VkFFT – Vulkan/CUDA/HIP/OpenCL/Level Zero GPU FFT library

Dmitrii Tolmachev[1]
[1] EPM Group, ETH Zurich, Switzerland, dmitrii.tolmachev@erdw.ethz.ch

ETH zürich · Earth and Planetary Magnetism Group (Institute of Geophysics) · CSCS · erc European Research Council Established by the European Commission

1 Introduction

VkFFT is an efficient GPU-accelerated multidimensional Fast Fourier Transform library for Vulkan/CUDA/HIP/OpenCL/Level Zero projects. VkFFT aims to provide the community with a cross-platform open-source alternative to vendor-specific solutions while achieving comparable or better performance. VkFFT is released under MIT license.

2 Theory

Discrete Fourier Transform is defined as:

$$X_k = \sum_{n=1}^{N-1} x_n \, e^{-\frac{2\pi i}{N}nk}$$

The fastest known algorithm for evaluating the DFT is known as Fast Fourier Transform. The Cooley-Tukey algorithm reformulates a composite size DFT $N = N_1 \cdot N_2$ as a combination of two DFTs:

1. Perform N_1 DFTs of size N_2 - radix of transformation
2. Perform O(N) multiplications by twiddle factors - complex roots of unity defined by the radix
3. Perform N_2 DFTs of size N_1

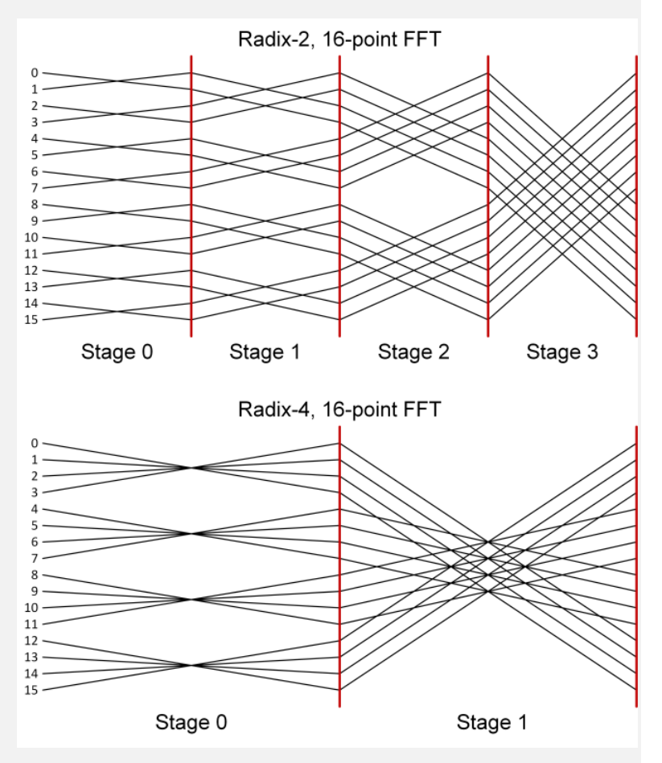
Recursively applying the algorithm, FFT can be done with a collection of small prime radix routines, written explicitly.

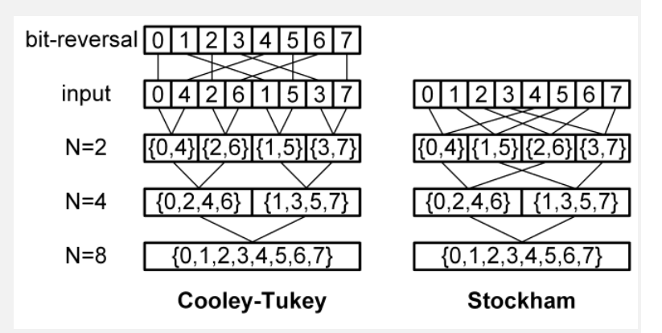
VkFFT uses a Stockham autosort version of the FFT algorithm, that has a better data-access pattern for GPUs by shuffling data between threads at each step instead of a single initial bit-reversal.

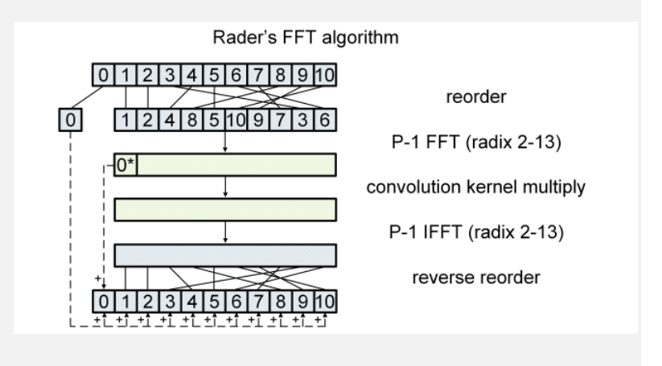
If a DFT has a prime radix that has no explicit routine (which is the case for big primes numbers), the Rader's FFT algorithm is used, calculating arbitrary prime radix as a $P-1$ length convolution, using convolution theorem: DFT$\{f * g\}$ = DFT$\{f\}$ · DFT$\{g\}$

