Runtime Analysis of Pennylane-Lightning Benchmarks

Md Mazder Rahman

January 2023

Abstract

This report presents an analysis of runtimes for pennylane-lightning executable 'bench_kernels'. Four execuatables are built using both GCC 11 and Clang 15 compilers through Spack-specific commands in both bare-metal hardware and docker container. The runtime statistics are obtained from running execuatables in both bare-metal and container environments. Experiments show that in both environments bare-metal hardware and docker container, the runtime of the executable built with Clang is lower than the runtime of that executable built with GCC. Therefore, clang will be the best compiler for Pennylane-Lightning in comparing with GCC.

1 Problem Definition

PennyLane Lightning is XANADU_AI C++20 backed device simulator, and supports all modern CPU architectures. It is often necessary to validate and verify performance across various different compilers, and within different runtime environments. The goal of this work is to identify the best compiler for PennyLane Lightning, comparing GCC/G++ 11 against Clang 14, within a container and running on bare-metal hardware.

2 Experimental Setup

A linux system with Ubuntu 2022 is used for experiments. The Table 1 shows the basic information about the CPU architecture and general runtime environmet. The subsequent taks are carried out for this experiments. All scripts, results and correspoding documents can be found in the repository

https://github.com/mazder/pennylane-lightning-benchmarks

- **Packaging:** Spack packages for PennyLane Lightning updated for v0.27.0, and targeting the benchmarking suite 'Bench_Kernels.cpp' file as a variant are created separately for gcc and clang compiler settings. Files can be

Table 1: System and Runtime Environment.

System
Linux mach2 5.15.0-46-generic #49-Ubuntu SMP Thu Aug 4 18:03:25 UTC 2022 x86_64 x86_64 x86_64 GNULinux
Runtime Environment
Run on (48 X 3000 MHz CPU s) CPU Caches: L1 Data 32 KiB (x24) L1 Instruction 32 KiB (x24) L2 Unified 1024 KiB (x24) L3 Unified 16896 KiB (x2) CPU::AVX: True CPU::AVX2: True CPU::AVX512F: True CPU::Brand: Intel(R) Xeon(R) Silver 4116 CPU @ 2.10GHz CPU::Vendor: GenuineIntel

found in the directory pennylane-lightning-benchmarks/spack_packages/ in the repo.

- **Building:**
  - Using system compiler GCC-11.3.0, the PennyLane Lightning benchmarking suite is built through Spack-specific commands. This process has been automated using the Makefile given in the repo.
  - Through Spack-specific commands, LLVM Clang/Clang++ 15.06 is installed followed by the PennyLane Lightning benchmarking suite is built. This process also has been automated using the Makefile given in the repo.
  - Through Spack-specific commands, GCC/GCC++ 11.3.0 and LLVM Clang/Clang++ 15.06 are installed followed by the PennyLane Lightning benchmarking suite is built in separate docker containers results in two docker images gcc-lightning and clang-lightning. Corresponding docker files can be found in the directories gcc_docker and llvm_gcc_docker in this repository. Building Pennylane-Lightning 'bench_kernels' through Spack-specific commands has also been automated using Makefile given in the repo.

  The building and installation commands are given in README.md file in this repository.

- **Running:**
  - **In Bare-Metal:** For all benchmarks suite, the runtime statistics for two different binaries (built with gcc and clang in bare-metal) for 'Bench_Kernels.cpp' are collected in files 'gcc_run_bare_metal.txt' and 'clang_run_bare_metal.txt' for post processing.
  - **In Container:** From the two different containers, for all benchmarks suite, the runtime statistics for two different binaries (built with GCC and Clang in containers) for 'Bench_Kernels.cpp' are collected in files 'gcc_run_docker_container.txt' and 'clang_run_docker_container.txt' for post processing.

  There are two sets runtime statistics are collected in text files. Files can be found in the data_post_processing directory in this repository.
  The run commands and Sample outputs of results from 4 different runs are given in README.md file in this repository.

3 Analysis of Experimental Results

In this experiment, all benchmarks run with 4 different binaries and results are collected for post-processing and analysis. The total number of operations for all benchmarks is 3021. With the 2 sets of data, a comparative analysis of performance of gcc and clang compiled binaries as well as impact of runtime in container are discussed in the subsequent sections.

