes (N)
10000 15000 20000 30000 # Ns
3            # of NB
100 92 104    # NBs
0            PMAP process mapping (0=Row-,1=Column-major)
1            # of process grids (P x Q)
4            Ps
8            Qs
16.0         threshold
1            # of panel fact
2 1        PFACTs (0=left, 1=Crout, 2=Right)
1            # of recursive stopping criterium
4 2          NBMINs (>= 1)
1            # of panels in recursion
2 4          NDIVs
1            # of recursive panel fact.
2 1 0             RFACTs (0=left, 1=Crout, 2=Right)
1            # of broadcast
```

```
1 3           BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1             # of lookahead depth
0 1 2         DEPTHs (>=0)
1             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)
```

Now rerun:

```
cat HPL.dat > xhpl_2.out ; mpirun -np32 ./xhpl>> xhpl_2.out
```

This will take several minutes. The output for this simple run is included at the end of this paper.

In this case, np32 means to run this distributed workload on 32 processors. To make this work, the product of PxQ set in HPL.dat would need to be 32, hence the 4 and 8.

A setting of np32 would be suitable for a two socket server using AMD Opteron™6200 Series processors.

This HPL.dat file is a suitable starting point for current AMD Opteron™6200 Series processor-based servers. The parameters have been chosen to provide good performance on such systems. You can experiment with combinations of these parameters to see how they affect performance. The basic methodology is to systematically try different parameters to see which ones provide the highest Gflops result. In general, do so with small problems (~10000) which run quickly. You can run a lot of experiments in short time this way. Refer to the tuning guide on the HPL web page for more information. Another useful reference is this description of the HPL benchmark. It helps to know how the code works when trying to tune it.

Example Results

Here are the results of running HPL on a Dinar platform with the HPL.dat parameters described above. These performance results are for smaller problems and are not representative of what can be achieved with larger problems.

Note that slightly better performance can be obtained for benchmarking and stress test purposes by turning off APM mode in the BIOS settings. This will force the processor to remain in the primary voltage/frequency state. Among the benefits of this is more consistency in the results. A similar BIOS change is to enable HPC mode if it is available. HPC mode sets most of the power management P-States to the same value, while still allowing higher frequency Boost P-States. If your BIOS does not have a HPC mode option, a simple hpcmode clock configuration utility can be downloaded from http://developer.amd.com/libraries/acml/downloads/pages/default.aspx. For a full description of the Advanced Power Management features of the AMD Opteron™ 6200 Series processors, refer to the BIOS and Kernel Developers Guide (BKDG) for AMD Family 15h Models 00h-0fh processors.

Notice how the choice of NB makes a big difference in performance. The best number, 100, is specific to ACML on these processors, other NBs will be used with other combinations of libraries and CPUs. Ideally, N should be a even multiple of NB.

```
mpirun -n 32 xhpl
================================================================================
HPLinpack2.0  --  High-Performance Linpack benchmark  --   September 10, 2008
Written by A. Petitet and R. Clint Whaley, Innovative Computing Laboratory,
UTK
Modified by Piotr Luszczek, Innovative Computing Laboratory, UTK
Modified by JulienLangou, University of Colorado Denver
================================================================================
```

```
An explanation of the input/output parameters follows:
T/V    : Wall time / encoded variant.
N      : The order of the coefficient matrix A.
NB     : The partitioning blocking factor.
P      : The number of process rows.
Q      : The number of process columns.
Time   : Time in seconds to solve the linear system.
Gflops : Rate of execution for solving the linear system.

The following parameter values will be used:
N      :   10000    15000    20000    30000
NB     :      92      104      100
PMAP   : Row-major process mapping
P      :       4
Q      :       8
PFACT  :   Right
NBMIN  :       4
NDIV   :       2
RFACT  :   Right
BCAST  :  1ringM
DEPTH  :       0
SWAP   : Spread-roll (long)
L1     : transposed form
U      : transposed form
EQUIL  : yes
ALIGN  : 8 double precision words
--------------------------------------------------------------------------------
- The matrix A is randomly generated for each test.
- The following scaled residual check will be computed:
      ||Ax-b||_oo / ( eps * ( || x ||_oo * || A ||_oo + || b ||_oo ) * N )
- 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
--------------------------------------------------------------------------------
WR01R2R4       10000    92     4     8               4.28              1.558e+02
--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=        0.0035143 ... PASSED
================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR01R2R4       10000   104     4     8               4.34              1.536e+02
--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=        0.0035900 ... PASSED
================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR01R2R4       10000   100     4     8               4.28              1.557e+02
--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=        0.0031279 ... PASSED
================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR01R2R4       15000    92     4     8              12.63              1.781e+02
--------------------------------------------------------------------------------
```

```
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=        0.0033686 ... PASSED
================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR01R2R4       15000   104     4     8             12.92           1.742e+02
--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=        0.0029765 ... PASSED
================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR01R2R4       15000   100     4     8             12.59           1.787e+02
--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=        0.0030555 ... PASSED
================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR01R2R4       20000    92     4     8             28.21           1.891e+02
--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=        0.0025704 ... PASSED
================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR01R2R4       20000   104     4     8             28.75           1.855e+02
--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=        0.0027287 ... PASSED
================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR01R2R4       20000   100     4     8             27.93           1.909e+02
--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=        0.0023810 ... PASSED
================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR01R2R4       30000    92     4     8             88.42           2.036e+02
--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=        0.0029272 ... PASSED
================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR01R2R4       30000   104     4     8             90.17           1.996e+02
--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=        0.0025377 ... PASSED
================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR01R2R4       30000   100     4     8             87.86           2.049e+02
--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=        0.0025044 ... PASSED
================================================================================

Finished     12 tests with the following results:
             12 tests completed and passed residual checks,
              0 tests completed and failed residual checks,
              0 tests skipped because of illegal input values.
--------------------------------------------------------------------------------
End of Tests.
```