Towards a Performance-Portable FFT Library for Heterogeneous Computing

Carlo C. del Mundo*, Wu-chun Feng*§
*Dept. of ECE, §Dept. of CS

Virginia Tech
Forecast (Problem)

AMD Radeon HD 6970 (VLIW)

Follow along at: goo.gl/1fs9G7
Forecast (Problem)

AMD Radeon HD 6970 (VLIW)  Performance-Portable?  NVIDIA Kepler K20c (Non-VLIW)

Follow along at: goo.gl/1fs9G7
Forecast (Problem)

• How to find portable set of optimizations for GPUs?

AMD Radeon HD 6970 (VLIW)  
NVIDIA Kepler K20c (Non-VLIW)

Follow along at: goo.gl/1fs9G7
“Too much heterogeneity within GPUs”
“Too much heterogeneity within GPUs”
“Too much heterogeneity within GPUs”
“Too much heterogeneity within GPUs”

Follow along at: goo.gl/1fs9G7
“Too much heterogeneity within GPUs”

AMD Radeon HD 6970 (VLIW)

NVIDIA C2075 (Non-VLIW)

Follow along at: goo.gl/1fs9G7
“Too much heterogeneity within GPUs”

- AMD Radeon HD 6970 (VLIW)
- NVIDIA C2075 (Non-VLIW)
- AMD Radeon HD 7970 (non-VLIW)

Follow along at: goo.gl/1fs9G7
“Too much heterogeneity within GPUs”
“Too much heterogeneity within GPUs”
“Too much heterogeneity within GPUs”

• Problem
“Too much heterogeneity within GPUs”

• Problem
  – How do we ...
“Too much heterogeneity within GPUs”

• Problem
  – How do we ...
    • simultaneously optimize for all GPUs?
“Too much heterogeneity within GPUs”

• **Problem**
  – How do we ...
    • simultaneously optimize for all GPUs?
    • provide insight on machine-level behavior?
“Too much heterogeneity within GPUs”

• **Problem**
  - How do we ...
    - simultaneously optimize for all GPUs?
    - provide insight on machine-level behavior?

• **Solution (Contribution)**
“Too much heterogeneity within GPUs”

• **Problem**
  – How do we …
    - simultaneously optimize for all GPUs?
    - provide insight on machine-level behavior?

• **Solution (Contribution)**
  – A *methodology* for determining portable optimizations for a class of algorithms on GPUs
“Too much heterogeneity within GPUs”

• **Problem**
  – How do we ...
    • simultaneously optimize for all GPUs?
    • provide insight on machine-level behavior?

• **Solution (Contribution)**
  – A *methodology* for determining portable optimizations for a class of algorithms on GPUs
    • FFTs used as a case study
FFT: a building block across disciplines

Follow along at: goo.gl/1fs9G7
FFT: a building block across disciplines

http://www.ajnr.org/content/27/6/1230/F1.large.jpg

Follow along at: goo.gl/1fs9G7
FFT: a building block across disciplines

http://www.ajnr.org/content/27/6/1230/F1.large.jpg

Follow along at: goo.gl/1fs9G7
FFT: a building block across disciplines

Follow along at: goo.gl/1fs9G7
FFT: a building block across disciplines

http://www.ajnr.org/content/27/6/1230/F1.large.jpg

Follow along at: goo.gl/1fs9G7
FFT: a building block across disciplines

http://www.wireless.vt.edu/symposium/2012/tutorials/sessionA2.html
http://www.ajnr.org/content/27/6/1230/F1.large.jpg

Follow along at: goo.gl/1fs9G7
FFT: a building block across disciplines

Follow along at: goo.gl/1fs9G7

http://www.wireless.vt.edu/symposium/2012/tutorials/sessionA2.html
http://www.ajnr.org/content/27/6/1230/F1.large.jpg
Survey of FFT libraries for CPU and GPU hardware

Follow along at: goo.gl/1fs9G7
Outline

• Forecast
• Introduction

• Background
• Approach (Optimizations)
• Results & Analysis
  – Optimizations in isolation
  – Optimizations in concert
  – Shuffle
• Conclusion

Follow along at: goo.gl/1fs9G7
Background (GPUs)

- GPU Memory Hierarchy

Follow along at: goo.gl/1fs9G7
Background (GPUs)

• GPU Memory Hierarchy

Follow along at: goo.gl/1fs9G7
Background (GPUs)

- GPU Memory Hierarchy
  - Global Memory
Background (GPUs)

- GPU Memory Hierarchy
  - Global Memory

Table: Memory Read Bandwidth for Radeon HD 6970

<table>
<thead>
<tr>
<th>Memory Unit</th>
<th>Read Bandwidth (TB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global</td>
<td>0.17</td>
</tr>
</tbody>
</table>

