GPUs and MeerKAT
Bruce Merry

Outline

1 Background
   - Radio Astronomy
   - MeerKAT

2 Median Filter
   - Serial
   - Parallel
   - Tuning

3 Transposition
   - Introduction
   - Shared Memory
Outline

1 Background
   - Radio Astronomy
   - MeerKAT

2 Median Filter
   - Serial
   - Parallel
   - Tuning

3 Transposition
   - Introduction
   - Shared Memory
Why Radio?

Gamma rays, X-rays and ultraviolet light blocked by the upper atmosphere (best observed from space).

Visible light observable from Earth, with some atmospheric distortion.

Most of the infrared spectrum absorbed by atmospheric gasses (best observed from space).

Radio waves observable from Earth.

Long-wavelength radio waves blocked.
Another View

http://xkcd.com/273
1 Jy = 1 \times 10^{-26} \text{ W m}^{-2} \text{ Hz}^{-1}

- **Sun**: $10^5$ Jy (at 5 GHz)
- **Cellphone**: $10^8$ Jy (1km away)
- **Source**: 1–10 $\mu$Jy
Interferometry

\[ V(u, v, w) = \int_{l, m} A(l, m, w) I(l, m) e^{-2\pi i [ul + vm]} \, dl \, dm \]
1 Background
   - Radio Astronomy
   - MeerKAT

2 Median Filter
   - Serial
   - Parallel
   - Tuning

3 Transposition
   - Introduction
   - Shared Memory
Prototype Imaging Node
This is Cool
Ingest

Ingest - Wideband - Node 1/4 - Narrowband - Node 1/5

CPU

GPU

Input Conditioning

SPREAD Decode / Host Buffer

(2 x 32 bit int)

Output Conditioning

Output Host Buffer and TM merge
Spectral Data / Continuum Data
Signal Display Data

2 Hz

convert & scale

Calculate precision

correct vis

0.5 Hz

time avg

2 Hz -> 0.5 Hz

(0.1 - 2 Hz)

flagging

masks (prior / operator)

flags (CAM / CBF)
detected RFI

predicted RFI

continuum avg

8192 -> 512

(Wb only)

time avg

2 Hz -> 0.5 Hz

(0.1 - 0.5 Hz)

continuum avg

8192 -> 256

4096 -> 128

masked avg time series

0.5 Hz

calculate percentiles

SDP

Core Switch

L0

Vis

SDP sd control

TM telescope state info

telescope state info

CBF Core Switch

40 GbE AO (30m)

2 Hz

Ingest - Wideband - Node 1/4 - Narrowband - Node 1/5

GPU

CPU

0.5 Hz

continuum avg

8192 -> 512

(Wb only)

calculate percentiles

masked avg time series

0.5 Hz

calculate percentiles

continuum avg

8192 -> 256

4096 -> 128

continuum avg

8192 -> 512

(Wb only)
Ingest is responsible for

- Converting from integer to float
- Correcting some instrumental effects
- Flagging RFI
- Reducing data in time
- Reducing data in frequency
- Computing signal statistics
Real-time Signal Displays

Spectrum at Mon Nov 10 10:37:09 2014

- auto
- 6v6v
- 5v5v

1650 1700 1750 1800 1850 1900 1950 MHz

-20 -10 0 10 20 dB
Lots of Data

- 2 frames per second
- 2080 baselines (64 antennas)
- 53248 frequency channels \((32k + 5 \times 4k)\)
- 4 polarisations
- 8 bytes per visibility \((2 \times \text{int32})\)
Lots of Data

- 2 frames per second
- 2080 baselines (64 antennas)
- 53248 frequency channels \((32k + 5 \times 4k)\)
- 4 polarisations
- 8 bytes per visibility \((2 \times \text{int32})\)

6.6 GiB per second input
Lots of Data

2 frames per second
2080 baselines (64 antennas)
53248 frequency channels \((32k + 5 \times 4k)\)
4 polarisations
8 bytes per visibility \((2 \times \text{int32})\)

