Hardware Comparison and Benchmarking Studies of FFTs with Gearshifft

Paula Fernanda Olaya Garcia
University of Tennessee, Knoxville
Department of Electrical Engineering and Computer Science
polaya@vols.utk.edu

***Abstract*— The fast Fourier transform (FFT) [1] is an algorithm widely used today in science and engineering [2]. It is one of the tests from The HPC Challenge (HPCC) Benchmark which allows to measure the performance from multiple perspectives: processor, memory subsystem, and system interconnect [3]. Therefore, we used Gearshifft, which is a new FFT benchmarking harness and test suite, to assess and compare performance of Hōkūle'a and Cori in a consistent and verifiable fashion.**

I. INTRODUCTION

The fast Fourier transform (FFT) is an algorithm widely used in application fields, such as Audio/Image/Video, Digital Signal Processing [4], due to the inherent data level parallelism exposed by the FFT, there are new multiple FFT algorithms exploiting it at various scales, dimensions, and optimization levels [5]. Gearshifft is a recently developed open source and vendor agnostic benchmark suite to process a wide variety of problem sizes and types of FFT implementations (fftw [6], clFFT, cuFFT) [7]. These variety of supported libraries allow for testing optimized versions of FFT algorithms on most if not all HPC platforms, performing forward and backward FFT transforms on various types of input data (both in shape, memory organization, precision and data type). Apart from running on Hōkūle'a and Cori, we used the results from the gearshifft results repository available in github, which let us compare performance between GPUs and interconnect, CPUs and also across GPUs and CPUs.

II. BACKGROUNG

A. Software

1) Gearshifft: Gearshifft is a tool which provides quick comparisons of systems, supply the results in a standard format for easy comparisons, standardize the data collection process by Github data results and determine which FFT variant is best for a given problem size on a given architecture. The parameters available to alter in Gearshifft are:

- Memory mode:
  - in-place: The input data stores the output data, therefore there will be low memory footprint and low bandwidth.
  - out-of-place: The input transform is located in a different place from where the original input resides on memory.
- Transform direction:
  - Forward: From time discrete to frequency.
  - Backward: From frequency to time discrete.
- Transform Dimensionality: 1D, 2D, ... , N-D.
- Transform Type: Real to Complex and Complex to Complex.
- Transform Precision: precision, i.e. 32-bit or 64-bit IEEE floating point number representation

2) FFT Libraries: FFT is computed as a succession of butterfly planes, where for each plane a butterfly operation is repeatedly performed on an independent set of data. Hence butterflies can be executed independently from each other, obtaining in principle a truly high speed-up when implemented on a parallel architecture [4]. Due to this high level of parallelism, there are multiple libraries and optimized algorithms. Specifically, Gearshifft supports libraries for CPUs like FFTW or FFTW+MKL and for GPUs (cuFFT).

- FFTW: Subroutine library in C for computing the discrete Fourier transform (DFT) in one or more dimensions, of arbitrary input size, and of both real and complex data. FFTW is free and often provides a portable interface with optimized versions available including distributed implementations using MPI, OpenMP or Pthreads.
- FFTW+MKL: Collection of FFTW3 wrappers to Intel MKL. The wrappers translate calls of FFTW3 functions to the calls of the Intel MKL Fourier transform (FFT).
- cuFFT: Simple interface for computing FTTs on a NVIDIA GPU using CUDA. It allows users to leverage the floating-point power and parallelism of the GPU in a highly optimized fashion.

In order to execute any of these libraries, first, the user needs to define the FFT problem in terms of dimension, input signal, precision, type of transformation and memory mode, then Gearshifft creates a planner which for FFTW finds the best suited radix factorization based on the shape of the input signal while cuFFT uses internal building blocks to optimize the transform for the given configuration and the particular GPU hardware selected. Once it is created, the plan is executed in different stages shown in figure 1. For purpose of this project, we are going only to focus on:

- Time Upload: Refers to the memcpy operation. Working in GPUs is expected to be higher because of the transfer

from main memory in CPU to GPU memory.

- Time Execute Forward FFT
- Total Time: All stages from allocation to cleaning resources.

