Using GPUs for Signal Correlation

Michael Clark
with
Paul La Plante and Lincoln Greenhill
Outline

• Motivation
• Fermi architecture
• Mapping the X-engine onto a GPU
• Comparison
• Summary
Motivation

- GPUs are an interesting platform
  - Harness commodity hardware
  - Low cost
  - Easy to develop
  - Forwards compatible
- How competitive are GPUs?
  - Vs. other commodity hardware
  - Vs. FPGAs
- Testcase: MWA
However, the MWA's location also brings some logistical challenges. Since the MWA is located in the desert, there is no power station nearby to provide energy for the computational backend of the telescope. Additionally, there is no fiber network to transmit the data off-site once it has been collected. Furthermore, the MWA is projected to generate $\sim 30$ kW per week, which makes storage of the raw images impossible. Thus, the MWA must process the raw data in real-time, then discard them. The current plans call for the MWA to collect data in a second interval, so that the images taken of the sky remain relevant.

2.3 Computational processes of the MWA

The computational backend of the MWA contains many different computational tasks that must be completed in a very short time span. The initial stage of the algorithm is a Fourier transform that converts the signals from real space into frequency space. The resulting images are then passed to the correlator, which cross correlates the signals in order to filter out background noise and determine the location in the sky of the stellar object being observed. Together, these processes are referred to as the “FX engine,” since it includes an “F” for a Fourier transform and an “X” for the correlator engine. Following the FX engine, the signal is passed to the imaging pipeline, which includes various geometrical transformations and other algorithms in order to complete the image of the sky. However, the precise algorithms of the imaging portion of the backend are very complex, and beyond the scope of this paper. Additionally, the implementation of the Fourier transform has been well-studied and well-documented, and is not unique to this application. Thus, this work will focus primarily on the correlator stage of the computational backend.

The current implementation of the correlator employs graphics processing units (GPUs) in order to perform the calculations, while the FX engine is deployed on specialty hardware known as field-programmable gate arrays (FPGAs). Currently, FPGAs are able to perform the calculations of the FX engine more quickly than GPUs. However, as the size of the array grows large, so does the amount of traffic between the FX engine and the later stages of the imaging pipeline. Thus, the purpose of this work is to examine how to deploy the correlator algorithm on a GPU efficiently so that in future implementations there will be less of a bottleneck between the FX engine and the imaging pipeline. This work focused on developing an algorithm that would complete the work quickly but also energy efficiently, since efficiency is another priority of the MWA implementation.

**Figure 1:** The imaging pipeline for the MWA. The correlator (specifically the cross-multiply and integrate) serves as the focus for this work.

---

### 3 Graphical Processing Units

Recent advancements in the development of GPUs have made them suitable for general-purpose scientific computing as a way to perform massively parallel calculations. In the past, programming of GPUs had to be done by using a graphics application programming interface (API), which proved to be alien to the GPUs. Therefore, the implementation of the Fourier transform has been well-studied and well-documented, and is not unique to this application. Thus, this work will focus primarily on the correlator stage of the computational backend.

The current implementation of the correlator employs graphics processing units (GPUs) in order to perform the calculations, while the FX engine is deployed on specialty hardware known as field-programmable gate arrays (FPGAs). Currently, FPGAs are able to perform the calculations of the FX engine more quickly than GPUs. However, as the size of the array grows large, so does the amount of traffic between the FX engine and the later stages of the imaging pipeline. Thus, the purpose of this work is to examine how to deploy the correlator algorithm on a GPU efficiently so that in future implementations there will be less of a bottleneck between the FX engine and the imaging pipeline. This work focused on developing an algorithm that would complete the work quickly but also energy efficiently, since efficiency is another priority of the MWA implementation.
MWA Correlator

- 512 antenna x 2 polarizations = 1024 inputs
- total bandwidth = 31 MHz
- X-engine requires
  - $1.6 \times 10^{13}$ CMACs = 128 TFLOPs
- How many GPUs required?
Fermi Architecture

GeForce GTX 480

- 480 processing cores
- 1.345 Tflops SP
- 177 GB/s memory bandwidth
- Power consumption 250 Watts
- Easy to program
- $450 (as of yesterday)
Fermi Architecture

