IndySCC SC23

Overview of the HPL Benchmark

Junjie Li
Texas Advanced Computing Center
The University of Texas at Austin

Benchmarks

- Application benchmark
  - Your favorite applications, SPEC suites, MLPerf suites, etc….
- Proxy app (mini app)
  - miniMD, miniDFT, Cloverleaf, Tealeaf, miniFE, etc..
  - Exascale project: https://proxyapps.exascaleproject.org/app/
  - Mantevo project: https://mantevo.github.io
- Kernel benchmark
  - HPL, HPCG, FFT, etc..

What is HPL?

- First Linpack benchmark appeared in 1979
- High Performance Linpack (HPL) for parallel computers.
  - source code: https://www.netlib.org/benchmark/hpl/index.html
- Solves dense linear algebra equation Ax=b in double precision (FP64).
- Reports FP64 floating point operations per second (FLOPS or flop/s).
- Top500 listing of world's fastest supercomputers ranked by HPL.

About HPL

- Performance dominated by dgemm operation from BLAS.
- Moderately depends on MPI P2P
  - Interconnect performance matters
  - Infiniband or equivalent is highly favored
- Scales to arbitrarily sized systems
  - adjustable problem size
  - Single core to Top500 size

Profile by Function
(768 MPI ranks, Haswell CPU 24C/node, Aries interconnect)

Samp%	Samp	Imb. Samp	Imb. Samp%	Group Function PE=HIDE
100.0%	292,803.4	--	--	Total
88.6%	259,534.9	--	--	BLAS
86.7%	253,956.4	9,168.6	3.5%	__sci_dgemm_
10.3%	30,145.0	--	--	MPI
5.3%	15,624.1	8,255.9	34.6%	MPI_Iprobe
2.6%	7,674.9	1,293.1	14.4%	MPI_Recv
2.0%	5,931.5	533.5	8.3%	MPI_Send
1.1%	3,098.5	--	--	USER

About HPL

- Algorithmic Intensity (A.I.) = $\frac{\text{floating pointing operations}}{\text{memory movement}}$
- A.I. of Matrix Multiplication C = A x B :
  - $C = \begin{bmatrix} \vdots & \dots \\ A_{i1} & \vdots \quad A_{in} \\ \vdots & \dots \end{bmatrix}_{N\times N} \times \begin{bmatrix} \dots & B_{1j} \\ \vdots & \vdots \\ \dots & B_{nj} \end{bmatrix}_{N\times N}$
  - $C_{ij} = \sum_{k=1}^{N} A_{ik} B_{kj}$: N multiplication and N-1 addition
  - Total FP operations N*N*(2N-1) = $2N^3-N^2$ → $O(N^3)$
  - Total memory operations (read A and B, write C): 3*N*N → $O(N^2)$
  - A.I. ~ O(N)
  - Heavily compute bound

Rpeak & Rmax

- Rpeak: theoretical Flops
  - Rpeak = ncores * freq * flops/cycle
  - Flops/cycle (Haswell)= 256/64 * 2 * 2 = 16
    vector length FMA FP unit
- Rmax: achieved Flops by HPL

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,699,904	1,194.00	1,679.82	22,703
2	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	537.21	29,899
3	**LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	2,220,288	309.10	428.70	6,016

Why HPL

A great benchmark with limitations:

The Pros

- Simple to run
- Highly tunable
- Great scalability
- Simple metrics, easy to users and marketing
- Represent widely used kernel in scientific computing

The Cons

- Only stress the flops
- Only represents a single kernel
- Highly tuned result may not represent performance of real apps
- Not measuring end-to-end performance

Build HPL

SC23 Denver, CO | i am hpc.