6.6 GiB per second input
1.0 GiB per second per server
“All problems in computer science can be solved by another level of indirection” — *David Wheeler*
Kernel Compilation

mako → mako → CUDA C → PyCUDA → cubin → CUDA
Example Template

```c
1 KERNEL REQD_WORK_GROUP_SIZE(${block}, ${block}, 1)
2 void accum(
3    % for i in range(outputs):
4        GLOBAL float2 * RESTRICT out_vis${i},
5        GLOBAL float * RESTRICT out_weights${i},
6        GLOBAL unsigned char * RESTRICT out_flags${i},
7    % endfor
8        const GLOBAL float2 * RESTRICT in_vis,
9        const GLOBAL float * RESTRICT in_weights,
10        const GLOBAL unsigned char * RESTRICT in_flags,
11        int out_stride,
12        int in_full_stride,
13        int in_kept_stride,
14        int channel_start)
15 {
16    ...
17 }
```
Example Template

```c
__global__ __launch_bounds__((16)*(16)*(1))
void accum(
    float2 * __restrict out_vis0,
    float * __restrict out_weights0,
    unsigned char * __restrict out_flags0,
    const float2 * __restrict in_vis,
    const float * __restrict in_weights,
    const unsigned char * __restrict in_flags,
    int out_stride,
    int in_full_stride,
    int in_kept_stride,
    int channel_start)
{
...
}
```
Example Template

```c
__kernel __attribute__((reqd_work_group_size(16,16,1)))
void accum(
    __global float2 * restrict out_vis0,
    __global float * restrict out_weights0,
    __global unsigned char * restrict out_flags0,
    const __global float2 * restrict in_vis,
    const __global float * restrict in_weights,
    const __global unsigned char * restrict in_flags,
    int out_stride,
    int in_full_stride,
    int in_kept_stride,
    int channel_start)
{
    ...
}
```
Outline

1. Background
   - Radio Astronomy
   - MeerKAT

2. Median Filter
   - Serial
   - Parallel
   - Tuning

3. Transposition
   - Introduction
   - Shared Memory
Definition
Definition
Sliding Window Algorithm

<table>
<thead>
<tr>
<th>10</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>22</td>
<td>3</td>
</tr>
<tr>
<td>34</td>
<td>4</td>
</tr>
</tbody>
</table>

14
Sliding Window Algorithm
Sliding Window Algorithm
1 Background
   - Radio Astronomy
   - MeerKAT

2 Median Filter
   - Serial
   - Parallel
     - Tuning

3 Transposition
   - Introduction
   - Shared Memory
Parallelisation

![Diagram showing channel and baseline with arrows moving from channel to baseline.](image-url)
More Parallelism
Sample code

```c
__device__ void median_filter_slide(
    median_filter *self, float new_sample) {
{
    float old_sample = self->samples[0];
    int new_rank = WIDTH - 1;
    #pragma unroll
    for (int j = 0; j < WIDTH - 1; j++) {
        self->samples[j] = self->samples[j + 1];
        int old_cmp = (old_sample <= self->samples[j]);
        int cmp = (new_sample < self->samples[j]);
        new_rank -= cmp;
    }
    self->samples[WIDTH - 1] = new_sample;
    self->rank[WIDTH - 1] = new_rank;
}
```
Memory access pattern
Memory access pattern
1. Background
   - Radio Astronomy
   - MeerKAT

2. Median Filter
   - Serial
   - Parallel
   - Tuning

3. Transposition
   - Introduction
   - Shared Memory
Block size

![Graph showing rate vs. block size for GTX 980 and K40 GPUs.]

- **GTX 980**
- **K40**

- **Rate (samples per second)**
  - Y-axis range: 0 to 120
  - Y-axis label: $10^9$

- **Block size**
  - X-axis range: 0 to 1,024

- **Occupancy (%)**
  - Y-axis range: 0 to 100

- **GTX 980**
  - Blue dots
  - Line style: solid

- **K40**
  - Red squares
  - Line style: dashed
Autotuning

Parameter space → Attributes → Tuner → Result cache

Kernel

Parameter space

Attributes

Benchmark

Tuner

Result cache
Python Is Easy

Autotuning for a fill kernel

```python
1 @classmethod
2 @tune.autotuner(test={'wgs': 128})
3 def autotune(cls, context, dtype, ctype):
4     queue = context.create_tuning_command_queue()
5     shape = (1048576,)
6     data = accel.DeviceArray(context, shape, dtype)
7     def generate(**tuning):
8         fn = cls(
9             context, dtype, ctype,
10            **tuning).instantiate(queue, shape)
11         fn.bind(data=data)
12         return tune.make_measure(queue, fn)
13     return tune.autotune(
14         generate, wgs=[64, 128, 256, 512])
```
Outline