Follow along at: goo.gl/1fs9G7
Background (GPUs)

- GPU Memory Hierarchy
  - Global Memory
  - Image Memory

Table: Memory Read Bandwidth for Radeon HD 6970

<table>
<thead>
<tr>
<th>Memory Unit</th>
<th>Read Bandwidth (TB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1/L2 Cache</td>
<td>1.35 / 0.45</td>
</tr>
<tr>
<td>Global</td>
<td>0.17</td>
</tr>
</tbody>
</table>

Follow along at: goo.gl/1fs9G7
Background (GPUs)

- GPU Memory Hierarchy
  - Global Memory
  - Image Memory
  - Constant Memory

Table: Memory Read Bandwidth for Radeon HD 6970

<table>
<thead>
<tr>
<th>Memory Unit</th>
<th>Read Bandwidth (TB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>5.4</td>
</tr>
<tr>
<td>L1/L2 Cache</td>
<td>1.35 / 0.45</td>
</tr>
<tr>
<td>Global</td>
<td>0.17</td>
</tr>
</tbody>
</table>

Follow along at: goo.gl/1fs9G7
Background (GPUs)

- GPU Memory Hierarchy
  - Global Memory
  - Image Memory
  - Constant Memory
  - **Local Memory**

### Table: Memory Read Bandwidth for Radeon HD 6970

<table>
<thead>
<tr>
<th>Memory Unit</th>
<th>Read Bandwidth (TB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>5.4</td>
</tr>
<tr>
<td>Local</td>
<td>2.7</td>
</tr>
<tr>
<td>L1/L2 Cache</td>
<td>1.35 / 0.45</td>
</tr>
<tr>
<td>Global</td>
<td>0.17</td>
</tr>
</tbody>
</table>

Follow along at: [goo.gl/1fs9G7](http://goo.gl/1fs9G7)
Background (GPUs)

- GPU Memory Hierarchy
  - Global Memory
  - Image Memory
  - Constant Memory
  - Local Memory
  - Registers

Table: Memory Read Bandwidth for Radeon HD 6970

<table>
<thead>
<tr>
<th>Memory Unit</th>
<th>Read Bandwidth (TB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registers</td>
<td>16.2</td>
</tr>
<tr>
<td>Constant</td>
<td>5.4</td>
</tr>
<tr>
<td>Local</td>
<td>2.7</td>
</tr>
<tr>
<td>L1/L2 Cache</td>
<td>1.35 / 0.45</td>
</tr>
<tr>
<td>Global</td>
<td>0.17</td>
</tr>
</tbody>
</table>

Follow along at: goo.gl/1f59G7
Outline

- Forecast
- Introduction
- Background
- Approach (Optimizations)
  - Results & Analysis
    - Optimizations in isolation
    - Optimizations in concert
    - Shuffle
- Conclusion

Follow along at: goo.gl/1f59G7
Approach ("Human Compilation")
Approach ("Human Compilation")

Optimizations in isolation

Follow along at: goo.gl/1fs9G7
Approach ("Human Compilation")

Optimizations in isolation

Follow along at: goo.gl/1fs9G7
Approach ("Human Compilation")

Optimizations in isolation

- Characterize
- Collect
- Measure

Follow along at: goo.gl/1fs9G7
Approach ("Human Compilation")

Optimizations in isolation

Optimizations in concert

- Characterize
- Collect
- Measure

Follow along at: goo.gl/1fs9G7
Approach ("Human Compilation")

Optimizations in isolation

- Characterize
- Collect
- Measure

Optimizations in concert

Follow along at: goo.gl/1fs9G7
Approach (Optimizations*)

- **System-level**
  1. Register Preloading (RP)
  2. Vectorized Access/\{Vector, Scalar\} Math (VAVM, VASM)
  3. Constant Memory Usage (CM)
  4. Common Subexpression Elimination (CSE)
  5. Inlining (IL)
  6. Coalesced Global Access Pattern (CGAP)

- **Algorithm-level**
  7. Naïve Transpose (LM-CM)
  8. Compute/Transpose via LM (LM-CC)
  9. Compute/No Transpose via LM (LM-CT)

- **Architecture- and Algorithm-Level**
  10. Shuffle (SHFL)

* For a complete list of optimization, refer to Table 4 in “Towards a Performance-Portable FFT Library for Heterogeneous Computing”

Follow along at: goo.gl/1fs9G7
System-level Optimizations

1. Register Preloading (RP)

Follow along at: goo.gl/1fs9G7
System-level Optimizations

1. Register Preloading (RP)

Without Register Preloading

79  __kernel void unoptimized(__global float2 *buffer)
80 {
81    int index = ...;
82    buffer += index;
83
84  FFT4_in_order_output(&buffer[0], &buffer[4],
                             &buffer[8], &buffer[12]);
System-level Optimizations

