Overview of the SCC25 Benchmarks

Ryan DeRue, Purdue University
Miro Hodak, AMD

SC25
St. Louis, MO | hpc ignites.

Benchmark Tasks

- Two Synthetic Benchmarks
    1. High Performance Linpack (HPL)
    2. HPL-Mixed Precision (HPL-MxP)
- One AI Benchmark
    1. MLPerf Inference

Benchmarking Notes

- Benchmarking portion will take place from Monday morning to Monday evening
  - More detailed logistics are available in the SCC overview webinar that was given earlier
- Detailed submission instructions will be shared later
  - Follow submission instructions carefully
- We plan to use scripts for automated grading
  - Name your files and folders as instructed
- Certifying your cluster requires
  - Staying within power budget (10kW) while running all 3 benchmarks
  - All benchmarks must be run with the same hardware configuration
  - After certification
    - You cannot change your hardware configuration or reboot any node
    - You can re-submit a single benchmark within the benchmarking window
      - Normal penalty for going over power budget will apply
- Suggestions
  - Familiarize yourself with the benchmarks ahead of time
  - Write scripts and automate where possible
  - Vendor provided binaries are acceptable if these are publicly available

Changes from Last Year (SCC24)

1. No mystery benchmark this year
2. No benchmarking interview this year
3. Power budget is increased from 6kW to 10kW

HPL

- One of the most popular benchmarks in the HPC world
  - TOP-500 listing of world's fastest supercomputers use HPL
    - Most recent list published June 2025
      - El Capitan, LLNL, 1.742 EFlops/s
- Solves a (random) dense linear system in double precision (FP64)
  - https://www.netlib.org/benchmark/hpl/index.html
- Used to measure "performance" of a computer or a cluster
- Input
  - Parameters for the benchmark run
- Output
  - No. of FLOP/s
- SCC24 best HPL score: Nanyang Technical University (236.4 Teraflops)
  - This was performed within a 6kW power budget

HPL

- Internally uses BLAS libraries for LA subroutines
  - Intel MKL
  - OpenBLAS
- Uses MPI for distributed memory parallelism
- Uses OpenMP for shared memory parallelism
- Can take variable amount of time based on problem size
  - Size of the matrix
- Problem size is typically determined by the memory (RAM) size

Building HPL

- Download and untar the HPL source tarball
  - `tar -xf hpl-2.3.tar.gz`
- Need to load the necessary 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`
- NVIDIA and AMD both provide containers for launching HPL on their GPUs
  - NVIDIA Documentation: https://docs.nvidia.com/nvidia-hpc-benchmarks/HPL_benchmark.html
  - AMD Documentation: https://github.com/amd/InfinityHub-CI/tree/main/rochpl

HPL Input File (HPL.dat)

- Copy HPL.dat to your working directory
- Edit the input file to reflect your setup
- Important parameters
  - `N      #Array size`
  - `NB     #Block size for LA operations`
  - `P      #Factorization rows`
  - `Q      #Factorization columns. PxQ must equal your MPI processes`
  - Change no. of algorithms to test
- Now run HPL
  - `mpirun -np 2 xhpl`

HPL Tuning Guide

- Typically, HPL on CPUs should occupy 80-90% of your memory for optimal performance
  - On GPUs, this is often closer to 90-100%
- `NB` (Block size) impacts performance
  - Try different `NB` sizes
  - Empirically find out which one is better
  - Optimal NB size depends on your architecture/GPU type
- Exact layout of MPI processes/OpenMP threads impact performance
  - Optimal layout depends on the processor architecture

HPL 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://developer.amd.com/spack/hpl-benchmark/

HPL-MxP

- Developed as a result of the shift towards reduced precision arithmetic
- HPL with FP16 factorization
  - Solver has a “refinement” step to bring the solution back to 64-bit accuracy
- A Top500 of HPL-MxP is also published bi-anually
  - El Capitan, LLNL, 16.680 EFlops/s