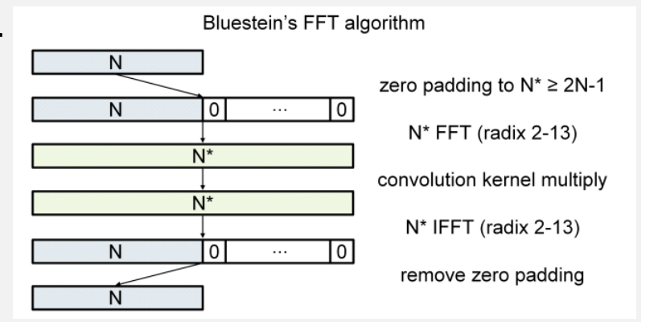
If $P-1$ is not decomposable as small primes (which is the case for Sophie Germain primes) Bluestein's FFT algorithm is used:

1. Pad sequence with zeros to a size of at least $2N-1$, that can be decomposed with supported prime radix kernels
2. Perform a convolution with a precomputed Bluestein's kernel








Bluestein's FFT algorithm
zero padding to N* ≥ 2N-1
N* FFT (radix 2-13)
convolution kernel multiply
N* IFFT (radix 2-13)
remove zero padding

3 VkFFT functionality

- 1D/2D/3D systems, forward and inverse directions of FFT.
- Support for big FFT dimension sizes: (2^32, 2^32, 2^32)
- Radix-2/3/4/5/7/8/11/13 FFTs and composite radix kernels.
- **New:** Rader's and Bluestein's FFT algorithms for all other sequences.
- Single, double and half-precision support.
- Complex to complex (C2C), real to complex (R2C), complex to real (C2R) transformations
- **New:** real to real (R2R) Discrete Cosine Transformations of types I, II, III and IV.
- Convolutions and cross-correlations.
- Native zero padding to model open systems.
- Works on Nvidia, AMD, Intel, Apple and mobile GPUs. And Raspberry Pi GPU.
- Works on Windows, Linux and macOS.
- VkFFT supports Vulkan, CUDA, HIP, OpenCL and Level Zero as backends.

4 VkFFT library design and API

VkFFT has a **hierarchical structure design**: Application ⇨ Plan ⇨ Code. This allows to make **code optimizations** for the target device architecture at runtime. Multiple levels of kernels that can be **merged and reused.** Below a more detailed description of the VkFFT platform structure is given.

Application manager
Input configuration, calls for plans' initialization and dispatch, binaries and resources management. Can contain multiple plans. User interacts with VkFFT through the application.

Plan manager
Optimization of parameters (example: number of threads, shared memory, LUT allocation) for a particular task and GPU architecture, calls for code generation and compilation.

Code manager
GPU kernel code generation. Produces a single kernel for the task defined by the plan manager.

Level 2 kernels
A clear description of the problem via a sequence of calls to lower levels, kernel layout configuration.

Level 1 kernels
Simple routines: matrix-vector multiplication, FFT, pre- and post-processing, R2C/R2R mappings.

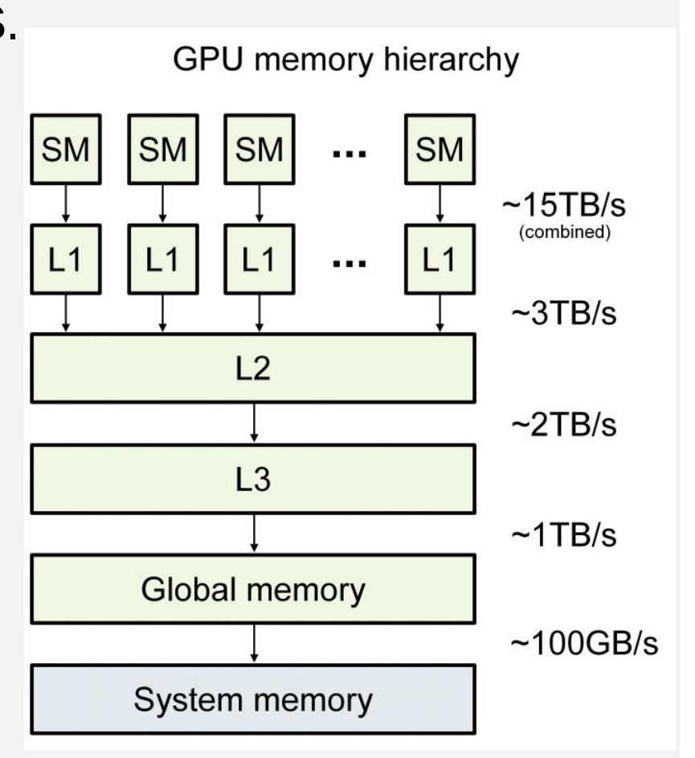
Level 0 kernels
Memory management, basic math functions inlining, subgroup functions, API-dependent definitions.

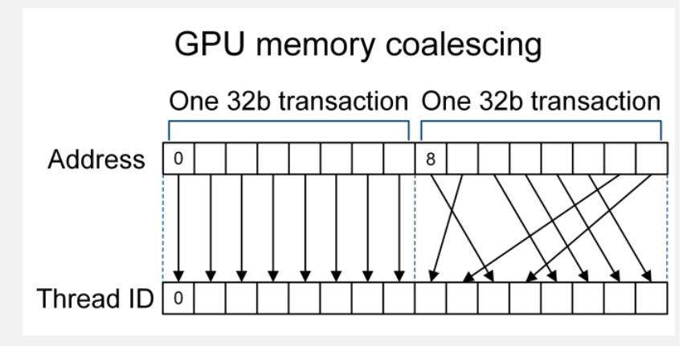
The platform will soon be released as a standalone project with VkFFT being a part of its kernel collection. With it, it will be possible to inline VkFFT in the user's kernels, reducing memory transfers even more.

