Achieving 1 TFLOP for the Radio Astronomy Correlator

Michael Clark[1], Paul La Plante[2], and Lincoln Greenhill[1]

1: Harvard-Smithsonian Center for Astrophysics, 2: Loyola University Maryland

Introduction

The Murchison Widefield Array (MWA), being built in Western Australia, represents the next-generation of radio telescopes. The MWA is composed of 512 antennas sampling over 768 frequency bands that capture raw data from the sky and create an image in real time through the use of interferometry. In order to build the power spectrum of the image, the signals from each antenna must be cross-correlated with every other, a process that scales as the square of the number of antenna. Following the correlator, the signals are sent to the imaging pipeline that creates the actual images of the sky. The current implementation of the correlator is deployed on field-programmable gate arrays (FPGAs) but the imaging pipeline is deployed using GPUs. The purpose of this investigation was to determine if the correlator algorithm for the MWA could be deployed on GPUs efficiently and effectively using the Fermi architecture.

Cross Correlation

The dominant part of the correlator algorithm, the so called X-engine, involves the sum over many vector outer products of an input vector and its conjugate transpose,

$$A(f) = \sum_t X_t^\dagger(f) \otimes X_t(f),$$

where $X_t(f)$ is the (complex-valued) signal vector at time t and frequency f and has dimensions $N=2n_s$ where n_s is the number of antenna and the factor of 2 comes from sampling X and Y polarizations. The resulting matrix A has dimensions NxN but is Hermitian, so only the lower or upper triangular part is required.

$$X \in [Time][Frequency][Antenna][Polarization]$$

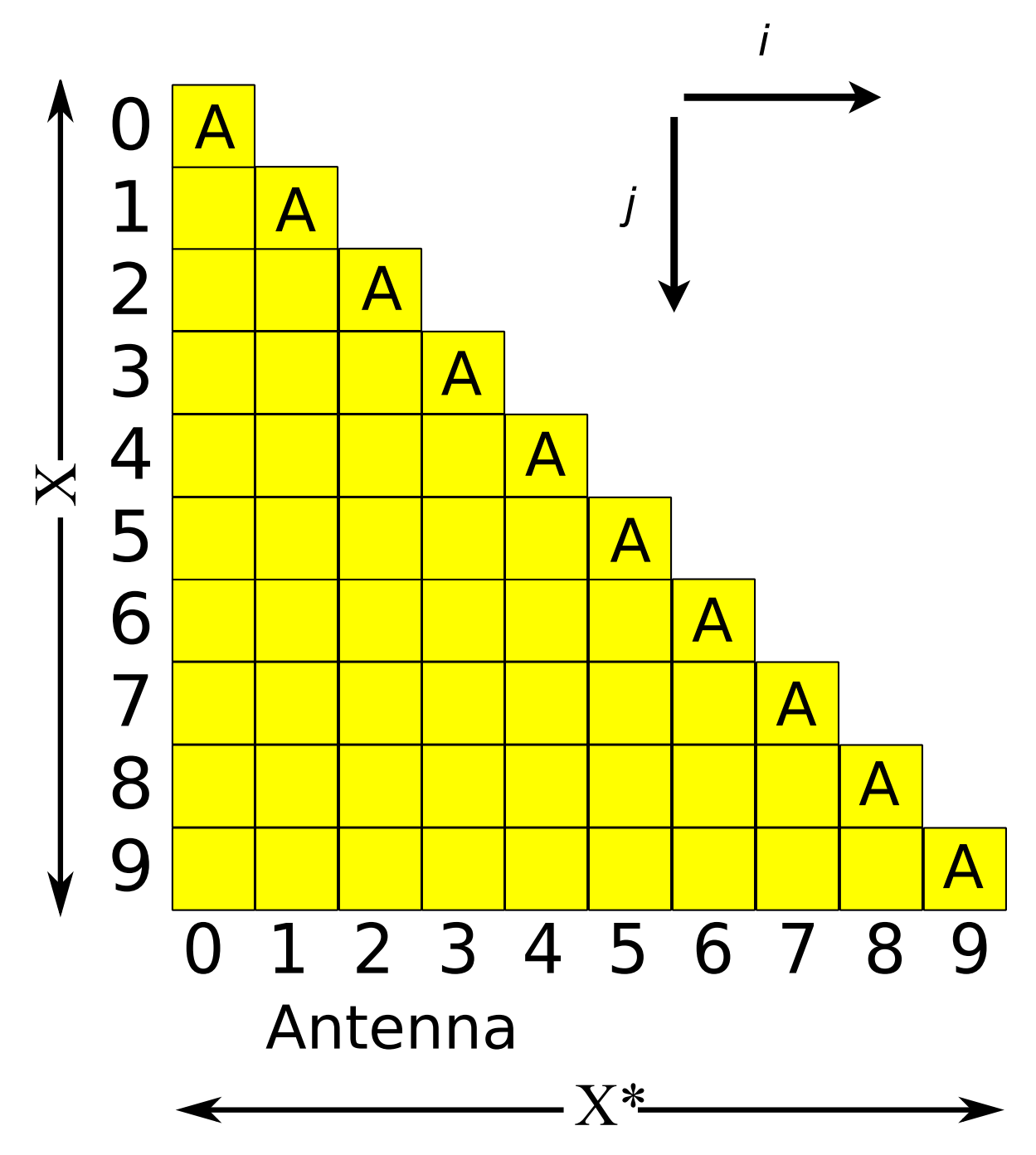
$$A \in [Frequency][Antenna][Antenna][Polarization][Polarization]$$

