2023 IEEE International Conference on High Performance Computing & Communications, Data Science & Systems, Smart City & Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)

2023 IEEE International Conference on High Performance Computing &amp; Communications, Data Science &amp; Systems, Smart City &amp; Dependability in Sensor, Cloud &amp; Big Data Systems &amp; Application (HPCC/DSS/SmartCity/DependSys) | 979-8-3503-3001-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/HPCC-DSS-SMARTCITY-DEPENDSYS60770.2023.00077

An Optimized GPU Kernel for Real-Time Radio Imaging

Karthik Reddy
School of Earth and Space Exploration
Arizona State University
Tempe, Arizona
karthik@asu.edu

Judd D. Bowman
School of Earth and Space Exploration
Arizona State University
Tempe, Arizona
judd.bowman@asu.edu

Adam P. Beardsley
Physics Department
Winona State University
Winona, Minnesota
adam.beardsley@winona.edu

Greg B. Taylor
Department of Physics and Astronomy
University of New Mexico
Albuquerque, New Mexico
gbtaylor@unm.edu

Jayce Dowell
Department of Physics and Astronomy
University of New Mexico
Albuquerque, New Mexico
jdowell@unm.edu

Craig Taylor
Department of Physics and Astronomy
University of New Mexico
Albuquerque, New Mexico
ctaylor98@unm.edu

***Abstract*—The E-Field Parallel Imaging Correlator (EPIC) is an algorithm designed to perform imaging at several frequencies using radio telescopes on much faster time scales $O(N \log N)$ than their traditional counterparts $O(N^2)$. This paper describes the new GPU kernel developed for a real-time EPIC-based imaging pipeline. Unlike the slower global memory used by the previous implementation, the kernel uses a new memory layout to store all intermediate products of the imaging sequence in on-chip memory to minimize time spent on memory transfers. Auxiliary data are stored in the shared memory for faster access. The imaging data is stored with 16-bit precision in the thread registers to reduce register pressure and simultaneously process two images. Each thread block processes one frequency, providing horizontal scalability. The optimized kernel processes 50000 all-sky images per second per frequency in real time. It is being deployed on the Long Wavelength Array in Sevilleta, New Mexico (LWA-SV).**

***Index Terms*—Radio telescope imaging, real-time, GPU acceleration, performance optimization**

I. Introduction

The need for concurrent wider imaging fields, higher sensitivity, and higher time resolution has expanded radio telescope arrays from a few tens to hundreds of antennas with dense layouts. Their correlators compute the integrated cross-power correlation between all antenna pairs for imaging. This computation scales as the square of the number of antennas, $\sim N_A^2$. Advances in Graphics Processing Unit (GPU)-based signal processing have allowed accelerating these intensive calculations [1, for example]. While modern telescopes with up to a few hundred antennas can be processed in correlators using a small set of GPUs, the associated computational complexity and costs of operation increase significantly with thousands of antennas planned in upcoming telescopes like the Square Kilometer Array (SKA) [2]. Hence, reducing the number of computations while maintaining the same scientific output is highly desirable.

For arrays with dense layouts, the direct imaging class of algorithms can perform a spatial Fast Fourier Transform (FFT) on the antenna-measured electric fields without calculating cross-correlations. These algorithms relax the number of computations from $\sim N_A^2$ to $\sim N_g \log N_g$, where N_g is the number of grid points in the FFT. Although direct imaging correlators were tested on a few telescopes [3], [4], they required identical antennas and uniform layouts. Conversely, the Modular Optimal Frequency Fourier (MOFF) [5] algorithm works with heterogeneous antennas with arbitrary layouts. MOFF outputs are identical to those produced with traditional correlators, and the algorithm is the optimal solution for direct imaging. EPIC is the first Python-based software implementation [6] of the MOFF algorithm that was later implemented on a GPU using CUDA [7]. Although the GPU version produced images of the sky in real-time, the output bandwidth was limited, and it imaged only a single polarization. This paper describes a new kernel optimized for NVIDIA GPUs that produces dual-polarized images with a much higher bandwidth. This kernel is a part of the new all-sky imaging pipeline under deployment at LWA-SV.