- High memory bandwidth
- BUT high flop:byte ratio 7.6:1
- Need high arithmetic intensity to keep GPU busy
- Use memory hierarchy
  - Shared memory
  - Registers
Mapping the X-engine onto a GPU

- Assume corner turn already done
- 525K matrix elements required / frequency
- With 12 channels => 1M threads
- Use FP32 throughout
- Launch a triangular grid of threads blocks
  - Stored as a 1-d index
  - 2-d index

\[
\begin{align*}
  j &= \left\lceil -\frac{1}{2} + \sqrt{\frac{1}{4} + 2 \cdot \text{blockIdx.x}} \right\rceil + \text{threadIdx.x} \\
  i &= \text{blockIdx.x} - \frac{j(j + 1)}{2} + \text{threadIdx.y}
\end{align*}
\]
Triangular grid of thread blocks (with $N_c$ copies)
Thread block

Matrix elements stored in registers

Each thread computes a 1x1 tile

flop/byte
Algorithm: 1
Hardware: 7.6
Each thread block

Matrix elements stored in registers

2nd warp loads in the column

shared memory storage

1st warp loads in the row

Each thread computes a 1x1 tile

flop/byte
Algorithm: 16
Hardware: 7.6

flop/byte
Algorithm: 1
Hardware: 1.5
Each thread block

flop/byte
Algorithm: 16
Hardware: 7.6

2nd warp loads in the column

shared memory storage

Matrix elements stored in registers

Each thread computes a 2x2 tile

flop/byte ratio
Algorithm: 2:1
Hardware: 1.5:1

1st warp loads in the row
90% development time

- 2x2 with shared memory (texture)
- 2x2 with shared memory
- 1x1 with shared memory
- Naive 2x2
- Naive 1x1
- CPU

GFLOPS
Multi-GPU

- Trivial to parallelise across frequency
  - Scales perfectly
- PCIe bus contention not a problem
- 4 TFLOPs sustained with 4 GPUs
  - Equivalent to 60 MHz with 64 inputs
  - (cf DW - 1.3 GPU, 2 CPU, 10.4 FPGA)
- 1300 Watts total
- $6K for workstation => $1.50 per GFLOP
# X-engine Performance Across Platforms

<table>
<thead>
<tr>
<th>Architecture</th>
<th>GFLOPS</th>
<th>GOPS/Watt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Core i7*</td>
<td>48.0</td>
<td>0.4</td>
</tr>
<tr>
<td>IBM BG/P*</td>
<td>13.1</td>
<td>0.5</td>
</tr>
<tr>
<td>IBM Cell*</td>
<td>187</td>
<td>2.7</td>
</tr>
<tr>
<td>Nvidia GT200*</td>
<td>243</td>
<td>1</td>
</tr>
<tr>
<td>Nvidia Fermi†</td>
<td>1010</td>
<td>~4 (3)</td>
</tr>
</tbody>
</table>

* van Nieuwpoort and Romein, †this work, ‡de Souza et al and Lonsdale et al, ‡‡Manley
## X-engine Performance Across Platforms

<table>
<thead>
<tr>
<th>Architecture</th>
<th>GFLOPS</th>
<th>GOPS/Watt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Core i7*</td>
<td>48.0</td>
<td>0.4</td>
</tr>
<tr>
<td>IBM BG/P*</td>
<td>13.1</td>
<td>0.5</td>
</tr>
<tr>
<td>IBM Cell*</td>
<td>187</td>
<td>2.7</td>
</tr>
<tr>
<td>Nvidia GT200*</td>
<td>243</td>
<td>1</td>
</tr>
<tr>
<td>Nvidia Fermi†</td>
<td>1010</td>
<td>~4 (3)</td>
</tr>
<tr>
<td>MWA Correlator #</td>
<td></td>
<td>(~10)</td>
</tr>
<tr>
<td>(Virtex 4 SX35)</td>
<td></td>
<td>(~58)</td>
</tr>
<tr>
<td>Roach II+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Virtex 6)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* van Nieuwpoort and Romein, †this work, ‡de Souza et al and Lonsdale et al, †Manley
However, the MWA's location also brings some logistical challenges. Since the MWA is located in the desert, there is no power station nearby to provide energy for the computational backend of the telescope. Additionally, there is no fiber network to transmit the data off-site once it has been collected. Furthermore, the MWA is projected to generate $\sim 30$ times bandwidth which makes storage of the raw images impossible. Thus, the MWA must process the raw data in real-time, then discard them.