Use autotools

- Download and untar the HPL source tarball
- tar -xf hpl-2.3.tar.gz
- Need to load the “intel-mkl” libraries for BLAS routines
- Verify that you have all the dependencies (compiler, MPI, BLAS)
- Now compile HPL
  - ./configure --prefix= ...
  - make -j16 && make install
- Add the binary location to your PATH
- export PATH=/path/to/xhpl/binary:$PATH
- Ready to run

Or edit Makefile

- $HPLROOT/setup/Make.$arch
- Change compiler, compiler flags, and BLAS lib

```
CC       = mpiicc

CCNOOPT  = $(HPL_DEFS)

OMP_DEFS = -openmp

CCFLAGS  = $(HPL_DEFS) -O3 -w -ansi-alias -i-static -z
noexecstack -z relro -z now -nocompchk -Wall

...

...

LAdir        = $(MKLROOT)
ifndef  LAinc
LAinc        = $(LAdir)/mkl/include
endif
ifndef  LAlib
LAlib        = -L$(LAdir)/mkl/lib/intel64 \
               -Wl,--start-group \
               $(LAdir)/lib/intel64/libmkl_intel_lp64.a \
               $(LAdir)/lib/intel64/libmkl_intel_thread.a \
               $(LAdir)/lib/intel64/libmkl_core.a \
               -Wl,--end-group -lpthread -ldl
endif
```

- Learn how to build from scratch even if you use a vendor provided binary for the final run.

HPL Input (HPL.dat)

Important parameters:

- N #Matrix dimension
- NB #Block size for LA operations
- P #Factorization rows
- Q #Factorization columns. PxQ must equal your MPI processes
- Change # of algorithms to test

Run HPL

export OMP_NUM_THREADS=12

mpirun –n 2 ./xhpl

or follow instructions of vendor provided binaries

HPL output

```
================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WC00C2R2      115584   192     1     2            1382.83              7.44458e+02
```

Tune HPL (BLAS)

- HPL is mostly BLAS call.
- Experiment different BLAS implementations
  - MKL (Intel)
  - OpenBLAS (open source)
  - BLIS (open source)
  - Libsci (Cray)
- Vendor provided binaries
  - Intel, Nvidia, AMD

Tune HPL (problem size)

- Bigger problem size
  - A.I ~ O(N) → higher A.I. → more compute bound
  - maximize computation/communication ratio.
  - Hides other miscellaneous overheads.
- In practice, use ~90% of your memory
  - (leave a few GB to OS)

Tune HPL (others)

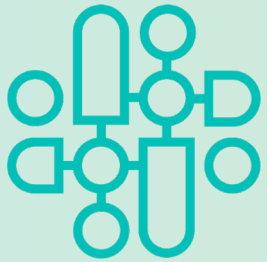
- NB (block size)
  - Try different values
  - Or follow suggested value for vendor provided binaries
- Use MPI for distributed memory parallelism
- Use OpenMP for shared memory parallelism
- Experiment different layouts of MPI processes/OpenMP threads.
  - Typically, 1 MPI process for each NUMA node or chiplet.

Resources

- https://www.netlib.org/benchmark/hpl/faqs.html
- https://frobnitzem.github.io/hpl-hpcg
- https://www.advancedclustering.com/act_kb/tune-hpl-dat-file
- https://ulhpc-tutorials.readthedocs.io/en/latest/parallel/mpi/HPL
- https://www.intel.com/content/www/us/en/docs/onemkl/developer-guide-linux/2023-1/run-the-intel-distribution-for-linpack-benchmark.html
- https://developer.amd.com/spack/hpl-benchmark
- https://catalog.ngc.nvidia.com/orgs/nvidia/containers/hpc-benchmarks

HPL Exercise

1. Build and install the HPL benchmark using your choice of BLAS and MPI. (30 points)
   - Describe your cluster
   - Briefly explain your choices of the libraries.
   - Include the scripts that you use to build HPL.
2. Run the HPL benchmark on a single node with your optimal setup. (40 points)
   - Either use the binary you built or the one from vendor.
   - Provide the final output
   - Briefly explain what tuning you have done.
3. Compute the theoretical peak FLOP/s for the node you use (30 points)
   - Which CPU frequency to use? Run an HPL on all cores and see the actual frequencies in /proc/cpuinfo, it could be much lower than the nominal frequency.



Questions?