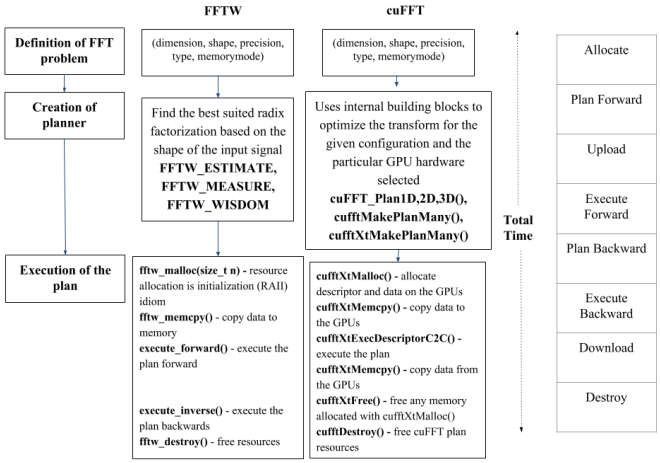

Fig. 1: FFT Library Execution: Definition FFT Problem, creation planner and execution. Functions deployed for every library (FFTW and cuFFT) and what stages are measure in time: allocation in memory, creation and execution of plan for specific direction transform, tranfer of data and clean memory.

B. Hardware

For the purpose of this project that is to compare the efficiency of GPUs and CPUs for benchmark codes, in this specific case Gearshifft. We used two machines, their description is explicit in Table I.

1) Hōkūle'a Architecture: Hōkūle'a is an IBM Power Systems S822LC, with IBM Power8+10C operating at 2.86 GHz, with an Infiniband EDR interconnect, and NVIDIA Tesla GP100 accelerators connected to the Power8+ nodes. Hōkūle'a has 32 compute nodes, each of those nodes has 4 NVIDIA Tesla Pascal P100 SXM2 GPUs, for a total of 128 GPUs. Each GPU has a theoretical peak of 5.3 TF for double precision, and 10.6 TF for single precision. Each GPU has 3584 cores, 16 GB of HBM2 stacked memory with 720 GB/s bandwidth. Each GPU has NVLink 1.0 with a total of 160 GB/s interconnect bandwidth per GPU. NVLink is used to connect a GPU to a POWER8 processor as well as to another GPU. The bandwidth between a GPU and a POWER8 processor is 40 GB/s in each direction (80 GB/s total), and the bandwidth between a GPU and another GPU is also 40 GB/s in each direction (80 GB/s total). Two GPUs are connected to a single POWER8. In the cluster, there are 2 POWER8 processors per compute node giving a GPUs per node. The original version of the Gearshifft benchmark runs only on single node.

2) Cori: Cori is a Cray XC40, ranked as the 8th most powerful supercomputer in the world on the November 2017 list of Top 500 supercomputers in the world. Cori is a unique among supercomputers of its size with two different kinds of nodes, 2388 Intel Xeon "Haswell" processor nodes operating at 2.3 GHz and 9688 Intel Xeon Phi "Knight's Landing" nodes at 1.4 GHz. Cori also features a 1.8 PB Cray Data Warp Burst Buffer with I/O operating at a world's-best 1.7 TB/sec. Each node from "Haswell" consists of 32 cores and 2 threads per core for a total of 64 thread per node, while each "KNL" node has 68 cores, each with 4 hardware threads for a total of 272 threads per node.

Specification	Haswell	KNL	Hōkūle'a
Cluster Size	2388 nodes	9688 nodes	32 nodes POWER8 each with 4 GPUs
Number of Threads	2 threads per core = 64 threads per node	4 hardware threads per core = 272 threads per node	32 threads per warp 64 max warps per multiprocessor 2048 max threads per multiprocessor
Double-Precision Performance	1.2 TF	3 TF	5.3 TF
Single-Precision Performance	2.4 TF	6 TF	10.6 TF
Memory	Each node has 128 GB DDR4 2133 MHz memory (four 16 GB DIMMs per socket); 298.5 TB total aggregate memory	96 GB DDR4 2400 MHz memory, six 16 GB DIMMs (102 GiB/s peak bandwidth)	16 GB of HBM2 stacked memory with 720 GB/s bandwidth

TABLE I: Description of Hōkūle'a and Cori architectures.

III. Methodology

Based in figure 2 there are multiple tests to do and analyze, but for this project we will focus in 1D, memory inplace, power of 2 values and real to complex transforms.

- Comparison of Interconnect: We run the same FFT problem in P100 GPU one from Hōkūle'a with NVLink interconnect and another from their Github repository with PCIe interconnect.
- Comparison between GPUs: Run in K80, P100, V100 with same interconnect to compare memory capacity in older and newer GPUs.
- Comparison between CPUs: Run in both CPUs from Cori to analyze in what point having more threads and cores, like KNL, affects the run time.
- Comparison between GPUs and CPUs: Run in Haswell as well as in P100 from Hōkūle'a to understand the benefits for accelerators and multicores.

