Comparison of CPU and Discrete GPU FFT Performance in Bellman

Finality Labs

This report details the initial findings of implementing GPGPU Fast Fourier Transform (FFT) algorithms for the Discrete Fourier Transform (DFT) over a finite field. The evaluation domain in our arithmetics library is that of the one used in the Bellman implementation of the BLS12-381 curve with sub group F_r chosen for efficiency.

Setup

Rust-lang NVPTX support is at Tier 3 limiting us from implementing a CUDA + Bellman implementation. Opencl supports both AMD and Nvidia hardware with comparable performance.

In order to focus on the FFT portion of Bellman, we initially constructed a test circuit that simply computed and proved knowledge of the output of 2^x=y to fill the polynomial evaluations with values. We later moved to a consistency test directly in the Bellman library that fills the coefficients with random finite field values.

Hardware:

--CPU--
CPU1: Quad core Intel Xeon E3-1241 v3 (-HT-MCP-) cache: 8192 KB
clock speeds: max: 3900 MHz 1: 3500 MHz 2: 3500 MHz 3: 3500 MHz
4: 3500 MHz 5: 3500 MHz 6: 3501 MHz 7: 3500 MHz 8: 3500 MHz
CPU2: Hexa core Intel Xeon E5-1650 v2 (-HT-MCP-) cache: 12288 KB
clock speeds: max: 3900 MHz 1: 1299 MHz 2: 1803 MHz 3: 1455 MHz
4: 1552 MHz 5: 1846 MHz 6: 1299 MHz 7: 1212 MHz 8: 1230 MHz
9: 1342 MHz 10: 1231 MHz 11: 1289 MHz 12: 1296 MHz
CPU3: Intel i7 6900k 8 cores 16 threads @3.20GHz

--GPU--
GPU1: Radeon RX 480 2G Mem
GPU2: Nvidia 750ti 2G Mem
GPU3: Nvidia 1080i 8G Mem

Implementation

All previous known GPU FFT implementations were done on the Complex Number domains requiring only two single precision float2s. Bellman implementation requires FF Integer domain elements of 256 bit length over eight uint32s. Our custom implementation of an opencl uint256 for BLS12-381 F_r field with large 2^n roots of unity can be found here.

We make use of “Decimation-in-Frequency” (DIF) in our various radix degree tests rather than Bellman’s “Decimation-in-Time” (DIT) over a radix 2 fft. Processing multiple iterations of the DFT per kernel is done to reduce the read+write times between kernel calls as each iteration of the FFT requires source and destination buffer exchange. An in-place and local memory allocated algorithm was explored but is significantly more involved and left for future work.