The correlation requires that all signals from all stations are cross-multiplied, and this is required independently for all frequencies, frequency can thus be trivially parallelized over. This leads to a computational cost that scales

$$\text{Cost} \sim F\left(\frac{N(N+1)}{2}\right),$$

where F is the number of frequencies. For the MWA, this requires ~128 TFLOPs of sustained computation. The future Square Kilometer Array (SKA) will feature $O(10{,}000)$ antenna, increasing cross-correlation to an Exaflop problem.

Right, pictorial representation of the correlator algorithm. Each antenna (station) includes 2 complex-valued polarization values. Since the two inputs of the outer product are the signal data and its conjugate transpose, the result is a Hermitian matrix, so only the lower triangle is necessary. The autocorrelations of a given station are shown along the diagonal. For a given antenna pair at a given frequency, the algorithm accumulates over all time steps.



Initial Strategy

Thread Indexing Each block of threads, of size *BxB* acts on a *BxB* block of the matrix, i.e., one thread per antenna pair. Since only the lower triangular half of the output matrix is required, the grid dimensions are set to

$$\text{grid} = \left(\frac{n_s}{2B}\left(\frac{n_s}{B}+1\right), F\right),$$

where the first grid dimension is used for triangular matrix index, and the second for the trivial frequency index. The global matrix coordinates *(i,j)* can then be obtained from

$$j = \left\lfloor -\frac{1}{2} + \sqrt{\frac{1}{4} + 2\,\text{blockIdx.x}} \right\rfloor + \text{threadIdx.x}$$

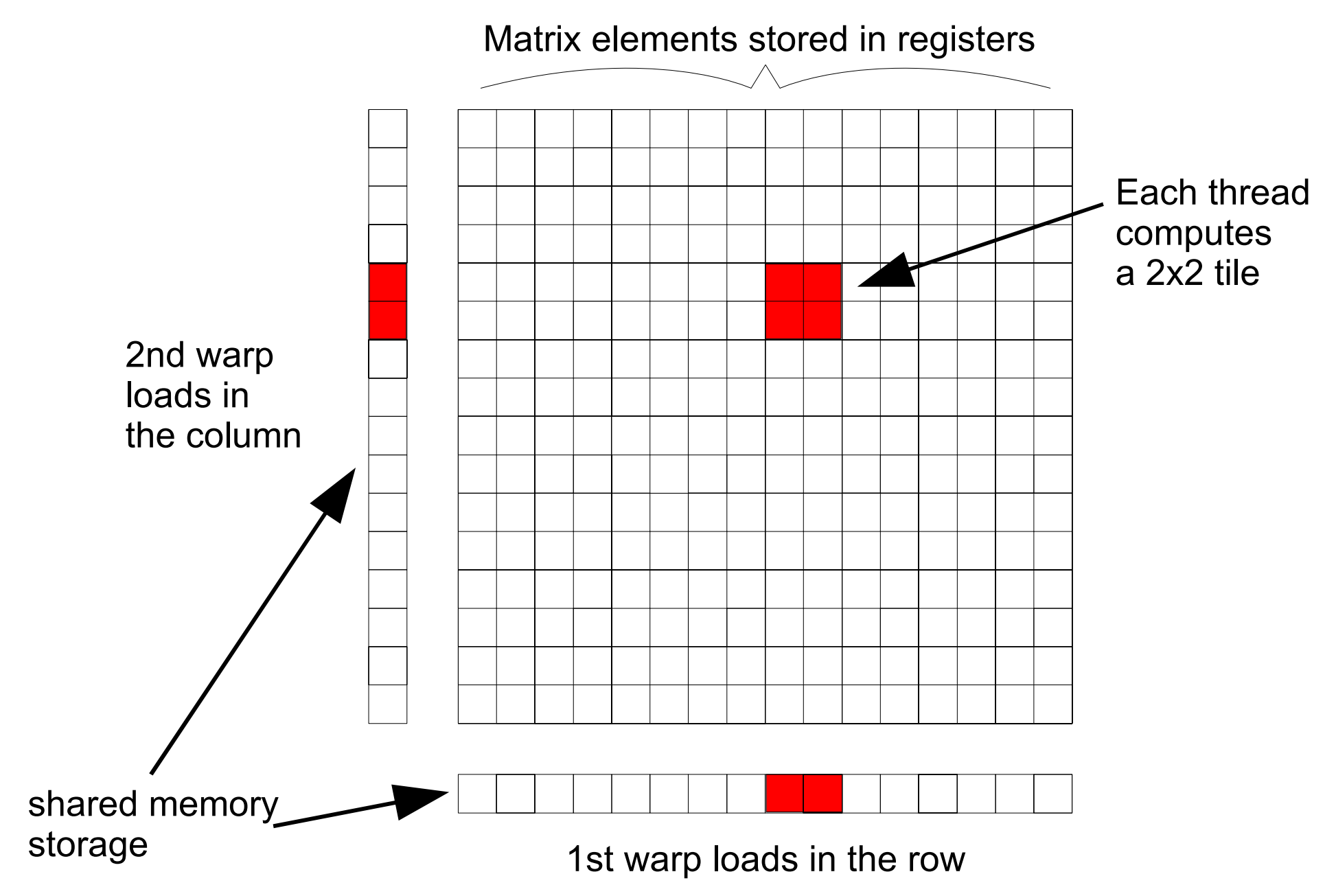
$$i = \text{blockIdx.x} - \frac{j(j+1)}{2} + \text{threadIdx.y}$$