The rest of the paper is organized as follows. Section II summarizes the EPIC algorithm and describes the imaging requirements for LWA-SV. Section III describes the optimized GPU implementation of EPIC and presents performance benchmarks.

II. EPIC Algorithm

EPIC operates on data from a telescope's F-Engine, which digitizes and channelizes the electric field time series from each antenna using a temporal Fourier transform. Following that process, three spatial operations—gridding, inverse FFT, and cross-multiply and accumulate—are performed sequentially to transform the complex electric field spectra from the antennas into a single sky image for each time step.

Gridding is performed at each time step to interpolate the antenna data points (complex values) from arbitrary positions in the aperture of the telescope onto a uniform grid to allow the use of FFT. Each data point is convolved with its antenna beam pattern and sampled at regular grid points. Gridding is generally an atomic operation, as multiple overlapping patterns may add values to the same grid points. The gridded data is inverse Fourier transformed and squared element-wise to produce an image for each time step. We average these images (typically hundreds) until we reach a specified signal-to-noise ratio.

A. Imaging Considerations for LWA-SV

LWA-SV [8] observes the sky in the 3-88 MHz frequency range with 256 dual-polarized antennas (X and Y), or $N = 512$ effective interferometric elements. Polarization is a property of incoming radiation that specifies the magnitude and direction of the electric field oscillations. We can use the polarized radio waves to understand the environments of distant objects that are otherwise inaccessible to optical telescopes. EPIC can use X and Y polarizations and output images for all four instrumental cross-polarizations, namely, XX^*, YY^*, XY^*, YX^*. The F-Engine at LWA-SV digitizes and channelizes the electric field time series from each antenna using 40 μs windows to yield a channel width of 25 kHz. That means the imaging code must be able to process 25000 sky images per second for each channel and polarization. It is desirable to oversample the telescope's primary beam by at least a factor of two at all frequencies and generate images that cover the entire visible hemisphere. This oversampling needs a minimum output image size of 128 sq. pixels with a spatial resolution of 1°. The imager, therefore, requires a GPU with a performance of at least a few GFLOPs for each channel and polarization with this configuration.

The F-Engine at LWA-SV comprises six nodes, each providing channelized data with a bandwidth of 3.3 MHz (132 channels) or an effective bandwidth of 19.8 MHz distributed over the entire frequency range. The channelized data is supplied as 4+4-bit complex values, yielding a raw data rate of about 13.5 Gb/s per node or about 81 Gb/s in total. For a typical accumulation time of 40 ms (1000 time steps) with an image size of 128 sq. pixels, the output image data rate is about 10.4 Gb/s for 132 channels or 63.3 Gb/s for all channels. Top-end GPUs available in 2023 can accommodate at least an order of magnitude higher than these data rates and provide performances over several TFLOPS, indicating a possibility of imaging the entire bandwidth on a single GPU, although memory access latency considerations (see sections below) for gridding, FFT, and accumulation favor imaging with one GPU per node in the F-Engine.

B. NVIDIA GPU Programming Overview

At a high level, NVIDIA GPUs are composed of streaming multi-processors (SM), each consisting of multiple cores capable of running many threads in parallel. The Compute Unified Device Architecture (CUDA) programming model provides a platform to leverage massive parallelism built into these GPUs. It provides a thread and memory hierarchy to simplify GPU programming.

A block comprises a group of threads up to a maximum of 1024, and a grid consists of a group of blocks where all threads execute the same kernel. CUDA provides 3D-indexing support for blocks and grids to distribute work among the threads appropriately. GPUs also provide different types of memory with different speeds. Registers are the fastest memory type on a GPU, private to each thread, which kernel variables use for computations. All threads in a block can also access a slightly slower shared memory (L1 cache). This memory is extremely limited and is less than 128 KiB in nearly all the GPUs. CUDA also provides a global memory space that is accessible by all threads and is much larger (typically a few to hundreds of GBs) than register or shared memory. Global memory is generally 100 times slower than shared memory. Any memory required by threads above the available registers is spilled into the local memory, a part of the global memory, mapped through L1 and L2 caches.