1. Register Preloading (RP)

Without Register Preloading

```c
79 __kernel void unoptimized(__global float2 *buffer)  
80 {  
81    int index = ...;  
82    buffer += index;  
83    FFT4_in_order_output(&buffer[0], &buffer[4],  
84                          &buffer[8], &buffer[12]);
```

With Register Preloading

```c
79 __kernel void optimized(__global float2 *buffer)  
80 {  
81    int index = ...;  
82    buffer += index;  
83    __private float2 r0, r1, r2, r3; // Register Declaration  
84    // Explicit Loads  
85    r0 = buffer[0]; r1 = buffer[1]; r2 = buffer[2]; r3 = buffer[3];  
86    FFT4_in_order_output(&r0, &r1, &r2, &r3);
```
System-level Optimizations

2. **Vector Access** (float{2, 4, 8, 16})
System-level Optimizations

2. **Vector Access** (float\{2, 4, 8, 16\})

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>y</td>
<td>Z</td>
<td>w</td>
</tr>
</tbody>
</table>

Follow along at: goo.gl/1fs9G7
System-level Optimizations

2. **Vector Access** (float{2, 4, 8, 16})

\[
\begin{array}{cccc}
  x & x & x & x \\
  y & y & y & y \\
  z & z & z & z \\
  w & w & w & w \\
\end{array}
\]

\[
\begin{array}{cccc}
\end{array}
\]

- **Scalar Math (VASM)**
  - float + float

Follow along at: goo.gl/1fs9G7
System-level Optimizations

2. **Vector Access** (float\{2, 4, 8, 16\})

- **Scalar Math** (VASM)
  - float + float

Follow along at: goo.gl/1fs9G7
System-level Optimizations

2. **Vector Access** (float\{2, 4, 8, 16\})

```
X  X  X  X
Y  Y  Y  Y
Z  Z  Z  Z
W  W  W  W
```

```
```

- **Scalar Math (VASM)**
  - float + float

```
float  +  float  =  float
```

- **Vector Math (VAVM)**
  - float4 + float4

```
X  +  X
Y  +  Y
Z  +  Z
W  +  W
```

Follow along at: goo.gl/1fs9G7
System-level Optimizations

2. **Vector Access** (float\{2, 4, 8, 16\})

\[
\begin{align*}
&X \\
&Y \\
&Z \\
&W
\end{align*}
\]

- **Scalar Math** (VASM)
  - float + float

\[
\begin{align*}
&X \\
&Y \\
&Z \\
&W
\end{align*}
\]

- **Vector Math** (VAVM)
  - float4 + float4

\[
\begin{align*}
&X \\
&Y \\
&Z \\
&W
\end{align*}
\]

Follow along at: goo.gl/1fs9G7
Architecture- and Algorithm-Level Optimization

10. Shuffle
10. Shuffle
   - Enable efficient data communication
Architecture- and Algorithm-Level Optimization

10. Shuffle
   - Enable efficient data communication
     • Local Memory (the “old” way)

Follow along at: goo.gl/1fs9G7
Architecture- and Algorithm-Level Optimization

10. Shuffle
   - Enable efficient data communication
     - Local Memory (the “old” way)

Follow along at: goo.gl/1fs9G7
Architecture- and Algorithm-Level Optimization

10. Shuffle
   – Enable efficient data communication
     • Local Memory (the “old” way)
     • Shuffle (the “new” way)

Follow along at: goo.gl/1fs9G7
Architecture- and Algorithm-Level Optimization

10. Shuffle
   - Enable efficient data communication
     - Local Memory (the “old” way)
       - Registers
         - 1
         - 2
         - 3
         - 4
       - Shared Memory
         - 1
         - 2
         - 3
         - 4
     - Shuffle (the “new” way)
       - Registers
         - 1
         - 2
         - 3
         - 4
       - Registers
         - 4
         - 1
         - 2
         - 3

Follow along at: goo.gl/1fs9G7
Architecture- and Algorithm-Level Optimization

10. Shuffle
   - Evaluate shuffle using matrix transpose
10. Shuffle
   - Evaluate shuffle using matrix transpose
     • Matrix transpose is a data communication step in FFT

Follow along at: goo.gl/1fs9G7
10. Shuffle

- Evaluate shuffle using matrix transpose
  - Matrix transpose is a data communication step in FFT
  - Devised Shuffle Transpose Algorithm
    - Consists of horizontal (inter-thread shuffles) and vertical (intra-thread)

Follow along at: goo.gl/1fs9G7
Outline

• Forecast
• Introduction
• Background
• Approach (Optimizations)

• Results & Analysis
  – Optimizations in isolation
  – Optimizations in concert
  – Shuffle

• Conclusion

Follow along at: goo.gl/1fs9G7
Results (Experimental Testbed)