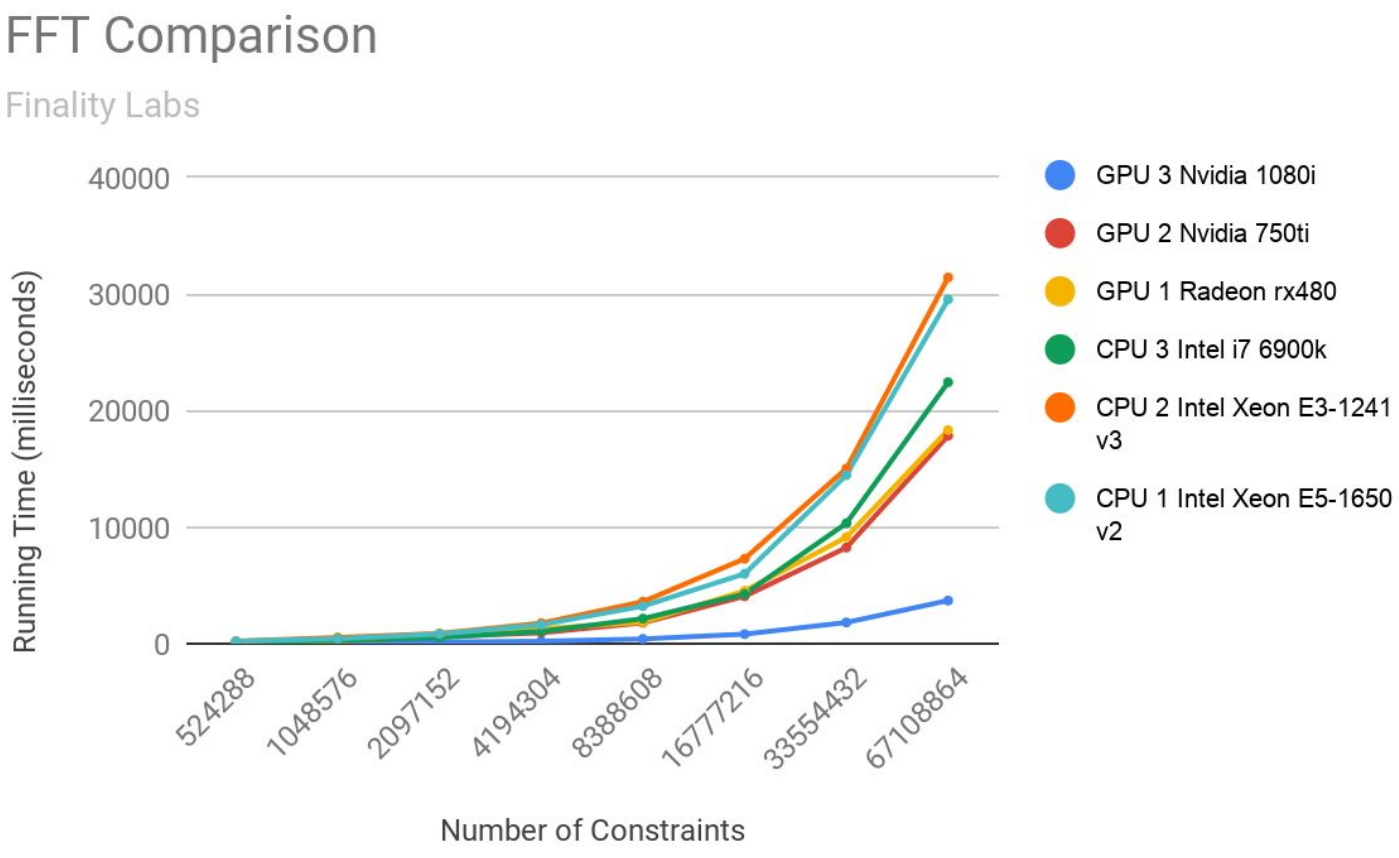
Our chosen Nvidia cards support local work sizes up to 1024 with prefered warp size multiples of 32 threads for maximum memory coalescence while the Radeon card maxes out at 128 with wave fronts of 64 threads. The implementation supports dynamically switching work group sizes and can detect GPUs to select optimal settings. Users may need to manually force drivers to allocate max memory (`GPU_MAX_ALLOC_PERCENT`) given the size of FFT input and auxiliary buffers.

This implementation is integrated into Bellman and is a drop-in improvement for any system using Bellman. Any host with opencl and a discrete GPU powerful enough will see prover time and parameter generation speed improvements.

Results

Our results show that FFT computation time improvement amount greatly depends on hardware, where Bellman already attempts to make use of parallel computing across multiple CPU cores. High end CPUs with 16+ concurrent threads perform with comparable numbers to low-mid end GPUs, but In all cases of hardware tested it is faster to use the 1080i GPU. Currently the expected FFT speed up is ~10x.

We find that processing 256 elements per kernel to be an optimal degree for n < 2^{26} elements and 1024 elements per kernel for n= 2^{27} . A single 8G card will OOM for n= 2^{28} . Nvidia cards of similar architecture time roughly the same while the 1080i out performs all other hardware tested.



Device comparison across three CPUs and three GPUs.

Further Improvements

The majority of prover time complexity is spent in FF modular multiplications needed to scale and produce products of polynomials with large degree. Future work to effectively speed Bellman prover times with the GPU will focus on bringing those multiplications in parallel to the GPU along with other elliptic curve operations on BLS12-381.

Optimizing the uint256 big arithmetics library is possible. A significant chunk of time is spent moving buffer values between CPU and GPU, further work on reducing this or utilizing the local memory register speeds could yield faster kernel results.