GPU kernels are classified based on their performance into two types: compute-bound and memory-bound. Compute-bound kernels spend most of their run time doing compute work using as many active threads as possible with minimal memory accesses. Memory-bound kernels are further classified into two types: latency-bound, where memory access latency forms the main bottleneck, and bandwidth-bound, where the finite memory transfer speed between device and global memory limits the performance. Optimized GPU kernels generally minimize accesses to slow memory types and maximize the compute work done per byte fetched from the global memory. Put another way, one must make the kernel compute-bound or compute and bandwidth bound for maximized performance.

C. Bottlenecks in the Previous GPU Implementation

The first version of EPIC imager deployed on LWA comprised three GPU kernels, executed sequentially, for gridding, inverse FFT, and squaring and accumulating images. On an NVIDIA RTX 2080 Ti GPU, the imager generates 64 sq. pixel single polarization images with 90 channels in real-time, which falls below the desired throughput described in section II-A. While the highly optimized cuFFT library performs FFT, profiling indicates gridding and accumulation kernels are bandwidth-bound. It is mainly because each kernel operates on the output from its previous kernel: the gridder writes the gridded data for each timestep for all channels and polarizations to global memory that is then Fourier transformed and written back to the same memory by cuFFT. The transformed data are then cross-multiplied, written to a different global memory block, and finally averaged into a single image per channel per polarization. These intermediate products require large global memory allocations; profiling indicates that although these kernels use efficient memory access patterns, they consume about 30% of the total run time for global memory transfers, leading to lower throughput.

III. Optimized EPIC correlator

The optimizations described in this paper aim to enable the kernel to generate dual-polarized images in real time. The new code employs on-chip memory (registers and shared memory), which is at least two orders of magnitude faster than global memory, to store and process all intermediate products. Because kernel-allocated on-chip memory gets destroyed after the kernel exits, the new code fuses all operations in the imaging sequence into a single unified kernel. In what follows, we refer to this unified kernel as the kernel unless otherwise stated. Below, we describe the implementation for each operation in the kernel, including an optimal on-chip memory layout to facilitate data sharing between different operations.

A. On-chip Memory Layout

In this layout, the kernel stores intermediate products, which are 2D arrays, in register (32-bit) arrays, uniformly spread across all threads in a thread block. One 2D input array stores gridded data for two polarizations that will be Fourier transformed in place. Two 2D output arrays store compressed sky image accumulations for all four cross-polarizations. Input and output arrays have identical dimensions. Each element in the input array comprises two complex numbers, one for each polarization. Storing each value at 32-bit precision requires four registers. A 128x128 array, therefore, would consume 65536 registers, the maximum number of on-chip registers CUDA allows for a thread block on any GPU. In this case, the compiler may partially or fully allocate memory for input and output arrays in the much slower local memory to free some registers for variables used in computations. This race for registers between variables and storage arrays creates what is commonly called the "register pressure". It introduces memory access latencies, leading to inefficient GPU compute power usage. Reducing this register pressure drove the decision to use half (16-bit) precision over 32-bits for input and output arrays, which halves the required number of storage registers.

With half-precision, we store each input array element in two 32-bit registers in an RR-II sequence. That means the real and imaginary values of two polarizations for each element are batched into the same registers. This sequence is required for on-chip FFT calculations (see section III-C for details). We launch the kernel using 2D thread blocks where we uniformly distribute each row in the input array to registers in one row of threads with a stride between elements.

Figure 1 depicts this layout for a thread block (128x8 threads) that generates a 128x128 sq. pixel image. Each thread stores elements from its respective row in the input array with a 16-element stride. For example, thread (0,0) stores elements with indices 0, 16, 32, 48... from the first row in the input array. The two output arrays are arranged similarly to store four cross-polarization sky images. Each element in the first array stores two half-precision numbers in a single register, one each for XX^* and YY^*. Each element in the second array only stores the real and imaginary parts of XY^* at half-precision in a single register, and the remaining YX^* polarization is simply the complex conjugate of this value. That means we effectively compress six values, one each for XX^* and YY^*, and two each for XY^* and YX^*, into four.

Finally, this layout also leads to horizontal scalability because a single thread block can generate all image products for one channel. Hence, we can image multiple channels in parallel by launching multiple kernel thread blocks without introducing additional latency. The scaling reaches a limit when the GPU delays launching additional thread blocks due to the unavailability of registers to perform computations. Adding additional GPUs would allow imaging of an arbitrary number of channels, each with the same computation time.