AMD Radeon HD 6970 (VLIW)

Follow along at: goo.gl/1fs9G7
Results (Experimental Testbed)

AMD Radeon HD 6970 (VLIW)

Follow along at: goo.gl/1fs9G7
Results (Experimental Testbed)

AMD Radeon HD 6970 (VLIW)

AMD Radeon HD 7970 (non-VLIW)

NVIDIA C2075 (Non-VLIW)

Follow along at: goo.gl/1fs9G7
Results (Experimental Testbed)

AMD Radeon HD 6970 (VLIW)

NVIDIA C2075 (Non-VLIW)

AMD Radeon HD 7970 (non-VLIW)

NVIDIA Kepler K20c (Non-VLIW)

Follow along at: goo.gl/1fs9G7
Results (Experimental Testbed)

- Application Setup
  - 1D FFT (batched), $N = 16\text{-}, 64\text{-}, \text{and } 256\text{- pts}$
  - 2D FFT (batched), $N = 256\times256$

- GPU Testbed

<table>
<thead>
<tr>
<th>Device</th>
<th>Cores</th>
<th>Peak Performance (GFLOPS)</th>
<th>Peak Bandwidth (GB/s)</th>
<th>Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMD Radeon HD 6970</td>
<td>1536</td>
<td>2703</td>
<td>176</td>
<td>VLIW</td>
</tr>
<tr>
<td>AMD Radeon HD 7970</td>
<td>2048</td>
<td>3788</td>
<td>264</td>
<td>Non-VLIW</td>
</tr>
<tr>
<td>NVIDIA Tesla C2075</td>
<td>448</td>
<td>1288</td>
<td>144</td>
<td>Non-VLIW</td>
</tr>
<tr>
<td>NVIDIA Tesla K20c</td>
<td>2496</td>
<td>4106</td>
<td>208</td>
<td>Non-VLIW</td>
</tr>
</tbody>
</table>

Follow along at: goo.gl/1fs9G7
Results

- Optimizations in Isolation
  - Radeon 7970
  - 256-pts

<table>
<thead>
<tr>
<th>Optimizations</th>
<th>Radeon 7970, 256-pts</th>
</tr>
</thead>
<tbody>
<tr>
<td>RP: Register Preloading</td>
<td></td>
</tr>
<tr>
<td>LM-{CM, CT, CC}: Local Memory-{Communication Only; Compute, No Transpose; Computation and Communication}</td>
<td></td>
</tr>
<tr>
<td>VASM{n}: Vectorized Access &amp; Scalar Math{floatn}</td>
<td></td>
</tr>
<tr>
<td>VAVM{n}: Vectorized Access &amp; Vector Math{floatn}</td>
<td></td>
</tr>
<tr>
<td>CM: Constant Memory Usage</td>
<td></td>
</tr>
<tr>
<td>CGAP: Coalesced Access Pattern</td>
<td></td>
</tr>
<tr>
<td>LU: Loop unrolling</td>
<td></td>
</tr>
<tr>
<td>CSE: Common subexpression elimination</td>
<td></td>
</tr>
<tr>
<td>IL: Function inlining</td>
<td></td>
</tr>
<tr>
<td>Baseline: VASM2</td>
<td></td>
</tr>
</tbody>
</table>
Results

- Optimizations in Isolation
  - NVIDIA K20c
  - 256-pts
Results (Observations)

1. Use scalar operations (e.g., vector access/scalar math)
2. Focus should be on memory subsystem (e.g., bus traffic)

Bus Traffic (MB) = 2^{-20} \times (\text{bytes}_{\text{loaded}} + \text{bytes}_{\text{stored}})
Results (Bus Traffic)

Radeon 7970: Execution Time (256-pts)

- Twiddles
- Transpose
- Cols

RP: Register Preloading; LM-\{CM, CT, CC\}: Local Memory-\{Communication Only; Compute, No Transpose; Computation and Communication\}; VASM\{n\}: Vectorized Access & Scalar Math\{float\}; VAVM\{n\}: Vectorized Access & Vector Math\{float\}; CM: Constant Memory Usage; CGAP: Coalesced Access Pattern; LU: Loop unrolling; CSE: Common subexpression elimination; IL: Function inlining; Baseline: VASM2.
Results (Bus Traffic)

Radeon 7970: Execution Time (256-pts)

Radeon 7970: Bus Traffic (256-pts)

RP: Register Preloading; LM-\{CM, CT, CC\}: Local Memory-\{Communication Only; Compute, No Transpose; Computation and Communication\}; VASM\{n\}: Vectorized Access & Scalar Math\{floatn\}; VAVM\{n\}: Vectorized Access & Vector Math\{floatn\}; CM: Constant Memory Usage; CGAP: Coalesced Access Pattern; LU: Loop unrolling; CSE: Common subexpression elimination; IL: Function inlining; Baseline: VASM2.
Results