The plots are obtain from two scripts. The first one in an application from Gearshifft, available in Github $mpicbg - scicomp/gearshifft - results$ which allows you to compare just 2 machines and the second is our own script for multiple systems.

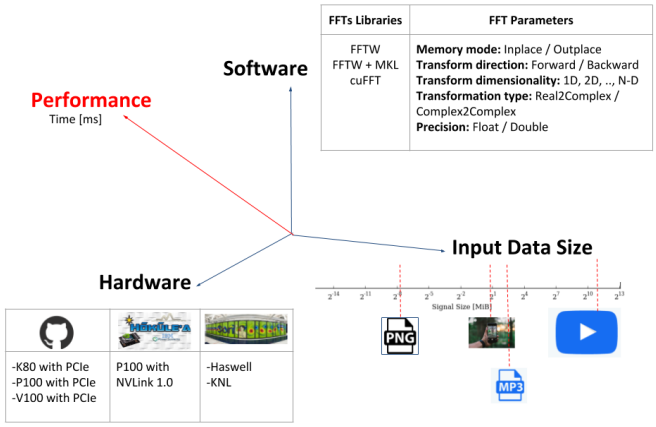

Fig. 2: There are multiple variables in our testing space: Parameters of FFT problem and the machine where is running the benchmark code. The independent axis (x-axis) is Input Data Size which goes from 2^{-15} MiB to 2^{15} MiB and we will measure time or speed up of specific stages.

IV. RESULTS

The GitHub site for the gearshifft benchmark contains data from various researchers running the benchmark on different systems. In this note, we perform specific comparisons with the GitHub data sets to obtain a better understanding the performance of FFTs on relevant GPU (P100) and CPU (Haswell and KNL) architectures.

A. Comparison between Interconnect

For the first test we run a 1D FFT with Float (32B) precision with the real inplace data being a power of 2 in our P100 GPU with NVLink interconnect and compared with P100 GPU using a slower PCIe interconnect. In figure 3 we see the difference in interconnect when we compare the time spent in the upload portion of the benchmark. We see that the Hōkūle'a P100 with NVLink gives consistently faster uploads for larger signal sizes. The difference in upload speed for the two P100 devices is seen more clearly when we plot the speedup of upload time using the NVLink interconnect compared to the PCIe interconnect as a function of signal size, like in figure 4. We see that the speedup is 1.6X or greater for signal sizes greater than 2^{-3} MiB.

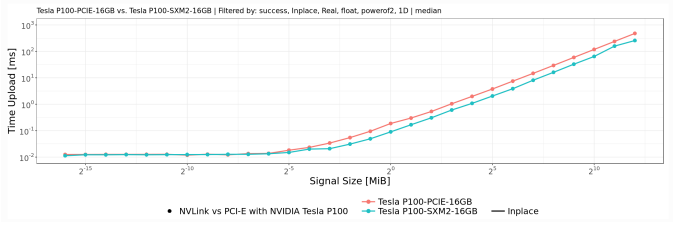

Fig. 3: The benefit of the NVLink interconnect on the Hōkūle'a NVIDA Tesla Pascal P100 GPU is shown. Faster upload speeds, obtained with the NVLink, are compared to results using a P100 GPU but with the PCIe interconnect.

As seen in both figure 3 and 4, the cross point is approximately 32 KB (2^{-3} MiB), the reason is that the PCIe do 2 B/T while NVLink 4 B/T, so it takes almost the double of time to transfer the data from the CPU to the GPU. Another observation taken from [7] is that the runtime curves of GPU FFT implementations follows an inverse roofline curve, for input signals smaller than the roofline turning point at 2^{-3} MiB the FFT implementation appears to be of constant cost, i. e. to be compute bound. Above the aforementioned threshold, the implementation appears to be memory bound and hence exposes a linear growth with growing input signals which corresponds to the $O(nlog(n))$ complexity, in other words, there is more access to memory to read data than operations done to the data.

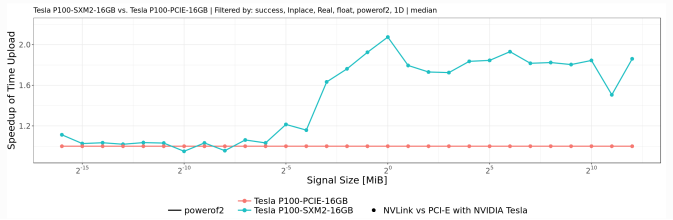

Fig. 4: Speed up of Time Upload as function of signal size, having as reference the P100 with PCIE interconnect. It is evident the benefit of the NVLink interconnect compared to the PCIe interconnect on the Hōkūle'a NVIDA Tesla Pascal P100 GPU.