1. Background
   - Radio Astronomy
   - MeerKAT

2. Median Filter
   - Serial
   - Parallel
   - Tuning

3. Transposition
   - Introduction
   - Shared Memory
Motivation

What if the incoming data looks like this?
Motivation

What if the incoming data looks like this?
Motivation

What if the incoming data looks like this?
Transposing the matrix:

\[
\begin{bmatrix}
\cos 90^\circ & \sin 90^\circ \\
-\sin 90^\circ & \cos 90^\circ
\end{bmatrix}
\begin{bmatrix}
a_1 \\ a_2
\end{bmatrix}
= \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}
\]

http://xkcd.com/184/
Naive Kernel

```c
__global__ void transpose(
  float *out, const float *in, int R, int C)
{
  int x = blockIdx.x * blockDim.x + threadIdx.x;
  int y = blockIdx.y * blockDim.y + threadIdx.y;
  float value = in[y * C + x];
  out[x * R + y] = value;
}
```
Naive Kernel Results

<table>
<thead>
<tr>
<th></th>
<th>L2 Cache</th>
<th>Unified Cache</th>
<th>Device Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Reads: 8388679</td>
<td>Local Loads: 16777216</td>
<td>Reads: 8388632</td>
</tr>
<tr>
<td></td>
<td>Writes: 67108870</td>
<td>Local Stores: 0</td>
<td>Writes: 8390508</td>
</tr>
<tr>
<td></td>
<td>Total: 75497549</td>
<td>Global Loads: 6177468</td>
<td>Total: 16779140</td>
</tr>
<tr>
<td></td>
<td>30.937 GB/s</td>
<td>Global Stores: 247.496 GB/s</td>
<td>30.937 GB/s</td>
</tr>
<tr>
<td></td>
<td>247.496 GB/s</td>
<td>Texture Reads: 8388608</td>
<td>30.944 GB/s</td>
</tr>
<tr>
<td></td>
<td>278.433 GB/s</td>
<td>Unified Total: 9227468</td>
<td>61.881 GB/s</td>
</tr>
</tbody>
</table>

Only 62 of 224 GB/s (GTX 980)
Naive Kernel Access Pattern

Input

Output
Naive Kernel Access Pattern
Background
- Radio Astronomy
- MeerKAT

Median Filter
- Serial
- Parallel
- Tuning

Transposition
- Introduction
- Shared Memory
Using Shared Memory

Input

Shared

Output
Using Shared Memory

- **Input**
- **Shared**
- **Output**

**Shared Memory**

- Using Shared Memory
- Input
- Shared
- Output
Using Shared Memory
Using Shared Memory

Input

Shared

Output
Using Shared Memory

Input

Shared

Output
Using Shared Memory

Input

Shared

Output
Using Shared Memory
Using Shared Memory
1 __global__ void transpose(
2 float *out, const float *in, int R, int C)
3 {
4     __shared__ float tile[TILE][TILE];
5     int x = blockIdx.x * blockDim.x + threadIdx.x;
6     int y = blockIdx.y * blockDim.y + threadIdx.y;
7     tile[threadIdx.y][threadIdx.x] = in[y * C + x];
8     __syncthreads();
9     int x2 = blockIdx.x * blockDim.x + threadIdx.y;
10    int y2 = blockIdx.y * blockDim.y + threadIdx.x;
11    out[x2 * R + y2] = tile[threadIdx.x][threadIdx.y];
12 }
Shared Memory Version 1

```c
#include <stdio.h>

__global__ void transpose(
    float *out, const float *in, int R, int C)
{
    __shared__ float tile[TILE][TILE];
    int x = blockIdx.x * blockDim.x + threadIdx.x;
    int y = blockIdx.y * blockDim.y + threadIdx.y;
    tile[threadIdx.y][threadIdx.x] = in[y * C + x];
    __syncthreads();
    int x2 = blockIdx.x * blockDim.x + threadIdx.y;
    int y2 = blockIdx.y * blockDim.y + threadIdx.x;
    out[x2 * R + y2] = tile[threadIdx.x][threadIdx.y];
}
```