Insight #1
Primary cost of FFT is in data movement
Results

Insight #1
Primary cost of FFT is in data movement

- Reduce bus traffic by ...
Results

Insight #1
Primary cost of FFT is in data movement

• Reduce bus traffic by ...
  – using optimizations that prefetch memory (RP)

RP: Register Preloading; LM-{CM, CT, CC}: Local Memory-{Communication Only; Compute, No Transpose; Computation and Communication}; VASM{n}: Vectorized Access & Scalar Math{floatn}; VAVM{n}: Vectorized Access & Vector Math{floatn}; CM: Constant Memory Usage; CGAP: Coalesced Access Pattern; LU: Loop unrolling; CSE: Common subexpression elimination; IL: Function inlining; Baseline: VASM2.
Results

Insight #1
Primary cost of FFT is in data movement

- Reduce bus traffic by ...
  - using optimizations that prefetch memory (RP)
  - staging transpose in scratchpad memory (LM-CM, LM-CC, LM-CT)

RP: Register Preloading; LM-{CM, CT, CC}: Local Memory-{Communication Only; Compute, No Transpose; Computation and Communication}; VASM{n}: Vectorized Access & Scalar Math{floatn}; VAVM{n}: Vectorized Access & Vector Math{floatn}; CM: Constant Memory Usage; CGAP: Coalesced Access Pattern; LU: Loop unrolling; CSE: Common subexpression elimination; IL: Function inlining; Baseline: VASM2.
Results

- Optimizations in Concert
  - AMD Radeon HD 7970
  - 256-pts
Results

- Optimizations in Concert
  - NVIDIA Tesla K20c
  - 256-pts

RP: Register Preloading; **LM-{CM, CT, CC}**: Local Memory-{Communication Only; Compute, No Transpose; Computation and Communication}; **VASM{n}**: Vectorized Access & Scalar Math{floatn}; **VAVM{n}**: Vectorized Access & Vector Math{floatn}; **CM**: Constant Memory Usage; **CGAP**: Coalesced Access Pattern; **LU**: Loop unrolling; **CSE**: Common subexpression elimination; **IL**: Function inlining; **Baseline**: VASM2.
Insight #2
One sequence of optimizations perform well for GPUs

- These optimizations are ...
  - RP (Register Preloading)
  - LM-CM (Local Memory Communication Only)
  - VASM2/4 (Vector Access, Scalar Math, float2/4)
  - CM (Constant Memory Usage)
  - CGAP (Coalesced Global Access Pattern)
Speed-up with Shuffle

• Overall Performance
  – Max. Speedup (Amdahl’s Law): \textbf{1.19}-fold
  – Achieved Speedup: \textbf{1.17}-fold

Follow along at: goo.gl/1fs9G7
Speed-up with Shuffle

• Overall Performance
  – Max. Speedup (Amdahl’s Law): 1.19-fold
  – Achieved Speedup: 1.17-fold

• Surprise Result
  – Goal: Accelerate communication (“gray bar”)

Follow along at: goo.gl/1fs9G7
Speed-up with Shuffle

- **Overall Performance**
  - Max. Speedup (Amdahl’s Law): *1.19-fold*
  - Achieved Speedup: *1.17-fold*

- **Surprise Result**
  - Goal: Accelerate communication (“gray bar”)
  - Result: Accelerated the computation also (“black bar”)

Follow along at: goo.gl/1fs9G7
Results

- 2D FFT (N = 256x256)
  - Optimizations:
    - RP
    - LM-CM
    - VASM2
    - CM
    - CGAP

<table>
<thead>
<tr>
<th>Device</th>
<th>Unoptimized</th>
<th>Optimized</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVIDIA Tesla K20c</td>
<td>300</td>
<td>11.76x</td>
</tr>
<tr>
<td>NVIDIA Tesla C2075</td>
<td>200</td>
<td>2.05x</td>
</tr>
<tr>
<td>AMD Radeon HD 6970</td>
<td>100</td>
<td>16.36x</td>
</tr>
<tr>
<td>AMD Radeon HD 7970</td>
<td>50</td>
<td>2.14x</td>
</tr>
</tbody>
</table>

GFLOPS

RP: Register Preloading; LM-\{CM, CT, CC\}: Local Memory-\{Communication Only; Compute, No Transpose; Computation and Communication\}; VASMn: Vectorized Access & Scalar Math\{floatn\}; VAVMn: Vectorized Access & Vector Math\{floatn\}; CM: Constant Memory Usage; CGAP: Coalesced Access Pattern; LU: Loop unrolling; CSE: Common subexpression elimination; IL: Function inlining; Baseline: VASM2.
Conclusion (Thank You!)