B. Comparison between GPUs

Since the idea of this test is to compare the GPUs we compared Time Upload (figure 5) and Time FFT (figure 6) from 3 GPUs with same PCIe interconnect. Because the K80 just accepted and older version of cuFFT and V100 the newest one, we run both versions in P100 GPU. From figure 5 we noticed that even with the same interconnect, the upload time is changed. Based on the architectures the Bus width [bit] and Memory size [GB] for P100 and V100 are the same, 4096 bits and 16 GB. While K80 bus is 768 bits and 12GB. That's why the time upload for P100 is approximately 1.4X faster than P100 and V100 and P100 have similar behaviours. Figure 6 shows that P100 is approximately 2.6X faster than the K80 and the V100 is 1.5X faster than the P100 (V100 is 3.9X faster uploading than K80). The reason is given by the relation of compute instructions to memory access operations and memory bandwidth, where P100 (720 GS/s) stacked memory features 3x the memory bandwidth of the K80 (240 GB/s), an important factor for memory-intensive applications. And V100 (900 GB/s) is 1.25X faster.

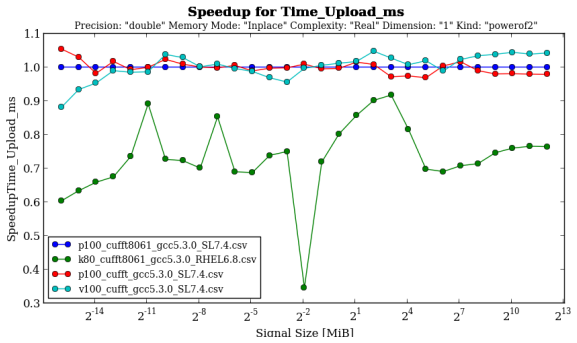

Fig. 5: Speedup of Time Upload for P100 with old (blue) and new (red) versions of cuFFT, K80 (grean) and V100 (cyan). The P100 GPU is significantly faster than the previous generation Kepler K80 GPU for FFT calculations as shown with speedup a function of signal size. The next generation NVIDA Tesla Volta V100 GPU is faster on FFT calculations as compared to the NVIDA Tesla Pascal P100 GPU as shown with speedup a function of signal size.

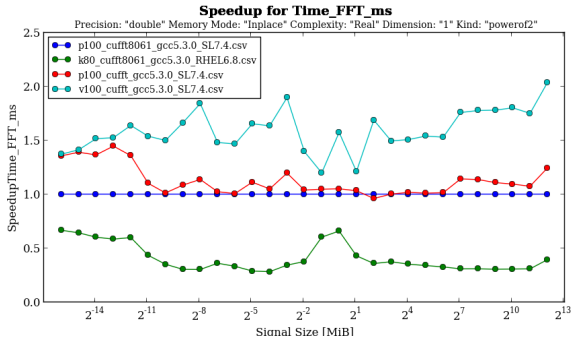

Fig. 6: Speedup of Time FFT for P100 with old (blue) and new (red) versions of cuFFT, K80 (grean) and V100 (cyan). The P100 GPU is approximately two times faster than the previous generation Kepler K80 GPU and the next generation NVIDA Tesla Volta V100 GPU is 1.5x faster on FFT calculations as compared to the NVIDA Tesla Pascal P100 GPU.

C. Comparison between CPUs

From figures 7 and 8 we can see the advantages of having a many-core processor working with size of data larger than 1 MB. Even when Xeon (2.3 GHz) is faster thank KNL (1.4 GHz), having 272 threads and a SIMD unit of length 8 (twice the SIMD unit of Haswell) helps to use less time with larger signal sizes. Multithreading enhanced performance by decreased development time, run simultaneous and parallelized occurrence of tasks, creates better of cache storage by utilization of resources.

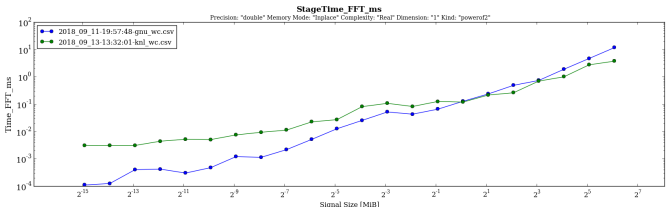

Fig. 7: The performance of FFTW + MKL run on the Intel Haswell CPU is compared with the Intel KNL CPU shown for inplace memory as a function of signal size. Using FFTW on the Haswell CPU gives faster FFT performance for smaller signal sizes but slower performance for larger signal sizes as compared to run in KNL.