3.1 Experiment #1

In this experiment, the runtime environments for 4 runs are different in CPU load average.

3.1.1 GCC VS CLANG

In this section, the runtimes for gcc and clang compiled binaries are compared. The Table 2 shows the total runtimes and iterations for gcc and clang compiled binaries in Bare-metal. The CPU load averages in both cases is greater than 1, therefore, both cases runtime incurs an extra overhead. According to results it can be noticed that when running clang compiled binary, the CPU load average is higher than that of the other. Considering that running clang compiled binary incurs more overhead than that of the other. However, 39.33% speed up in total runtimes for clang compiler against gcc compiler. Therefore, the performance of clang compiler is better than that of gcc compiler.

The Table 3 shows the total runtimes and iterations for gcc and clang compiled binaries in docker container. When running clang compiled binary, the CPU load average is lower than the other. It shows that 31.89% speed up in total runtimes for clang compiler against gcc compiler. It can be noticed that

Table 2: Experimental Results GCC VS CLANG in Bare-Metal.

On Total Operations: 3021					
Compilers	Runtime Env	Load Average	Real Time(ns)	CPU Time(ns)	Iterations
GCC	BareMetal	1.64, 1.64, 1.47	1362412287527	1362284099865	55256390
CLANG	Bare-Metal	2.37, 1.92, 1.59	826563149671	826495396679	62740219
Speed up			39.3309%	39.3302%	-13.5438%
On Per Operation					
GCC			4.50981e+08	4.50938e+08	18290.8
CLANG			2.73606e+08	2.73583e+08	20768

both runs incurred overheads with close differece in CPU load average. However, with the less overhead incurring, the performance of clang compiler is better than that of gcc compiler.

Table 3: Experimental Results GCC VS CLANG in Container.

On Total Operations: 3021					
Compilers	Runtime Env	Load Average	Real Time(ns)	CPU Time(ns)	Iterations
GCC	Container	1.00, 0.63, 0.57	1339273758987	1339029609678	49916923
CLANG	Container	0.89, 0.80, 0.63	912143378870	911899816179	50765470
Speed up			31.8927%	31.8985%	-1.69992%
On Per Operation					
GCC			4.43321e+08	4.43241e+08	16523.3
CLANG			3.01934e+08	3.01854e+08	16804.2

In summary, this experiment shows that for both runs in bare-metal and container, clang compiler outperforms against gcc, however, it is still necessary to runs with ideal runtime environments as discussed in Experiment#2.

3.1.2 Bare-Metal VS Container

In this section, the runtimes in container for gcc and clang compiled binaries are compared against the runtimes in bare-metal for gcc and clang compiled binaries to verify the impacts of performance in container.

The Table 4 compares the runtime of gcc in bare-metal and the runtime of gcc in container. It is observed that the CPU load average in container less than that of bare-metal. Obviously in container the runtime of applications incurs less overhead, however, it is not much than that in bare-metal. It can be noticed that runtime in container gets speed up around 1.7%. It is also noticed that runtime of per operation in both bare-metal and container are very closed. If we consider that the achieved speed up is due to the fact of less CPU

Table 4: Experimental Results GCC in Bare-Metal VS GCC in Container.

Compilers	Runtime Env	Load Average	Real Time(ns)	CPU Time(ns)	Iterations
GCC	BareMetal	1.64, 1.64, 1.47	1362412287527	1362284099865	55256390
GCC	Container	1.00, 0.63, 0.57	1339273758987	1339029609678	49916923
Speed up			1.69835%	1.70702%	9.66308%
On Per Operation					
GCC			4.50981e+08	4.50938e+08	18290.8
GCC			4.43321e+08	4.43241e+08	16523.3

load average, then there is no impacts of runtime in container when using gcc compiled biranies.

Table 5: Experimental Results GCC VS CLANG in Bare-Metal.