B. Gridding

The previous kernel used nearest neighbor gridding to reduce global memory transfers. That means the data point from each antenna was assigned to the nearest cell in the aperture plane after scaling them with the appropriate antenna phases. This gridding is equivalent to convolving the data points with a pillbox or a constant function with the size of a single cell and sampling it at cell centers.

EPIC prescribes using antenna beam patterns as convolution kernels. The antenna kernel size depends on the frequency and ranges from 3x3 to 5x5 cells on an aperture plane with 1° spatial resolution. These kernel sizes are 9 to 25 times larger than a pillbox, indicating the pillbox kernel severely undersamples the electric field spectra.

The new kernel grids the data in shared memory and transfers it to the thread registers. Our tests (see section III-D) indicate this method supports kernel sizes up to 7x7 to generate 128 sq. pixel images. The kernel allocates a 2D array in the shared memory for use as scratch pad memory with the same format as the input array described in section III-A. We only allocate half the input array size to ensure the allocation is below the shared memory typically available on GPUs. For example, a 128x128 input array occupies 128 KiB, a shared memory size available only on a few of NVIDIA's GPUs (typical shared memory is between 64-100 KiB). The kernel grids data in the shared memory for the upper half of the aperture plane and transfers it to respective thread registers. It repeats the same procedure for the lower half, filling all registers with the gridded data.

The gridding function divides all threads in the thread block into groups of the number of cells covered by the gridding kernel (e.g., 25 threads per group). Each thread group grids one antenna at a time and atomically adds it to the array allocated in the shared memory. The thread groups grid all antennas covering the appropriate half of the aperture plane.

The gridding operation requires four inputs: antenna positions, phases (different for each polarization), kernel weights, and the electric field spectra. Their memory access times primarily determine the total run time of gridding. Antenna positions and phases have smaller memory sizes than typical shared memory sizes. For example, the 2D positions and phases for 256 antennas only require 2 KiB and 4 KiB, respectively. Hence, we copy the antenna positions and phases into the shared memory of each thread block once per accumulation. Because antenna positions remain fixed throughout the observation, we pre-compute kernel weights for each antenna