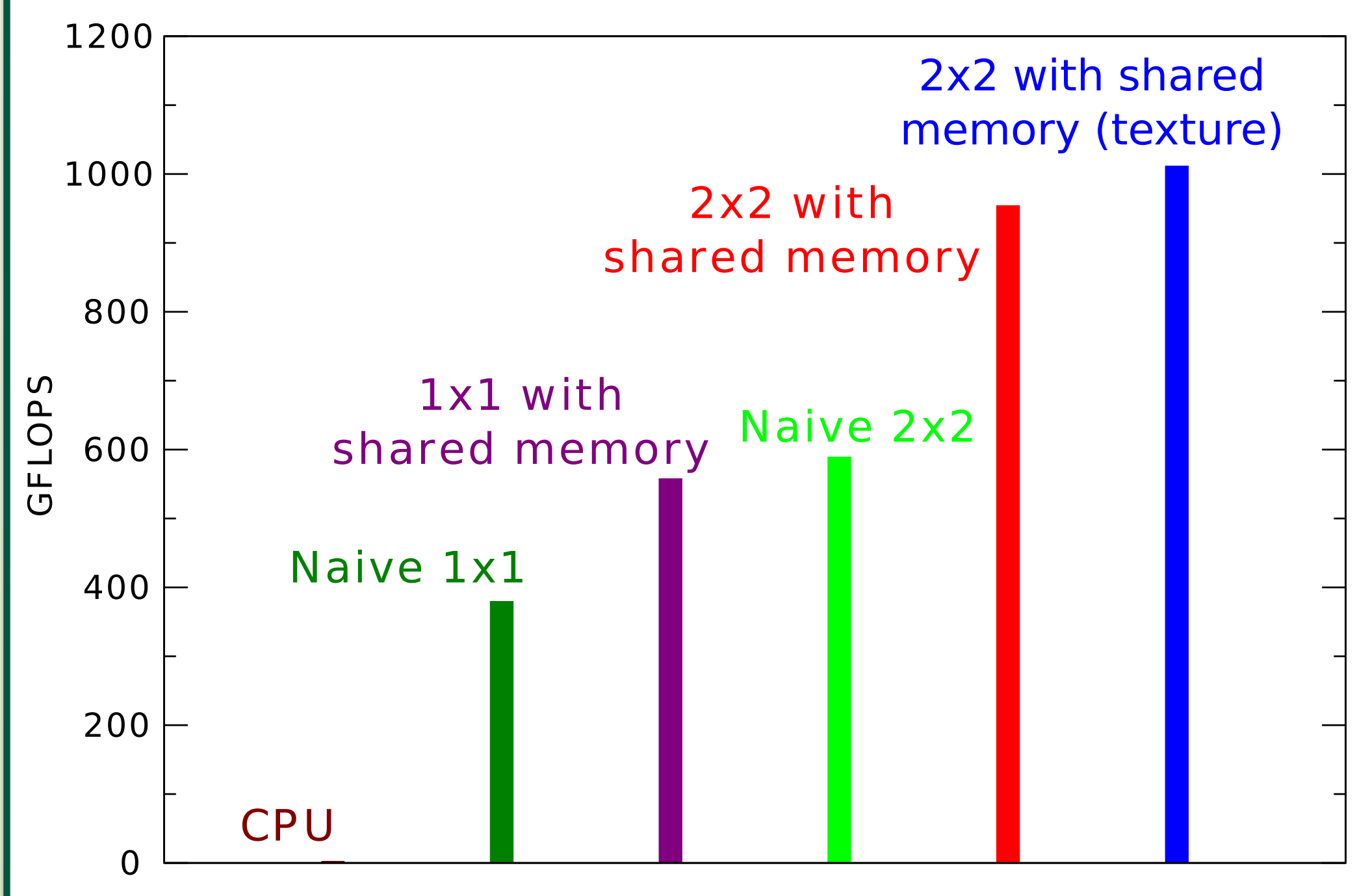
Naïve Implementation Each thread iterates through the time dimension, accumulating the result for a fixed antenna pair in registers and is responsible for loading all of the data required. This equates to two float4 loads: 1 row value and 1 column value, each consisting of 2 polarizations x 2 from complexity. The time average is over many thousands of samples, and so the cost of writing the matrix elements is negligible. The resulting arithmetic complexity of 1 flop per byte means the naïve implementation is bandwidth bound (which is around 8 on a GTX 480), though the presence of L1 and L2 cache does help, achieving 390 GFLOPS.