Compilers	Runtime Env	Load Average	Real Time(ns)	CPU Time(ns)	Iterations
Clang	BareMetal	2.37, 1.92, 1.59	826563149671	826495396679	62740219
CLANG	Container	0.89, 0.80, 0.63	912143378870	911899816179	50765470
Speed up			-10.3537%	-10.3333%	19.0862%
On Per Operation					
Clang			2.73606e+08	2.73583e+08	20768
CLANG			3.01934e+08	3.01854e+08	16804.2

The Table 5 compares the runtime of clang in bare-metal and the runtime of gcc in container. It can be noticed -10.33% speed up in runtimes, i.e, the runtime of clang compiled binary in container is higher than that of clang compiled binary in bare-metal even though runtime environment in container has significant less overhead in comparing with bare-metal. So in this case, a performance degradation is noticed when using clang compiled binary in container.

In summary of experimet#1, it is found that the performance of runtime in pennylane-lighting 'Bench_Kernels.cpp' compiled with clang compiler is better than that of compiled with gcc. In both bare-metal and docker container, similar results for gcc vs clang are observed. However, it shows that there is a negative impact on runtime of clang compiled binary in contaier in compred to bare-metal.

3.2 Experiment #2

In this section, another set of results and analysis are presented. The Table 6 shows the results. It is observed that when running simulation, all 4 runtime environment are very close with the CPU load averages. However, the

same trend of results in comparsion of runtimes can be noticed as obtained in experiment #1. That is, the performance of runtime in pennylane-lighting 'Bench_Kernels.cpp' compiled with clang compiler is better than that of compiled with gcc, however, still showing impacts on running in container.

4 Conclusion and Future Work

This experiment is done in a server which is not ideally bare-metal since other jobs are running. Container also runs on the same server. To get ideal CPU work loads as much as possible all 4 simulations run serially. According to this experiment, Clang compiler is a better choice for Pennylane-Lightning in comparing with gcc even though there is a little performance degradation noticed when running in container as shown in both the first and the second experiments. It would be better to run further this exeperiment on ideally bare-metal hardware and verify whether CPU load average is the reason of degradation in this experiment or not. Moreover, whereas running container on bare-metal results in the best performance of CPU runtime but it could be beneficial to verify this results running with cloud virtual environments as considered the end users will run Pennylane-lightning on the cloud.

Table 6: Experimental Results GCC VS CLANG and Bare-Metal vs Container.

GCC vs CLANG in Bare-Metal					
Compilers	Runtime Env	Load Average	Real Time(ns)	CPU Time(ns)	Iterations
GCC	BareMetal	0.69, 0.60, 0.70	1336008819551	1335759811010	55934441
CLANG	Bare-Metal	0.52, 0.65, 1.03	830112010356	829955389480	62061339
Speed up			37.8663%	37.8664%	-10.9537%
			On Per Operation		
GCC			4.42241e+08	4.42158e+08	18515.2
CLANG			2.74781e+08	2.74729e+08	20543.3

Table 6(a)

GCC vs CLANG in Container					
Compilers	Runtime Env	Load Average	Real Time(ns)	CPU Time(ns)	Iterations
GCC	Container	0.62, 0.58, 0.72	1327611718045	1327415567695	50684446
CLANG	Container	0.70, 0.56, 0.55	896466824302	896279860413	51617049
Speed up			32.4752%	32.4793%	-1.84002%
On Per Operation					
GCC			4.39461e+08	4.39396e+08	16777.4
CLANG			2.96745e+08	2.96683e+08	17086.1

Table 6(b)

GCC in Bare-Metal vs GCC in Container					
GCC	BareMetal	0.69, 0.60, 0.70	1336008819551	1335759811010	55934441
GCC	Container	0.62, 0.58, 0.72	1327611718045	1327415567695	50684446
Speed up			0.628521%	0.624681%	9.38598%
On Per Operation					
GCC			4.42241e+08	4.42158e+08	18515.2
GCC			4.39461e+08	4.39396e+08	16777.4

Table 6(c)

CLANG in Bare-Metal vs CLANG in Container					
Compilers	Runtime Env	Load Average	Real Time(ns)	CPU Time(ns)	Iterations
CLANG	Bare-Metal	0.52, 0.65, 1.03	830112010356	829955389480	62061339
CLANG	Container	0.70, 0.56, 0.55	896466824302	896279860413	51617049
Speed up			-7.99348%	-7.99133%	16.829%
On Per Operation					
Clang			2.74781e+08	2.74729e+08	20543.3
CLANG			2.96745e+08	2.96683e+08	17086.1

Table 6(d)