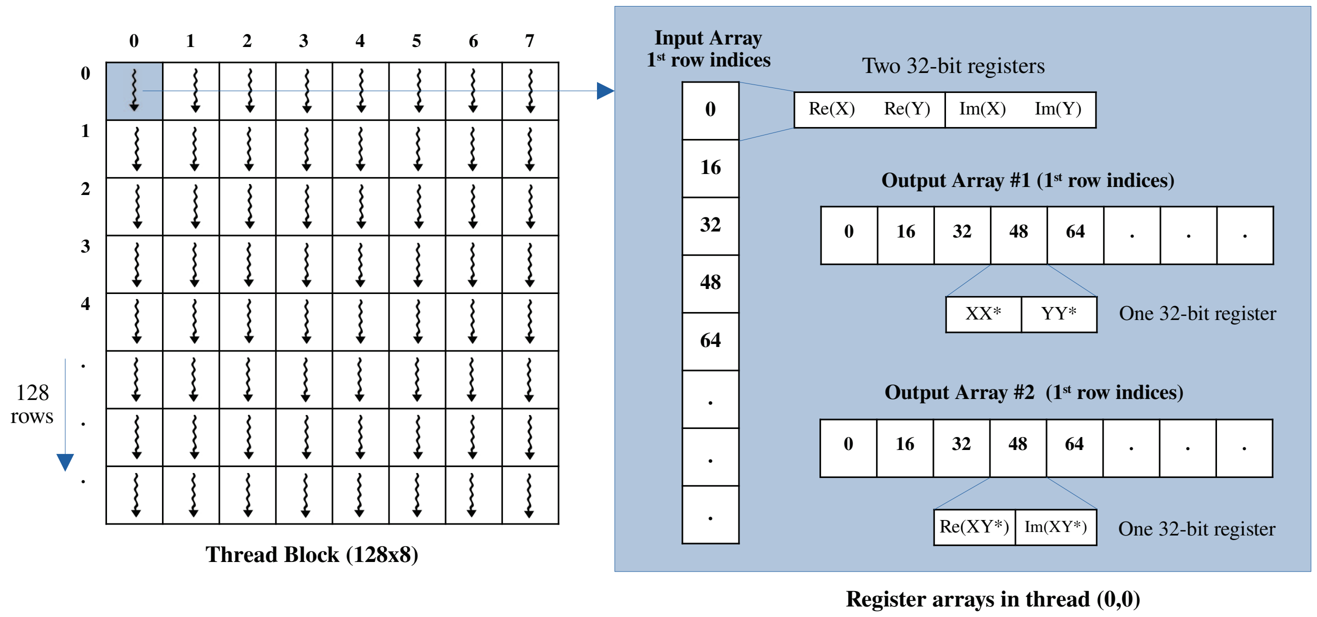

Fig. 1. On-chip memory layout for a kernel generating 128 sq. pixel images. The left image shows the logical layout of a 2D thread block with 128 rows and 8 columns. One input array and two output arrays are distributed across thread registers (see section III-A for details). The right image shows the register arrays allocated in thread (0,0). This thread stores elements from the first row of the input and output arrays with a stride of 16 elements between them. Each element in the input array is a set of two half-precision complex numbers, one for each polarization, stored in two 32-bit registers with an RR-II layout. Each output element is a set of two half-precision numbers stored in a 32-bit register. The first output array stores values for XX* and YY*, and the second one stores the real and imaginary parts of XY*.

and frequency. For a gridding kernel with a size of 5x5, the weights for 256 antennas occupy a space of about 26 KiB per channel, which we also copy to the shared memory. We can store the weights in the global memory for GPUs with smaller shared memory and map them through the L2 cache.

C. *FFT and Accumulation*

We use a specialized library called cuFFTDx, distributed as a part of the NVIDIA MathDx package, to perform FFT completely using on-chip memory. cuFFTDx uses the batched half-precision register layout described in section III-A for efficient calculations. It performs 1D FFT individually on each row of threads or equivalently on each row of the input register array and writes back the transformed values into the same registers. It uses shared memory as a scratch pad memory for all calculations. The required space depends on the FFT size and is equal to 64 KiB for two 128x128 arrays.

The kernel obtains 2D FFT of the aperture plane by performing row-wise and column-wise FFT on the input array. Column-wise FFT is row-wise FFT performed on the transposed input array. The kernel transposes the array by exchanging the upper and lower triangles in three steps. Each thread with registers in the upper triangle copies the upper triangle values into the shared memory. The threads in the lower triangle then swap values of the appropriate registers with values of the upper triangle. Finally, the upper-triangle threads copy back the lower-triangle values from the shared memory. During the transpose, we also normalize all register values with N^2, where N is the total number of elements in the input array. Normalizing values prevents floating-point overflow because we perform all calculations at half-precision. After the Fourier transform, all threads cross-multiply values in each register to generate images and add them to the output array described in section III-A. Once the desired number of images are accumulated, each thread collects the four output values in every pixel into `float4` vectors and transfers them to the global memory for further processing.

D. *Performance Benchmarks*

The kernel achieves the desired throughput of 128 sq. pixel images with all four cross-polarizations per channel in real-time. At LWA-SV, we generate sky images with a 40 ms cadence. That means the kernel accumulates images for 1000 time steps to produce one sky image per channel or effectively processing 50000 (25000 per polarization) images per second on a single thread block. On an NVIDIA RTX 4090 GPU, the kernel takes about 35 ms to process these 1000 images spanning 40 ms with a 5x5 gridding kernel size (39 ms for a 7x7 gridding kernel). This processing time leaves more than 10% of the gulp duration to account for any GPU throttling and other latencies the operating system introduces.

We profiled the kernel to quantify its GPU resource and power utilization. The kernel spends more than 98% of the time on computations and less than 2% on global memory transfers, which is much smaller than ~30% spent by the previous kernel. The kernel utilizes 87% of the GPU compute resources and less than 30% of its memory bandwidth. It also requires a global memory space of less than 600 MiB, which is about 100 times smaller than what would be needed for the previous kernel to generate the same imaging output. The GPU draws a power of 280 W in a steady state. Future optimizations will aim to reduce the GPU power consumption while maintaining the same throughput. This will also include reducing the GPU voltages without affecting the GPU clock speeds.