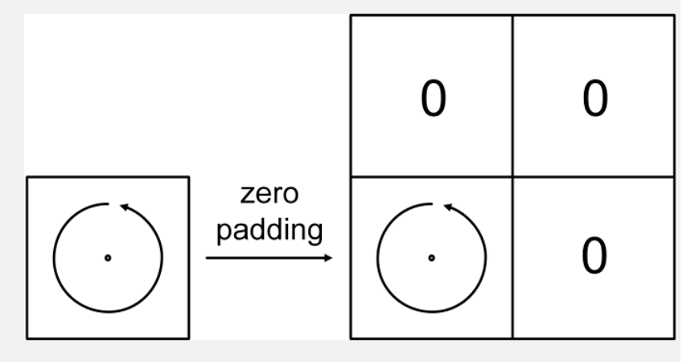
5 Memory management techniques implemented in VkFFT

GPUs employ a wide range of memory types to achieve high bandwidth and FLOPS. FFT is an extremely global memory bandwidth-limited algorithm, which means having two global memory data round trips instead of one often decreases performance by a factor of two. VkFFT incorporates the following techniques to reduce memory transfers:

- All FFT sequences that can fit in shared memory are done as a single upload from global memory. Including all advanced FFT algorithms, like Rader's/Bluestein's algorithms.
- Big sequences use the four-step FFT algorithm – the sequence is split into two (three for even bigger sizes) uploads with an inlined twiddle multiplication.
- VkFFT does multiple small FFTs at once – increases thread block size and allows to access continuous blocks of memory.
- VkFFT does not have a transposition for the strided FFT axes – they are done by coalescing the neighboring sequences. Strided FFTs are done in a single upload/download, just like non-strided FFTs.
- Zero padding support - VkFFT can omit sequences full of zeros and do not perform the corresponding memory transfers and computations, as the output result will be zero. This way it is possible to get up to two times the speed increase in the 2D case and up to 3x increase in the 3D case.
- Removing additional last forward FFT/first inverse FFT memory requests for convolutions by inlining kernel multiplication in the generated code. Removes one data round-trip.
- VkFFT utilizes R2C/C2R Hermitian symmetry properties. Reduces calculations and data transfers by a factor of two.
- All R2R pre- and post-processing are inlined in the FFT algorithm.





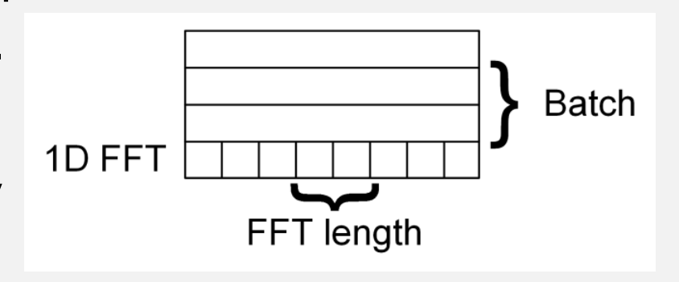


6 Nvidia A100 Performance

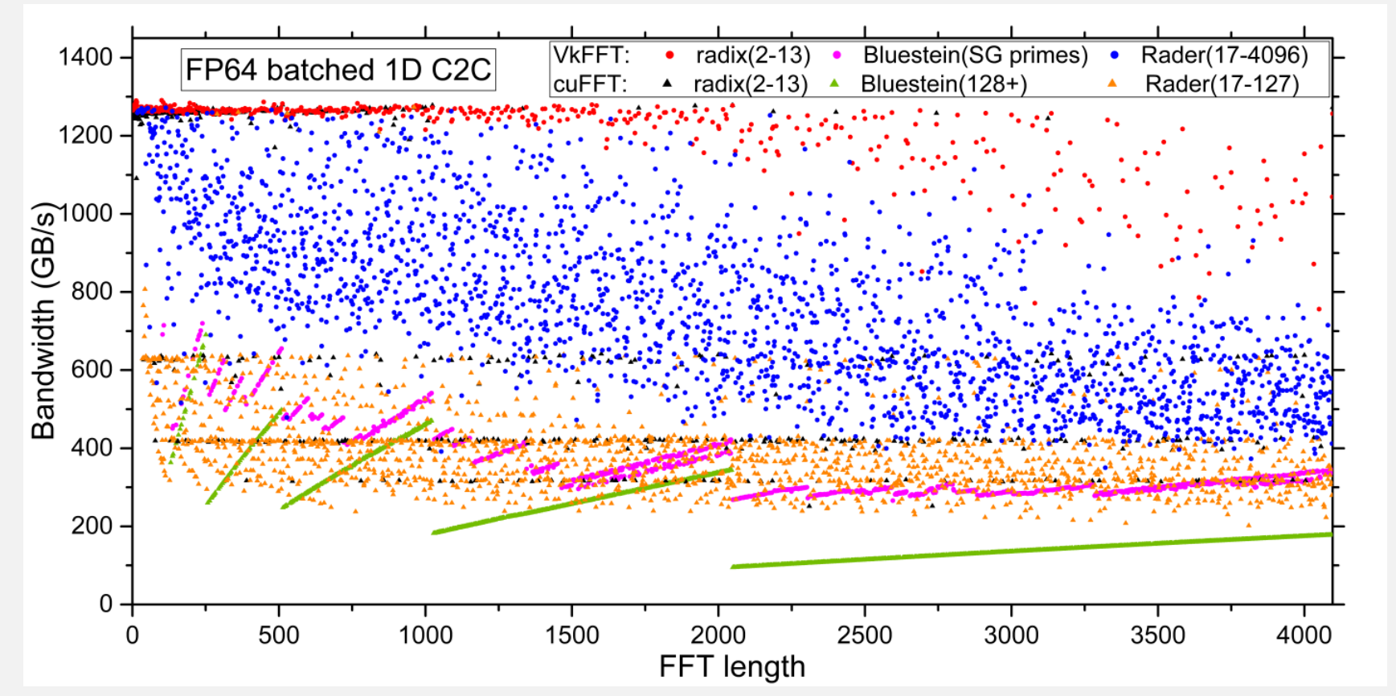
We compare the VkFFT performance against Nvidia's cuFFT on Nvidia A100 HPC GPU (40GB, 250W, P0 profile, CUDA 11.7)