• **Title:** “Towards a Performance-Portable FFT Library for Heterogeneous Computing”

• **Contribution:** A methodology for determining portable optimizations for a class of algorithms
  - Optimization principles for FFT on GPUs
  - An analysis of GPU optimizations applied in isolation and in concert on AMD and NVIDIA GPU architectures

• **Insight #1:** Primary cost of FFT computation is in data movement (e.g., memory bound)

• **Insight #2:** One sequence of optimizations perform well for GPUs
  - [1D FFT] **31.5-fold** improvement over baseline GPU; **9.1-fold** improvement over multi-core FFTW CPU with AVX.

Follow along at: goo.gl/1fs9G7
Appendix Slides

Follow along at: goo.gl/1fs9G7
Background (Optimizing on GPUs)

1. **RP (Register Preloading)** - All data elements are first preloaded onto the register file of the respective GPU. Computation is facilitated solely on registers.

2. **CGAP (Coalesced Global Access Pattern)** - Threads access memory contiguously (the kth thread accesses memory element k).

3. **VASM2/4 (Vector Access, Scalar Math, float{2/4})** - Data elements are loaded as the listed vector type. Arithmetic operations are scalar (float x float).

4. **LM-CM (Local Memory, Communication Only)** - Data elements are loaded into local memory only for communication. Threads swap data elements solely in local memory.

5. **LM-CT (Local Memory, Computation, No Transpose)** - Data elements are loaded into local memory for computation. The communication step is avoided by algorithm reorganization.

6. **LM-CC (Local Memory, Computation and Communication)** - All data elements are preloaded into local memory. Computation is performed in local memory, while registers are used for scratchpad communication.

7. **CM-[K,L] (Constant Memory – {Kernel, Literal})** - The twiddle multiplication stage of FFT is precomputed on the CPU and stored in the GPU constant memory for fast look up. CM-K refers to constant memory as a kernel argument, while CM-L refers to a static global declaration in the OpenCL kernel.

8. **CSE (Common Subexpression Elimination)** - A traditional optimization that collapses identical expressions in order to save computation. This optimization may increase register live time, therefore, increasing register pressure.

9. **IL (Function Inlining)** - A function's code body is inserted in place of a function call. It is used primarily for functions that are frequently called.

10. **LU (Loop Unrolling)** – A loop is explicitly rewritten as an identical sequence of statements without the overhead of loop variable comparisons.

11. **Shuffle** - The transpose stage in FFT is performed entirely in registers eliminating the use of local memory. This optimization is only possible with NVIDIA Kepler GPUs (e.g., Tesla K20c).

Follow along at: goo.gl/1fs9G7
S3: Constant Memory

- Fast cached lookup for frequently used data

Follow along at: goo.gl/1fs9G7
S3: Constant Memory

- Fast cached lookup for frequently used data

```
16 __constant float2 twiddles[16] = { (float2)(1.0f,0.0f), (float2)
(1.0f,0.0f), (float2)(1.0f,0.0f), (float2)(1.0f,0.0f),
... more sin/cos values};
```

**Without Constant Memory**

```
61 for (int j = 1; j < 4; ++j)
62 {
63  double theta = -2.0 * M_PI * tid * j / 16;
64  float2 twid = make_float2(cos(theta), sin(theta));
65  result[j] = buffer[j*4] * twid;
66 }
```

**With Constant Memory**

```
61 for (int j = 1; j < 4; ++j)
62  result[j] = buffer[j*4] * twiddles[4*j+tid];
```

Follow along at: goo.gl/1fs9G7
System-level Optimizations

Follow along at: goo.gl/1fs9G7
Approach

- System-level Optimizations (applicable to any application)
  1. Register Preloading
  2. Vector Access/{Vector,Scalar} Arithmetic
  3. Constant Memory Usage
  4. Dynamic Instruction Reduction
  5. Memory Coalescing
  6. Image Memory

- Algorithm-level Optimizations

Algorithm-level optimizations

- Transpose – elements across the diagonal are exchanged
Algorithm-level optimizations