June 2025

Rank	Site	Computer	Cores	HPL-MxP (Eflop/s)	TOP500 Rank	HPL Rmax (Eflop/s)	Speedup
1	DOE/SC/LLNL	El Capitan	11,039,616	16.680	1	1.7420	9.6
2	DOE/SC/ANL	Aurora	8,159,232	11.643	3	1.0120	11.5
3	DOE/SC/ORNL	Frontier	8,560,640	11.390	2	1.3530	8.4
4	AIST	ABCI 3.0	479,232	2.363	15	0.1451	16.3
5	EuroHPC/CSC	LUMI	2,752,704	2.350	9	0.3797	6.2

HPL-MxP Input

- HPL-MxP input looks very similar to HPL but is typically passed through the command line
  - `N -> --n  <int>`
  - `NB -> --nb <int>`
  - `P -> --nprow <int>`
  - `Q -> --npcol <int>`
- A few new required arguments:
  - `--gpu-affinity <string> //colon delimited list of GPUs`
  - `--nporder <string> //”row” or “column”`
- There are also a few more optional arguments depending on the implementation used
  - Pay particular attention to options regarding GPU-aware MPI

HPL-MxP Implementations

- The authors of HPL-MxP created a reference implementation
  - You would need to build this implementation from source
- Most teams will want to use a vendor optimized implementation
  - NVIDIA's HPL-MxP Container
  - AMD's HPL-MxP Container
  - Intel's HPL-AI* Container
- Each of these implementations have slightly different optional arguments
  - Read the documentation for the one you are using
- You are allowed to use any *publicly available* vendor implementations

HPL-MxP Resources

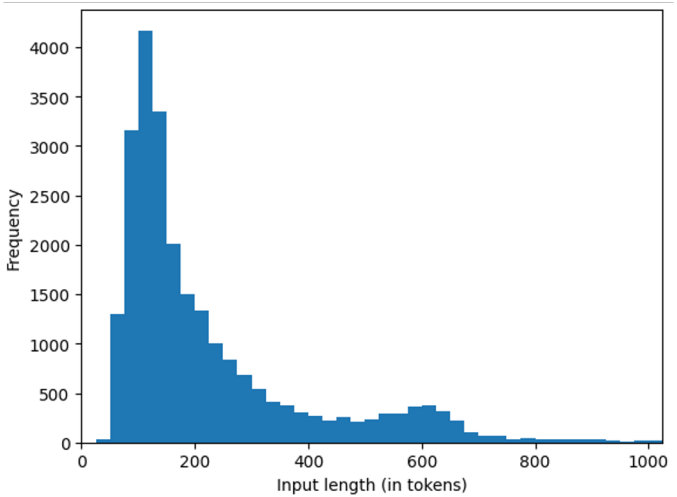
- https://hpl-mxp.org/
- https://bitbucket.org/icl/hpl-ai/src/main/
- https://catalog.ngc.nvidia.com/orgs/nvidia/containers/hpc-benchmarks
- https://www.amd.com/en/developer/resources/infinity-hub/hpl-mxp.html
- https://docs.nvidia.com/nvidia-hpc-benchmarks/HPL_MxP_benchmark.html
- https://www.intel.com/content/www/us/en/docs/onemkl/developer-guide-linux/2024-1/overview-intel-distribution-for-hpl-ai-benchmark.html

MLPerf Benchmark for SCC'25

Llama2-70b Inference

Llama2-70b Inference Benchmark

- Most popular MLPerf Inference benchmark
- Why Llama2-70b
  - Well known
  - Popular
  - Licensed for free use
  - Quality



- Q&A task using OpenOrca dataset
  - 24,576 samples, ISL distribution on the right

More information: https://mlcommons.org/2024/03/mlperf-llama2-70b/

MLPerf Inference Test Modes

Offline

- All data received in the beginning, measures how long it takes to process the dataset
- Throughput test
- Metric: Tokens/sec