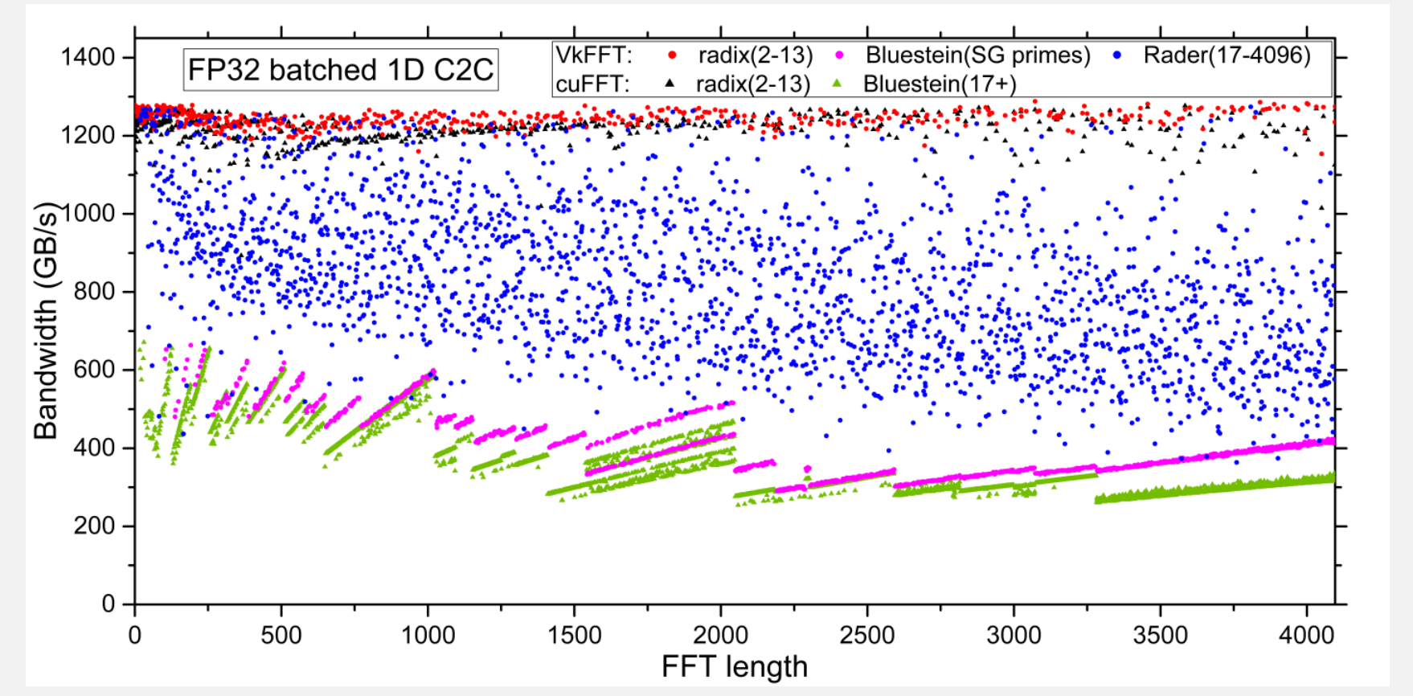
The test configuration takes multiple 1D FFTs of all lengths from the 2 to 4096 range, batches them together so the full system takes ~ 500MB to 1GB of data and performs multiple consecutive FFTs.

After that, time per single FFT is obtained by averaging the result. Total system size is divided by the time taken by a single transform, resulting in the achieved bandwidth. The peak global memory bandwidth of A100 is ~1.3TB/s.



Tests are performed in double (left) and single (right) precision.





For double precision, both VkFFT and cuFFT use radix decomposition for sequences representable as a multiplication of arbitrary number of primes up to 13.

Both VkFFT and cuFFT have Rader's algorithm implementation. VkFFT uses FFT version of it for sequences decomposable as a multiplication of primes up to 4096 (if P-1 FFT can be done with the first algorithm). cuFFT only uses Rader's algorithm for primes up to 127 and implements it as a direct matrix multiplication.

In both codes Bluestein's algorithm is used for all other sequences - Sophie Germain primes and their multiples for VkFFT and primes after 128 and their multiples for cuFFT.