- Transpose – elements across the diagonal are exchanged

4x4 matrix

Transposed matrix

Follow along at: goo.gl/1fs9G7
Algorithm-level optimizations

- Transpose – elements across the diagonal are exchanged

Follow along at: goo.gl/1fs9G7
Algorithm-level optimizations

- Transpose – elements across the diagonal are exchanged

4x4 matrix

Transposed matrix

Follow along at: goo.gl/1fs9G7
Algorithm-level optimizations

- Transpose – elements across the diagonal are exchanged

4x4 matrix  Transposed matrix
Algorithm-level optimizations

- Transpose – elements across the diagonal are exchanged

Follow along at: goo.gl/1fs9G7
Algorithm-level optimizations

Follow along at: goo.gl/1fs9G7
Algorithm-level optimizations

1. Naïve Transpose (LM-CM)

Follow along at: goo.gl/1fs9G7
Algorithm-level optimizations

1. Naïve Transpose (LM-CM)
Algorithm-level optimizations

1. Naïve Transpose (LM-CM)

Follow along at: goo.gl/1fs9G7
Algorithm-level optimizations

1. Naïve Transpose (LM-CM)

Follow along at: goo.gl/1fs9G7
Algorithm-level optimizations

3. The pseudo transpose (LM-CT)

Follow along at: goo.gl/1fs9G7
Algorithm-level optimizations

3. The pseudo transpose (LM-CT)

Follow along at: goo.gl/1fs9G7
Algorithm-level optimizations

3. The pseudo transpose (LM-CT)
   - Idea:
     • Load data to local memory

Follow along at: goo.gl/1fs9G7
Algorithm-level optimizations

3. The pseudo transpose (LM-CT)
   - Idea:
     * Load data to local memory

Follow along at: goo.gl/1fs9G7
Algorithm-level optimizations

3. The pseudo transpose (LM-CT)
   - Idea:
     - Load data to local memory
     - Perform computation on columns,

Follow along at: goo.gl/1fs9G7
Algorithm-level optimizations

3. The pseudo transpose (LM-CT)
   - Idea:
     • Load data to local memory
     • Perform computation on columns, then **rows**.

Follow along at: [goo.gl/1fs9G7](http://goo.gl/1fs9G7)
Algorithm-level optimizations

3. The pseudo transpose (LM-CT)
   - Idea:
     - Load data to local memory
     - Perform computation on columns, then rows.
   - Advantage:
     - Skips the transpose step

Follow along at: goo.gl/1fs9G7
Algorithm-level optimizations

3. The pseudo transpose (LM-CT)
   - Idea:
     - Load data to local memory
     - Perform computation on columns, then rows.
   - Advantage:
     - Skips the transpose step
   - Disadvantage:
     - Local memory has lower throughput than registers.

Follow along at: goo.gl/1fs9G7
Architecture-level Optimization: Shuffle

Software (Transpose)

Hardware (K20c and shuffle)

Follow along at: goo.gl/1fs9G7
Results (Shuffle)

- Bottleneck: **Intra-thread** data movement
Results (Shuffle)

- **Bottleneck:** Intra-thread data movement

Code 1: *(NAIVE)*

```c
for (int k = 0; k < 4; ++k)
    dst_registers[k] = src_registers[(4 - tid + k) % 4];
```

Follow along at: [goo.gl/1fs9G7](https://goo.gl/1fs9G7)
Results (Shuffle)

Code 1 (NAIVE)

```c
63    for (int k = 0; k < 4; ++k)
64      dst_registers[k] = src_registers[(4 - tid + k) % 4];
```

General strategies
- Registers are fast.
- CUDA local memory is slow.

- Compiler is forced to place data into CUDA local memory if array indices CANNOT be determined at compile time.

Follow along at: goo.gl/1fs9G7
Results (Shuffle)

Code 1 (NAIVE)

```c
for (int k = 0; k < 4; ++k)
dst_registers[k] = src_registers[(4 - tid + k) % 4];
```

Code 2 (DIV)

```c
int tmp = src_registers[0];
if (tid == 1)
{
    src_registers[0] = src_registers[3];
    src_registers[3] = src_registers[2];
    src_registers[2] = src_registers[1];
    src_registers[1] = tmp;
}
else if (tid == 2)
{
    src_registers[0] = src_registers[2];
    tmp = src_registers[1];
    src_registers[1] = src_registers[3];
}
else if (tid == 3)
{
    src_registers[0] = src_registers[1];
    src_registers[1] = src_registers[2];
    src_registers[2] = src_registers[3];
}
```

General strategies

- Registers are fast.
- CUDA local memory is slow.

Compiler is forced to place data into CUDA local memory if array indices CANNOT be determined at compile time.

Follow along at: goo.gl/1fs9G7
**Results (Shuffle)**

### Code 1 (NAIVE)

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>for (int k = 0; k &lt; 4; ++k)</td>
</tr>
<tr>
<td>64</td>
<td>dst_registers[k] = src_registers[(4 - tid + k) % 4];</td>
</tr>
</tbody>
</table>

**General strategies**

- Registers are fast.
- CUDA local memory is slow.

- Compiler is forced to place data into CUDA local memory if array indices CANNOT be determined at compile time.