Server

- Data sent according to a set frequency, have to be returned with a given latency
- Latency-bound throughput
- Not part of SCC'25

Accuracy

- Accuracy test required to have a valid result
- Better than 99.9% of
  - ROUGE-1 = 44.4312
  - ROUGE-2 = 22.0352
  - ROUGE-L = 28.6162

MLPerf Inference Implementations

MLPerf Inference provides reference code: Code repo, Llama2-70b code

- Basic, example code that should work on most devices
- Not performant

Optimized implementations

- MLPerf allows submitters to rewrite the reference code as long as it satisfies model equivalency rules
  - Model quantization and different inference frameworks are allowed
- Optimized models are much faster than reference code
- Existing submitted implementations: Nvidia (Hopper, Blackwell), AMD (Instinct MI300X, MI325X and MI355X).

Implementations to use for SCC’25

- Use Nvidia or AMD implementations strongly suggested
- Optimized implementations found in MLPerf Inference 5.0 submitted results
  - Nvidia: https://github.com/mlcommons/inference_results_v5.0/tree/main/closed/NVIDIA/code/llama2-70b-99.9/tensorrt
  - AMD: https://github.com/mlcommons/inference_results_v5.0/tree/main/closed/AMD/measurements/8xMI325X_2xEPYC_9575F/llama2-70b-99.9/Offline
  - https://rocm.blogs.amd.com/artificial-intelligence/reproducing-amd-mlperf-inference-submission/README.html
- On 9/9, new version of MLPerf Inference, 5.1, will be released. Updated LLama2-70b code will be released. You should use the new version

How to run for SCC’25

Two options

- Use MLCFLow automation
  - Supports Nvidia implementation, AMD to be added
  - More SCC’25 instructions will be provided later
  - https://docs.mlcommons.org/inference/benchmarks/language/llama2-70b/
- Use MLPerf optimized code directly without automation
  - It is best to wait for MLPerf 5.1 release - happens on 9/9
  - Expected link for AMD: https://github.com/mlcommons/inference_results_v5.1/tree/main/closed/AMD/code/llama2-70b-99.9
  - Expected link for Nvidia:https://github.com/mlcommons/inference_results_v5.1/tree/main/closed/NVIDIA/code/llama2-70b-99.9/tensorrt
  - Updates will be provided later

MLCFlow Automation

MLCFlow automates multiple steps of the MLPerf inference implementations like Nvidia into a single command.

It eases the usability by easily supporting more generic GPU configurations

It provides a centralized docs page - https://docs.mlcommons.org/inference which works across multiple MLPerf inference implementations and devices

It makes it easier to try different configurations like batch size

Prerequisites for Running

- Model
  - Downloading the base model and quantizing takes a long time, alternative methods
  - Nvidia quantization: will be hosted on MLCommons resources, link will be provided later
  - AMD quantization: https://huggingface.co/amd/Llama-2-70b-chat-hf_FP8_MLPerf_V2

- Dataset
  - Easy to download, already hosted on MLCommons resources

What is Required for This Benchmark

- Run Offline and Accuracy runs
- Get valid runs - performance run is valid and accuracy criterion is satisfied
- Package and submit your results - scripts will be shared later

Tips to be successful

Do test runs before competition

- Experience is helpful, MLPerf is difficult for beginners

Understand the prerequisites, requirements and process

Get model and dataset ahead of time - bring it to SCC’25

Ask questions as you prepare

Summary

Llama2-70b is the most popular MLPerf Inference benchmark

Use optimized code for this benchmark rather than reference

Offline performance run with Accuracy is required

Submitting the results is also required

Additional information will be provided later

Test and ask questions early

Questions

Acknowledgements

- MLCommons for supporting the SCC
- Arjun Suresh (MLCommons)
- Anandhu Sooraj (MLCommons)
- Amiya Maji (Purdue)