For single precision, VkFFT uses the same algorithm configuration, as for double precision (radix+Rader+Bluestein).

cuFFT does not use Rader's algorithm in FP32 and switches to Bluestein's algorithm for primes after 17. Rader's algorithm implementation in VkFFT works just as well in FP32 as in FP64.

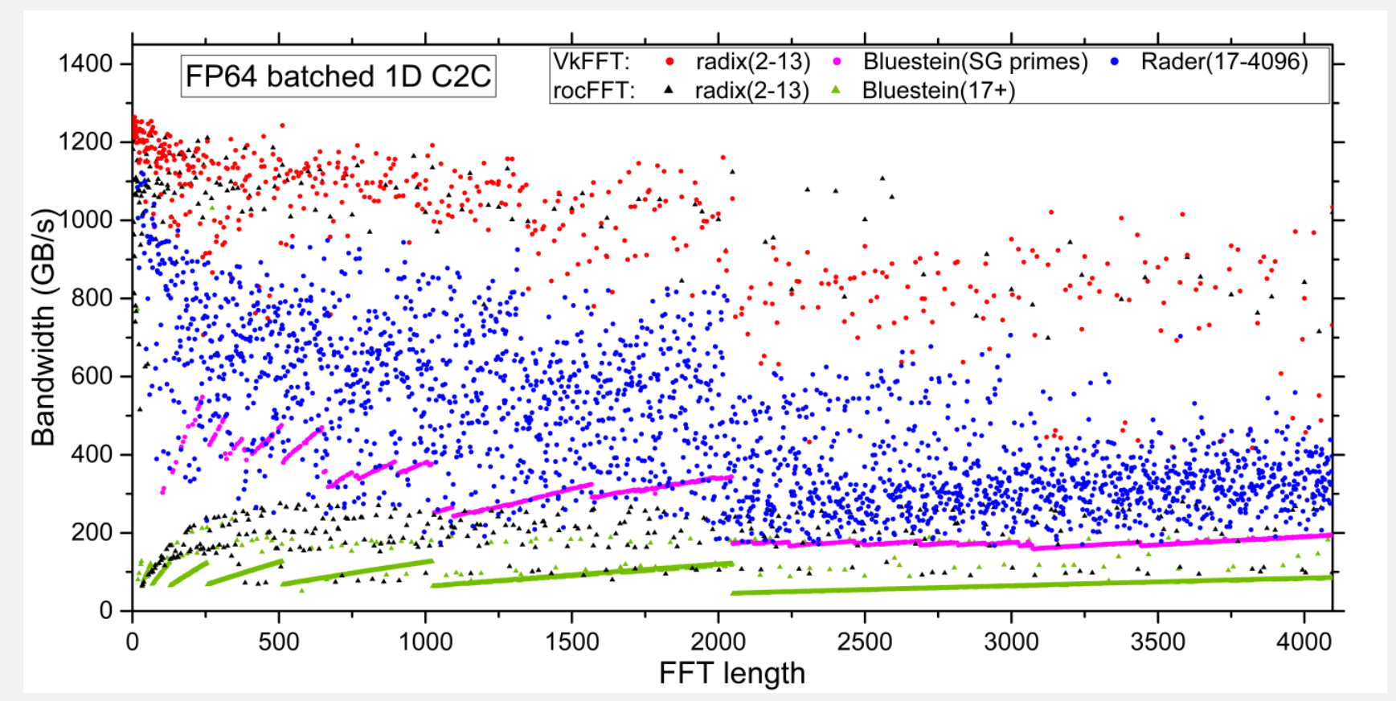
In Bluestein's algorithm VkFFT has an option to manually decide to which sequence to pad for the best per-GPU performance.