102 GB/s (GTX 980)
Bank Conflicts

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Shared
Bank Conflicts

Shared Memory

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
Padding
### Padding

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

**Shared**
Padding

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Shared
__global__ void transpose(
    float *out, const float *in, int R, int C)
{
    __shared__ float tile[TILE][TILE + 1];
    int x = blockIdx.x * blockDim.x + threadIdx.x;
    int y = blockIdx.y * blockDim.y + threadIdx.y;
    tile[threadIdx.y][threadIdx.x] = in[y * C + x];
    __syncthreads();
    int x2 = blockIdx.x * blockDim.x + threadIdx.y;
    int y2 = blockIdx.y * blockDim.y + threadIdx.x;
    out[x2 * R + y2] = tile[threadIdx.x][threadIdx.y];
}
Shared Memory Version 2

```c
__global__ void transpose(
    float *out, const float *in, int R, int C)
{
    __shared__ float tile[TILE][TILE + 1];
    int x = blockIdx.x * blockDim.x + threadIdx.x;
    int y = blockIdx.y * blockDim.y + threadIdx.y;
    tile[threadIdx.y][threadIdx.x] = in[y * C + x];
    __syncthreads();
    int x2 = blockIdx.x * blockDim.x + threadIdx.y;
    int y2 = blockIdx.y * blockDim.y + threadIdx.x;
    out[x2 * R + y2] = tile[threadIdx.x][threadIdx.y];
}
```

144 GB/s (GTX 980)
Thread Coarsening

Instruction breakdown

1. Load from Global
2. Store to Shared
3. Load from Shared
4. Store to Global
5. Barrier (\texttt{__syncthreads})
Thread Coarsening

Instruction breakdown

1 Load from Global
1 Store to Shared
1 Load from Shared
1 Store to Global
1 Barrier (__syncthreads)
32 Other
What if we apply *less* parallelism?

```
Input  

Shared  

Output  
```
Thread Coarsening

What if we apply *less* parallelism?

Input

Shared

Output
What if we apply less parallelism?
Thread Coarsening

What if we apply *less* parallelism?

Input

Shared

Output
Thread Coarsening

What if we apply *less* parallelism?

Input

Shared

Output
Thread Coarsening

What if we apply less parallelism?

Input

Shared

Output
What if we apply *less* parallelism?

![Diagram showing input, shared, and output grids]

Input

Shared

Output
Thread Coarsening

What if we apply less parallelism?

Input

Shared

Output
What if we apply *less* parallelism?
Thread Coarsening Code

1 __shared__ float tile[SCALE * TILE * (TILE + 1)];
2 int x0 = blockIdx.x * (SCALEX * TILE);
3 int y0 = blockIdx.y * (SCALEY * TILE);
4 int addr0 = (y0+threadIdx.y) * C + x0+threadIdx.x;
5 int tile0 = threadIdx.y * (TILE + 1) + threadIdx.x;
6 for (int i = 0; i < SCALE; i++)
7    tile[tile0+ofs.in_tile[i]] = in[addr0+ofs.in[i]];
8 __syncthreads();
9 addr0 = (x0 + threadIdx.y) * R + (y0 + threadIdx.x);
10 tile0 = threadIdx.x * (TILE + 1) + threadIdx.y;
11 for (int i = 0; i < SCALE; i++)
12    out[addr0+ofs.out[i]] = tile[tile0+ofs.out_tile[i]];
Thread Coarsening

New instruction breakdown

4 Load from Global
4 Store to Shared
4 Load from Shared
4 Store to Global
1 Barrier (\texttt{__syncthreads})
79 Other
Thread Coarsening

New instruction breakdown

- 4 Load from Global
- 4 Store to Shared
- 4 Load from Shared
- 4 Store to Global
- 1 Barrier (__syncthreads)
- 79 Other

154 of 224 GB/s (GTX 980)
Kernel Fusion

Convert

Transpose

Global Memory

- Kernel Fusion
- Convert
- Transpose
- Global Memory

Shared Memory
Kernel Fusion

Convert

Transpose

Global Memory
Summary
Think about the memory system.