### 2.3 Computational processes of the MWA

The computational backend of the MWA contains many different computational tasks that must be completed in a very short time span. The initial stage of the algorithm is a Fourier transform that converts the signals from real space into frequency space. The resulting images are then passed to the correlator, which cross-correlated the signals in order to filter out background noise and determine the location in the sky of the stellar object being observed. Together, these processes are referred to as the "FX engine" since it includes an "F" for a Fourier transform and an "X" for the correlator engine.

Following the FX engine, the signal is passed to the imaging pipeline, which includes various geometrical transformations and other algorithms in order to complete the image of the sky. However, the precise algorithms of the imaging portion of the backend are very complex and beyond the scope of this paper. Additionally, the implementation of the Fourier transform has been well-studied and well-documented and is not unique to this application. Thus, this work will focus primarily on the correlator stage of the computational backend.

The current implementation of the correlator employs graphics processing units (GPUs) in order to perform the calculations, while the FX engine is deployed on specialty hardware known as field-programmable gate arrays (FPGAs). Currently, FPGAs are able to perform the calculations of the FX engine more quickly than GPUs. However, as the size of the array grows larger, so does the amount of traffic between the FX engine and the later stages of the imaging pipeline. Thus, the purpose of this work is to examine how to deploy the correlator algorithm on a GPU efficiently so that in future implementations there will be less of a bottleneck between the FX engine and the imaging pipeline.

This work focused on developing an algorithm that would complete the work quickly but also energy efficient since efficiency is another priority of the MWA implementation.

### Figure 3:

The imaging pipeline for the MWA. The correlator (specifically the cross-multiply and integrate) serves as the focus for this work.

### 3 Graphical Processing Units

Recent advancements in the development of GPUs have made them suitable for general-purpose scientific computing as a way to perform massively parallel calculations. In the past, programming of GPUs had to be done by using a graphics application programming interface (API) alien to the GPUs, which proved difficult. Now, GPUs can be programmed like any other general-purpose processor.

- **FPGA F-engine (4 kW)**
- **128x GPU 42 kW (12 kW)**
- **64x GPU (30 kW)**
FPGA F-engine (4 kW)

42 kW
128xGPU
Solves a large $n$ bottleneck?

O($n$) data

O($n^2$) data kept in GPU memory
It's not all Nvidia

- AMD / ATI
- > 2x higher peak
- more power efficient

Next step
- Expect parity with MWA correlator
- Incompatible with current RTS

GPU Computing Efficiency Trend

Thread Processors
- 5-way VLIW Architecture
- 4 Stream Cores and 1 Special Function Stream Core
- Separate Branch Unit
- All 5 cores co-issue
- Scheduling across the cores is done by the compiler
- Each core delivers a 32-bit result per clock
- Thread Processor writes 5 results per clock

SIMD Engines
- Diagram shows 2 SIMD Engines
- Each SIMD Unit includes:
  - 16 Thread Processors (80 shader cores) + 32KB Local Data Share
  - Its own Thread Sequencer which operates a shared set of threads
  - A dedicated fetch unit with an 8KB L1 cache

ATI Radeon™ HD 5870 Compute Architecture
- 20 SIMD Engines
- 1600 shader cores
- Ultra-Threaded Dispatch Processor
- Instruction and Constant Caches
- Memory Export Buffer
- Fetch path with multi-level caches
- Global Data Store
It’s not all Nvidia

- AMD / ATI
  - > 2x higher peak
  - more power efficient
- Next step
- Expect parity with MWA correlator
- Incompatible with current RTS
Summary

• X-engine is a perfect match to the GPU
  • 78% peak performance
• Low cost and development time
  • Easy to keep with bleeding edge
• Not power competitive with FPGAs
• Future: combine X-engine with Calibration and Imaging?
  • Increases power efficiency
  • No $O(n^2)$ data transfer