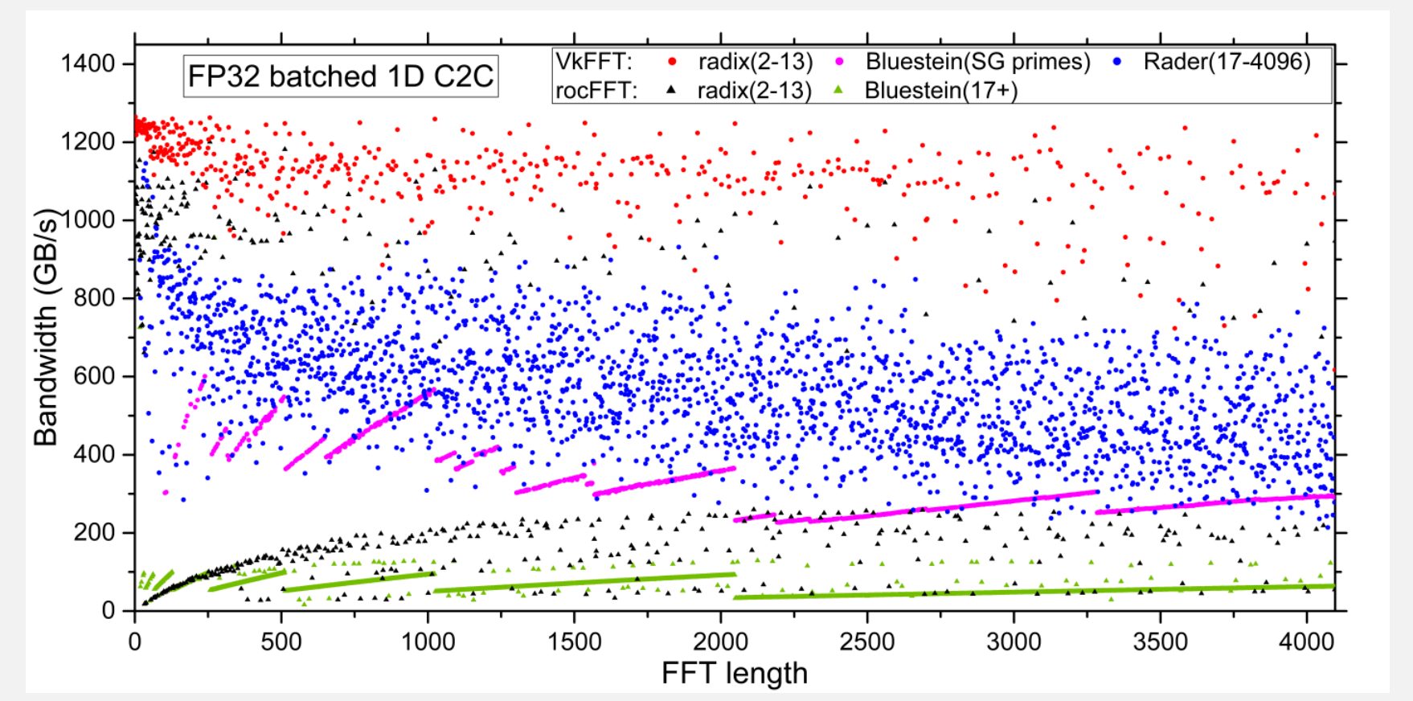
Overall, for both FP32 and FP64, VkFFT has better coverage for radix (2-13) decomposition, better Bluestein's algorithm performance and better Rader's algorithm prime coverage and performance.

7 AMD MI250 Performance

We compare the VkFFT performance against AMD's rocFFT on AMD MI250 HPC GPU (64GB, 250W, single-chip, ROCm 5.2)

The test configuration is the same as for the Nvidia A100 GPU above. The peak global memory bandwidth of MI250 is ~1.3TB/s.





For double precision, VkFFT uses similar to A100 configuration: radix decomposition for primes up to 13, Rader-FFT algorithm for non-Sophie Germain primes up to 4096, Bluestein's algorithm for other sequences.

Due to the smaller size of available shared memory compared to A100 (64KB vs 192KB), VkFFT has to switch to double-upload scheme for Bluestein's algorithm after 2048 in FP64 and 4096 in FP32, reducing effective bandwidth up to 2x.

For single precision, both VkFFT and rocFFT have patterns similar to double precision.

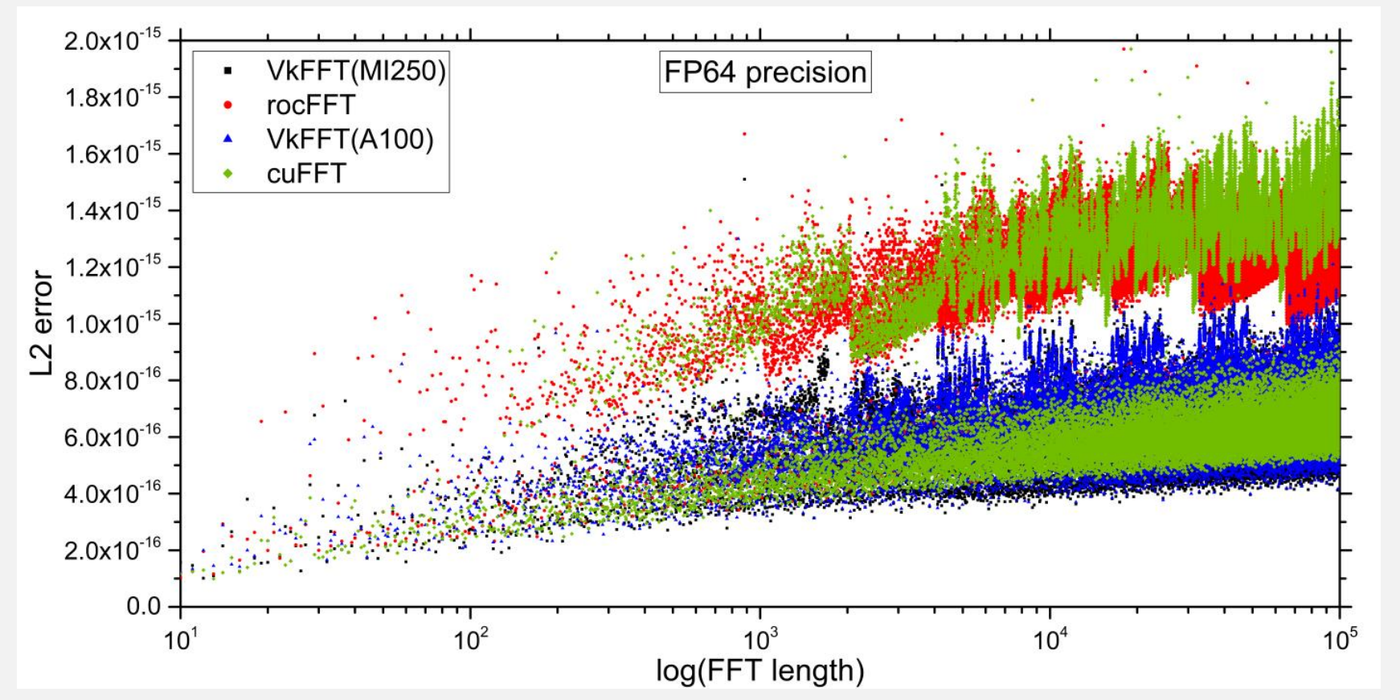
VkFFT achieves close to peak bandwidth for radix decomposition, only limited by less refined architecture (compared to A100), which is harder to optimize for.