### Code 2 (DIV)

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>15X</td>
<td>int tmp = src_registers[0];</td>
</tr>
<tr>
<td></td>
<td>if (tid == 1)</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>src_registers[0] = src_registers[3];</td>
</tr>
<tr>
<td></td>
<td>src_registers[3] = src_registers[2];</td>
</tr>
<tr>
<td></td>
<td>src_registers[2] = src_registers[1];</td>
</tr>
<tr>
<td></td>
<td>src_registers[1] = tmp;</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>else if (tid == 2)</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>src_registers[0] = src_registers[2];</td>
</tr>
<tr>
<td></td>
<td>src_registers[1] = src_registers[3];</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td>else if (tid == 3)</td>
</tr>
<tr>
<td></td>
<td>{</td>
</tr>
<tr>
<td></td>
<td>src_registers[0] = src_registers[1];</td>
</tr>
<tr>
<td></td>
<td>src_registers[1] = src_registers[2];</td>
</tr>
<tr>
<td></td>
<td>src_registers[2] = src_registers[3];</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

**Divergence**

Follow along at: goo.gl/1fs9G7
Results (Shuffle)

Code 1 (NAIVE)

```
63   for (int k = 0; k < 4; ++k)
64       dst_registers[k] = src_registers[(4 * tid + k) % 4];
```

Code 2 (DIV)

```
int tmp = src_registers[0];
if (tid == 1)
{
    src_registers[0] = src_registers[3];
    src_registers[3] = src_registers[2];
    src_registers[2] = src_registers[1];
    src_registers[1] = tmp;
}
else if (tid == 2)
{
    src_registers[0] = src_registers[2];
    tmp = src_registers[1];
    src_registers[1] = src_registers[3];
}
else if (tid == 3)
{
    src_registers[0] = src_registers[1];
    src_registers[1] = src_registers[2];
    src_registers[2] = src_registers[3];
}
```

Code 3 (SELP OOP)

```
65   dst_registers[0] = (tid == 0) ? src_registers[0] : dst_registers[0];
69
70   dst_registers[0] = (tid == 1) ? src_registers[3] : dst_registers[0];
73   dst_registers[1] = (tid == 1) ? src_registers[0] : dst_registers[1];
74
75   dst_registers[0] = (tid == 2) ? src_registers[2] : dst_registers[0];
76   dst_registers[2] = (tid == 2) ? src_registers[0] : dst_registers[2];
79
80   dst_registers[0] = (tid == 3) ? src_registers[1] : dst_registers[0];
83   dst_registers[3] = (tid == 3) ? src_registers[0] : dst_registers[3];
```

General strategies
- Registers are fast.
- CUDA local memory is slow.

Compiler is forced to place data into CUDA local memory if array indices CANNOT be determined at compile time.

Follow along at goo.gl/1fs9G7
Results (Shuffle)

### Code 1 (NAIVE)

63 for (int k = 0; k < 4; ++k)
64 dst_registers[k] = src_registers[(4 - tid + k) % 4];

### Code 2 (DIV)

int tmp = src_registers[0];
if (tid == 1)
{
    src_registers[0] = src_registers[3];
    src_registers[3] = src_registers[2];
    src_registers[2] = src_registers[1];
    src_registers[1] = tmp;
}
else if (tid == 2)
{
    src_registers[0] = src_registers[2];
    tmp = src_registers[1];
    src_registers[1] = src_registers[3];
}
else if (tid == 3)
{
    src_registers[0] = src_registers[1];
    src_registers[1] = src_registers[2];
    src_registers[2] = src_registers[3];
}

### Code 3 (SELP OOP)

65 dst_registers[0] = (tid == 0) ? src_registers[0] : dst_registers[0];
69
70 dst_registers[0] = (tid == 1) ? src_registers[3] : dst_registers[0];
73 dst_registers[1] = (tid == 1) ? src_registers[0] : dst_registers[1];
74
75 dst_registers[0] = (tid == 2) ? src_registers[2] : dst_registers[0];
76 dst_registers[2] = (tid == 2) ? src_registers[0] : dst_registers[2];
79
80 dst_registers[0] = (tid == 3) ? src_registers[1] : dst_registers[0];
83 dst_registers[3] = (tid == 3) ? src_registers[0] : dst_registers[3];

Follow along at goo.gl/1fs9G7
Results (Shuffle)

Execution Time (ms)

- Shm
- Naive
- DIV
- SELP (IP)
- SELP (OOP)
- SELP (IP)

% improvement for communication:
- Shm: 6%
- Naive: 17%
- DIV: 44%

Follow along at: goo.gl/1fs9G7
Results (Shuffle)

<table>
<thead>
<tr>
<th>Method</th>
<th>Execution Time (ms)</th>
<th>% improvement for communication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shm</td>
<td>37.5%</td>
<td></td>
</tr>
<tr>
<td>Naive</td>
<td>44%</td>
<td>6%</td>
</tr>
<tr>
<td>DIV</td>
<td>14%</td>
<td>17%</td>
</tr>
<tr>
<td>SELP (IP)</td>
<td>15x</td>
<td>44%</td>
</tr>
<tr>
<td>SELP (OOP)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Follow along at: goo.gl/1fs9G7