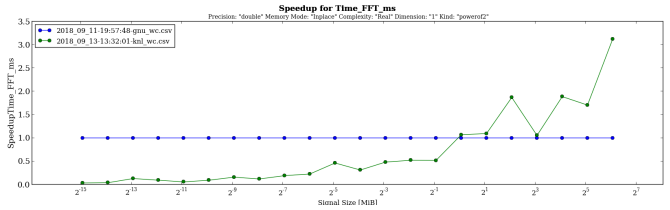

Fig. 8: Speedup of Execution Time of FFT using FFTW + MKL run on the Intel Haswell CPU is compared with the Intel KNL CPU.

D. Comparison between GPU and CPU

In figure 9 we compare FFT runtimes for data that is inplace and outplace as a function of signal size. We see that the Haswell CPU is faster for signal sizes less than 2^{-5} MiB and slower for larger signal sizes. Which tells that the relative performance of running in CPU and GPU is a strong function of signal size. For small signal sizes, the P100 can be 1-2 order of magnitude slower than the Haswell CPU but can be an order of magnitude faster for large signal sizes. Also we can notice that the impact of changing precision or memory mode, i.e., inplace vs outplace, on total time is much larger on the Haswell CPU as compared to P100 GPU.

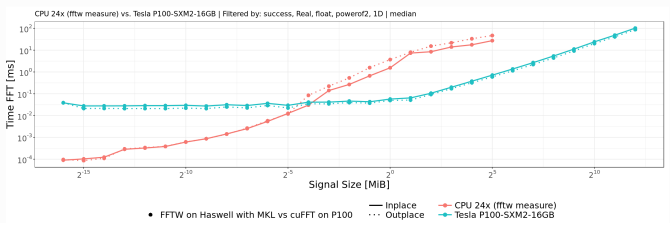

Fig. 9: The performance of cuFFT run on the Hōkūle'a NVIDA P100 GPU is compared to FFTw with Intel MKL wrappers run on an Intel Haswell CPU is shown for both inplace and outplace memory as a function of signal size. Using FFTw on the Haswell CPU gives faster FFT performance for smaller signal sizes but slower performance for larger signal sizes as compared to run cuFFT on P100 GPU.

V. CONCLUSIONS

With this project we were able to analyze the importance of interconnects, where NVLink interconnect has a significant impact on the overall time for larger signal sizes through faster upload and download times, as future work it is good to keep analyzing the crossover point (32 B). The increase in performance going from the P100 to the V100 is around 1.2X for smaller signal sizes and increases to 1.5X for larger signal sizes. The difference in performance in going from the older K80 to the P100 is about twice as larger as the jump going from P100 to newer V100 GPUs. In contrast to comparing to older or newer NVIDA GPUs, we show that the relative performance of cuFFT running on a NVIDA P100 compared to FFTW running on Haswell CPU is a strong function of signal size and the impact of bus width and memory size. The relative performance of running Haswell or KNL is a also strong function of signal size, and all because of the advantages of DLP and having 272 threads and a SIMD unit of length 8 (twice the SIMD unit of Haswell).

REFERENCES

[1] J.W.Cooley and J.W.Tukey, "An algorithm for the machine computation of the complex Fourier series", Mathematics of Computation, vol.19.Arp 1965.pp.297–301.

[2] Naohiko Shimizu-Takehiko Watanabe, "High performance parallel FFT on distributed memory parallel computers", Journal of Supercomputing, vol. 15, 2000, pp.207-228.

[3] Guohua Ji, John Mellor, Crummey, Laksono Adhianto, William Iii, Chaoran Yang, "Implementation and Performance Evaluation of the HPC Challenge Benchmarks in Coarray Fortran 2.0", IEEE International Parallel and Distributed Processing Symposium, 2011, pp.1090-1100.

[4] Claudio Brunelli, Roberto Airoldi and Jari Nurmi, "Implementation and benchmarking of FFT algorithms on multicore platforms", International Symposium on System on Chip, 2010, pp.59-62.

[5] Z. Cui-xiang, H. Guo-qiang, and H. Ming-he, "Some New Parallel Fast Fourier Transform Algorithms", Proc. of the Sixth International Conference on Parallel and Distributed Computing, Applications and Technologies, 2005 (PDCAT 2005), pp.624–628.

[6] Matteo Frigo, Steven G. Johnson. "FFTW: An Adaptive Software Architecture for the FFT". Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 1998, pp.1381-1384.

[7] Peter Steinbach, Matthias Werner, "gearshifft – The FFT Benchmark Suite for Heterogeneous Platforms", ISC High Performance 2017, pp. 1-19.