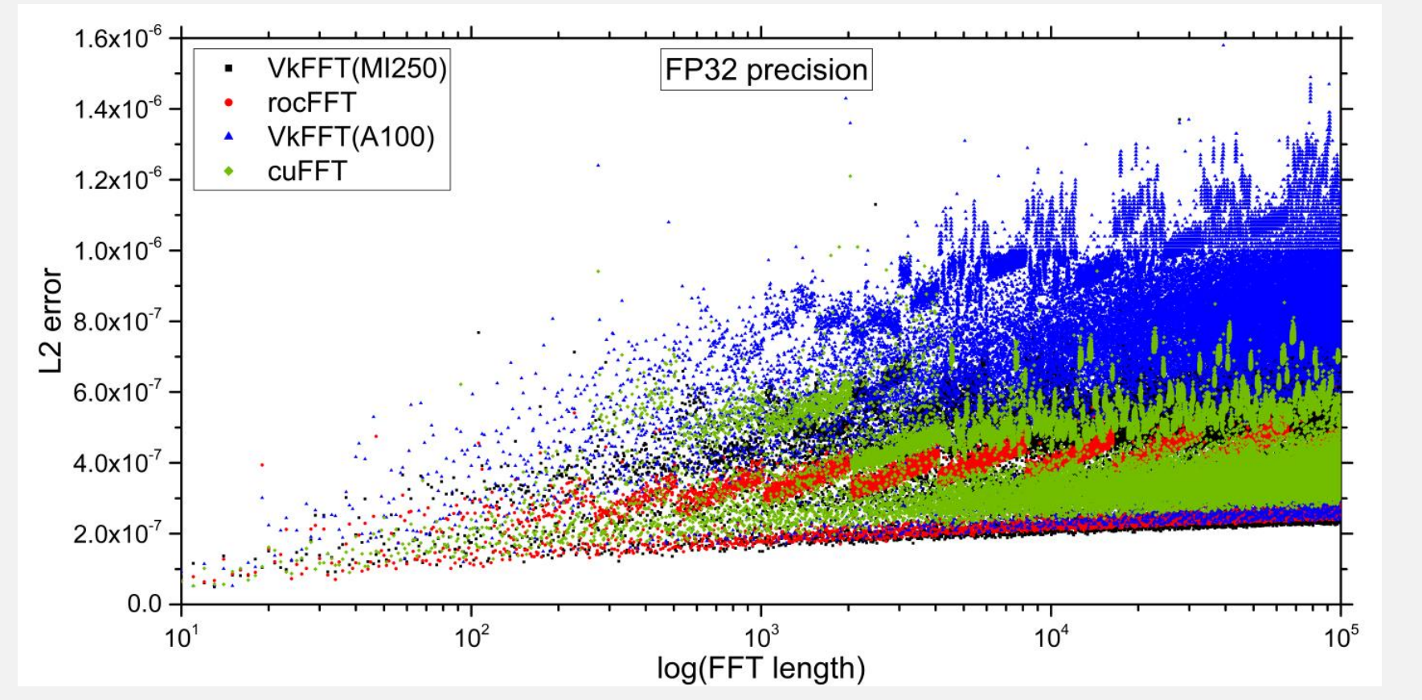
rocFFT does not have Rader's algorithm implementation.

VkFFT has ~2-4x faster Bluestein's algorithm implementation.

8 Precision verification

VkFFT precision is verified by comparing its results with FP128 version of FFTW. We test all FFT lengths from the [2, 100000] range. We perform tests in single and double precision on random input data from [-1;1] range.





For both precisions, all tested libraries exhibit logarithmic error scaling. The main source of error is imprecise twiddle factor computation – sines and cosines used by FFT algorithms. For FP64 they are calculated on the CPU either in FP128 or in FP64 and stored in the lookup tables. With FP128 precomputation (left) VkFFT is more precise than cuFFT and rocFFT.

For FP32, twiddle factors can be calculated on-the-fly in FP32 or precomputed in FP64/FP32. With FP32 twiddle factors (right) VkFFT is slightly less precise in Bluestein's and Rader's algorithms. If needed, this can be solved with FP64 precomputation.