Tiling

Shared memory tiling To increase the arithmetic intensity, a shared memory tiling strategy was employed. With 64 threads per thread block, it is natural to use 8x8 tiling. This requires loading both a row and a column of 32 numbers (tile size x polarization x complexity). In this scheme, the first warp would load the row vector, and the second warp reads in the column vector. Each thread in a block only reads in a single float, and full coalescing is obtained. When combined with the fined-tuning optimizations this raised performance to 560 GFLOPS.

Register tiling Although shared memory has much faster bandwidth and lower latency than device memory, it is no substitute for registers. Using a 2x2 register tiling (the maximum achievable with 64 registers) increases the arithmetic intensity on the shared memory sourced loads to 4 which is enough to hide this bottleneck. When combining the tiling strategies, the shared memory tile size is quadrupled to 16x16, and each thread now loads a float2 from device memory. Performance is increased to 960 Gflops, and over the magical 1 TFLOP with the addition loading through the texture unit.


Matrix elements stored in registers
Each thread computes a 2x2 tile
2nd warp loads in the column
shared memory storage
1st warp loads in the row



Performance using a GeForce GTX 480 with N=1024 (n_s=512) and F=12. The 1x1 or 2x2 labels the register tile size. The naïve case represents all threads loading all data, the shared memory uses the shared memory for an extra level of tiling.

Fine-Tuning Optimizations

Pre-calculation of row/column offsets With shared memory tiling, it was necessary to differentiate warps threads for loading the row and column data. Since the same threads loaded data corresponding to the same row or column from time step to time step, we determined which thread loaded which data outside of the loop over time. Once inside the time loop, we simply iterated a pointer to coincide with the appropriate number for the next time step, removing the logical statements from the main loop body.

Shared Memory Buffering The performance gain from unrolling the time loop was limited by the necessary thread synchronizations. To reduce these we doubled the size of the shared memory per block combined with an unroll by of time loop by 2. By using different buffers for even and odd time, it made it possible to read from shared memory and to write to shared memory without thread synchronization in between. The result is a halving of the number of synchronization points, and a significant speedup.

Textures By binding the input array to a 2D texture, with dimensions *[Time]* x *[Frequency.Antenna.Polarization]*, the texture unit performs the global memory indexing for free. This brought an additional advantage of reducing register pressure which had increased from the addition of time loop unrolling.

Results and Conclusions

In this work we have demonstrated that cross-correlation can be very efficiently performed using the Fermi architecture. Key to achieving this high performance was a multi-level tiling strategy which required using shared memory tiling to minimize global memory accesses, and register tiling to hide shared memory accesses. We were able to achieve 78% of peak performance, which on the GeForce GTX 480 corresponds to 1040 GFLOPS. To our knowledge, this is the first time that a non-trivial application has achieved this performance. With a sustained performance of 1 TFLOP, it would require 128 GPUs to replace the current FPGA correlator for the MWA.

References

1. R. G. Edgar, *et al.* “Enabling a High Throughput Real Time Data Pipeline for a Large Radio Telescope with GPUs.” *Computer Physics Communications* (to appear), 2010.
2. C. J. Lonsdale, *et al.* “The Murchison Widefield Array: Design Overview.” *Proceedings of the Ieee*, 97(8):1497-1506, August 2009.