Although on-chip memory enables the kernel to operate in real-time, it also limits the number of channels that can be imaged on the RTX 4090 to slightly below the desired level. Each thread block consumes all the registers available on an SM, forcing the GPU to launch only one thread block per SM. An RTX 4090 contains 128 SMs, allowing us to image 128 channels on a single GPU, four channels short of the 132 channels produced on each node in the F-engine. The kernel will be deployed on 6 GPUs that deliver a combined bandwidth of 19.2 MHz, which is only 0.6 MHz lower than the expected 19.8 MHz bandwidth and has essentially no effect on the planned science objectives of detecting radio transient events.

IV. Conclusions

We implemented an optimized GPU kernel for an EPIC-based real-time radio imager we deploy on LWA-SV. It can simultaneously generate images for all four cross-polarizations for one channel in a single thread block. The kernel generates 128 sq. pixel images with 1° spatial resolution and a bandwidth of 3.2 MHz per GPU, which meets the desired output for operation at LWA-SV. It does so with an optimal on-chip memory layout that stores intermediate imaging products at half-precision within thread registers. The kernel uses shared memory as a scratch pad for calculations and stores all auxiliary data within it, eliminating the need for global memory transfers during accumulation. On an RTX 4090, the kernel processes 50000 images per second on a single thread block. It has a global memory footprint 100 times smaller than the previous implementation to produce an identical output. The code is open source and available in github[1].

References

[1] N. Denman, M. Amiri, K. Bandura, L. Connor, M. Dobbs, M. Fandino, M. Halpern, A. Hincks, G. Hinshaw, C. Höfer *et al.*, "A gpu-based correlator x-engine implemented on the chime pathfinder," in *2015 IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2015, pp. 35–40.

[2] J. Lazio, "The square kilometer array," in *AIP Conference Proceedings*, vol. 1035, no. 1. American Institute of Physics, 2008, pp. 303–309.

[3] G. Foster, J. Hickish, A. Magro, D. Price, and K. Zarb Adami, "Implementation of a direct-imaging and fx correlator for the best-2 array," *Monthly Notices of the Royal Astronomical Society*, vol. 439, no. 3, pp. 3180–3188, 2014.

[4] H. Zheng, M. Tegmark, V. Buza, J. S. Dillon, H. Gharibyan, J. Hickish, E. Kunz, A. Liu, J. Losh, A. Lutomirski *et al.*, "Miteor: a scalable interferometer for precision 21 cm cosmology," *Monthly Notices of the Royal Astronomical Society*, vol. 445, no. 2, pp. 1084–1103, 2014.

[5] M. F. Morales, "Enabling next-generation dark energy and epoch of reionization radio observatories with the moff correlator," *Publications of the Astronomical Society of the Pacific*, vol. 123, no. 909, p. 1265, 2011.

[1]https://github.com/epic-astronomy/LWA_EPIC/tree/test/4090/src

[6] N. Thyagarajan, A. P. Beardsley, J. D. Bowman, and M. F. Morales, "A generic and efficient e-field parallel imaging correlator for next-generation radio telescopes," *Monthly Notices of the Royal Astronomical Society*, vol. 467, no. 1, pp. 715–730, 2017.

[7] H. Krishnan, A. P. Beardsley, J. D. Bowman, J. Dowell, M. Kolopanis, G. Taylor, and N. Thyagarajan, "Optimization and commissioning of the EPIC commensal radio transient imager for the long wavelength array," *Monthly Notices of the Royal Astronomical Society*, vol. 520, no. 2, pp. 1928–1937, Apr. 2023.

[8] G. Taylor, S. Ellingson, N. Kassim, J. Craig, J. Dowell, C. Wolfe, J. Hartman, G. Bernardi, T. Clarke, A. Cohen *et al.*, "First light for the first station of the long wavelength array," *Journal of Astronomical Instrumentation*, vol. 1, no. 01, p. 1250004, 2012.