9 Discrete Cosine Transforms (R2R)

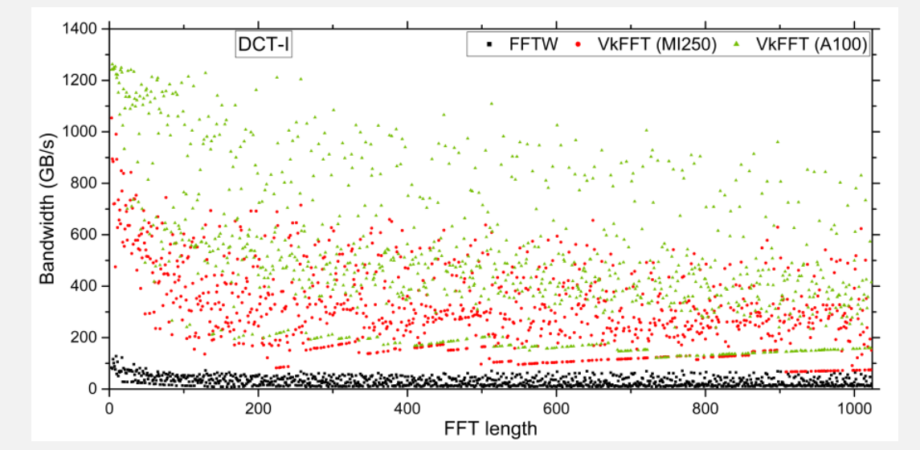
Discrete Cosine Transforms are defined as:

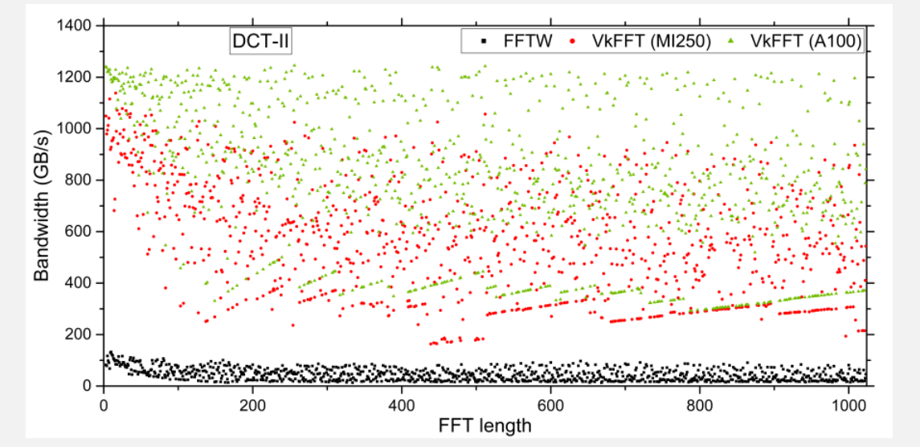
- **DCT-I:** $X_k = x_0 + (-1)^k x_{N-1} + 2\sum_{n=1}^{N-2} x_n \cos\left(\frac{\pi}{N-1}nk\right)$
- **DCT-II:** $X_k = 2\sum_{n=1}^{N-1} x_n \cos\left(\frac{\pi}{N}\left(n+\frac{1}{2}\right)k\right)$, inverse of **DCT-III**
- **DCT-IV:** $X_k = 2\sum_{n=1}^{N-1} x_n \cos\left(\frac{\pi}{N}\left(n+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)\right)$

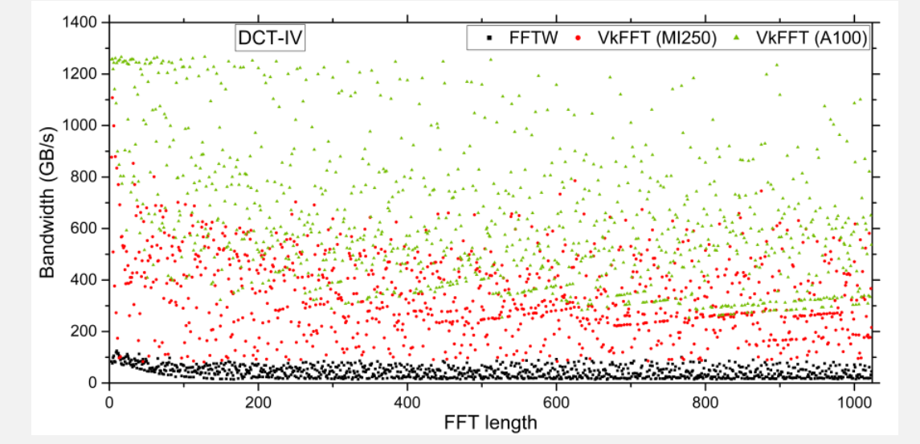
- No existing fast GPU DCT library before.
- Implemented as a mapping between DCT and a regular Fourier transform of a similar length, performed with VkFFT.
- Mapping is done in shared memory as a part of a generated FFT kernel with no additional global memory transfers.
- The same $O(logN)$ accuracy scaling as FFTW.

The test configuration is the same as for the C2C in double precision. We compare the performance of AMD EPYC 7742 (64 cores) CPU with threaded FFTW with Nvidia A100 and AMD MI250 GPUs with VkFFT.

The high bandwidth of GPU memory allows to greatly outperform CPU implementation in FFTW.







10 Conclusions

- Developed an open-source, cross-platform, optimized GPU FFT library VkFFT.
- VkFFT matches in performance with Nvidia's cuFFT library on Nvidia GPUs for small sequences and outperforms it on big ones.
- VkFFT outperforms AMD's rocFFT library.
- VkFFT has logarithmic error scaling as other FFT implementations
- First performant implementation of DCTs on GPUs.
- VkFFT is available on GitHub: https://github.com/DTolm/VkFFT

11 Financial support

This project has received funding from the European Research Council under the European Union's Horizon 2020 research and innovation programme grant agreement No 833848 (UEMHP) and from The Swiss National Supercomputing Centre's PASC initiative supporting the AQUA-D software development project.

VkFFT GitHub: