Today

GPU performance

MPI performance

Parallel Patterns
Outline

GPU performance
  Understanding GPUs
  GPUs and Memory
  Summary

MPI performance

Parallel Patterns
GPU performance

Understanding GPUs

GPUs and Memory

Summary

MPI performance

Parallel Patterns
Recap

• SIMD performance impact?
Recap

- SIMD performance impact?
- How can GPU code deal with latency?
Recap

• SIMD performance impact?
• How can GPU code deal with latency?
• Difference: \# FPUs / \# scheduling slots?
### Comparing architectures

<table>
<thead>
<tr>
<th></th>
<th>Nvidia GF100</th>
<th>Nvidia GF104</th>
<th>Nvidia GK104</th>
<th>AMD GCN</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td># Warps/core</td>
<td>48</td>
<td>48</td>
<td>64</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>Core clock</td>
<td>1400</td>
<td>1300</td>
<td>823</td>
<td>925</td>
<td>MHz</td>
</tr>
<tr>
<td>Reg File</td>
<td>128</td>
<td>128</td>
<td>256</td>
<td>256</td>
<td>kiB</td>
</tr>
<tr>
<td>Lmem/core</td>
<td>64</td>
<td>64</td>
<td>64</td>
<td>64</td>
<td>kiB</td>
</tr>
<tr>
<td>Lmem BW/core</td>
<td>64</td>
<td>64</td>
<td>128</td>
<td>128</td>
<td>B/clock</td>
</tr>
<tr>
<td>GMem Bus</td>
<td>384</td>
<td>256</td>
<td>256</td>
<td>384</td>
<td>Bits</td>
</tr>
<tr>
<td>GMem Clock</td>
<td>3696</td>
<td>3600</td>
<td>6008</td>
<td>5500</td>
<td>MHz</td>
</tr>
<tr>
<td>Core clock</td>
<td>1400</td>
<td>1300</td>
<td>823</td>
<td>925</td>
<td>MHz</td>
</tr>
<tr>
<td>Reg File</td>
<td>128</td>
<td>128</td>
<td>256</td>
<td>256</td>
<td>kiB</td>
</tr>
<tr>
<td>Lmem/core</td>
<td>64</td>
<td>64</td>
<td>64</td>
<td>64</td>
<td>kiB</td>
</tr>
<tr>
<td>Lmem BW/core</td>
<td>64</td>
<td>64</td>
<td>128</td>
<td>128</td>
<td>B/clock</td>
</tr>
<tr>
<td>GMem Bus</td>
<td>384</td>
<td>256</td>
<td>256</td>
<td>384</td>
<td>Bits</td>
</tr>
<tr>
<td>GMem Clock</td>
<td>3696</td>
<td>3600</td>
<td>6008</td>
<td>5500</td>
<td>MHz</td>
</tr>
</tbody>
</table>

What are the main limits for programs? What happens if you exceed them? GPU performance, MPI performance, Parallel Patterns.
### Comparing architectures

<table>
<thead>
<tr>
<th></th>
<th>Nvidia GF100</th>
<th>Nvidia GF104</th>
<th>Nvidia GK104</th>
<th>AMD GCN</th>
<th>Units</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td># Warps/core</td>
<td>48</td>
<td>48</td>
<td>64</td>
<td>40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Warp Size</td>
<td>32</td>
<td>32</td>
<td>32</td>
<td>64</td>
<td>W.Item</td>
<td></td>
</tr>
<tr>
<td>SP FPUs/core</td>
<td>32</td>
<td>48</td>
<td>192</td>
<td>64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cores</td>
<td>15</td>
<td>7</td>
<td>8</td>
<td>32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Core clock</td>
<td>1400</td>
<td>1300</td>
<td>823</td>
<td>925</td>
<td>MHz</td>
<td></td>
</tr>
<tr>
<td>Reg File</td>
<td>128</td>
<td>128</td>
<td>256</td>
<td>256</td>
<td>kiB</td>
<td></td>
</tr>
<tr>
<td>Lmem/core</td>
<td>64</td>
<td>64</td>
<td>64</td>
<td>64</td>
<td>kiB</td>
<td>kiB</td>
</tr>
<tr>
<td>Lmem BW/core</td>
<td>64</td>
<td>64</td>
<td>128</td>
<td>128</td>
<td>B/clock</td>
<td></td>
</tr>
<tr>
<td>GMem Bus</td>
<td>384</td>
<td>256</td>
<td>256</td>
<td>384</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GMem Clock</td>
<td>3696</td>
<td>3600</td>
<td>6008</td>
<td>5500</td>
<td>MHz</td>
<td></td>
</tr>
</tbody>
</table>

What are the main limits for programs?

What happens if you exceed them?
Occupancy calculator
Performance in three sentences

Flops are cheap
Bandwidth is money
Latency is physics

[M. Hoemmen]
Parallel Memories: Different Approaches

Problem

Digital memories have only one data bus.
Problem
Digital memories have only one data bus.
So how can multiple threads read multiple data items from memory simultaneously?
Problem
Digital memories have only one data bus.
So how can multiple threads read multiple data items from memory simultaneously?

Solutions: Parallel Access to Memory

• Split a really wide data bus, but have only one address bus
Parallel Memories: Different Approaches

Problem
Digital memories have only one data bus. So how can multiple threads read multiple data items from memory simultaneously?

Solutions: Parallel Access to Memory

- Split a really wide data bus, but have only one address bus
- Have many “small memories” (“banks”) with separate address busses. Pick bank by LSB of address.
Global Memory

Rule of thumb

\[ n = \min \left( \frac{\text{Bus width in bits}}{\text{Word size in bits}}, \text{SIMD group size} \right) \]

work items access global memory simultaneously. Full utilization only if all bits in bus transaction are useful.
Global Memory

Rule of thumb

\[ n = \min \left( \frac{\text{Bus width in bits}}{\text{Word size in bits}}, \text{SIMD group size} \right) \]

work items access global memory simultaneously. Full utilization only if all bits in bus transaction are useful.

\( n \) words

\[ \cdots \]
Global Memory

Rule of thumb

\[ n = \min \left( \frac{\text{Bus width in bits}}{\text{Word size in bits}}, \text{SIMD group size} \right) \]

work items access global memory simultaneously. Full utilization only if all bits in bus transaction are useful.

\[ n \text{ words} \]

OK: `global_variable[get_global_id(0)]`
(Single transaction)
Global Memory

Rule of thumb

\[ n = \min \left( \frac{\text{Bus width in bits}}{\text{Word size in bits}}, \text{SIMD group size} \right) \]

work items access global memory simultaneously. Full utilization only if all bits in bus transaction are useful.

\[ n \text{ words} \]

Bad: `global_variable[5+get_global_id(0)]`
(Two transactions)
Global Memory

Rule of thumb

\[ n = \min \left( \frac{\text{Bus width in bits}}{\text{Word size in bits}}, \text{SIMD group size} \right) \]

work items access global memory simultaneously. Full utilization only if all bits in bus transaction are useful.
Global Memory

Rule of thumb

\[ n = \min \left( \frac{\text{Bus width in bits}}{\text{Word size in bits}}, \text{SIMD group size} \right) \]

work items access global memory simultaneously. Full utilization only if all bits in bus transaction are useful.
Global Memory

Rule of thumb

\[ n = \min \left( \frac{\text{Bus width in bits}}{\text{Word size in bits}}, \text{SIMD group size} \right) \]

work items access global memory simultaneously. Full utilization only if all bits in bus transaction are useful.

Bad: \text{global\_variable}[2*\text{get\_global\_id}(0)]
(Two transactions)
GPU global access patterns demo
Local Memory: Banking

Bank  Address

Example: Nvidia GT200 has 16 banks. Work items access local memory in groups of 16.
Local Memory: Banking

Bank

3  7  11  15  19  23  ...

2  6  10  14  18  22  ...

1  5  9  13  17  21  ...

0  4  8  12  16  20  ...

Address

Bad: local variable[get local id(0)]
(Single cycle)

OK: local variable(BANK COUNT*get local id(0))
(BANK COUNT cycles)

OK: local variable[(BANK COUNT+1)*get local id(0)]
(Single cycle)

OK: local variable[ODD NUMBER*get local id(0)]
(Single cycle)

Bad: local variable[2*get local id(0)]
(BANK COUNT/2 cycles)

OK: local variable[f(get group id(0))]
(Broadcast–single cycle)

Example: Nvidia GT200 has 16 banks.
Work items access local memory in groups of 16.

GPU performance  MPI performance  Parallel Patterns
Local Memory: Banking

Example: Nvidia GT200 has 16 banks. Work items access local memory in groups of 16.
Local Memory: Banking

Example: Nvidia GT200 has 16 banks. Work items access local memory in groups of 16.

OK: `local_variable[get_local_id(0)]`, (Single cycle)
Local Memory: Banking

**Bad:** local_variable[BANK_COUNT*get_local_id(0)]
(BANK_COUNT cycles)
Local Memory: Banking

Example: Nvidia GT200 has 16 banks. Work items access local memory in groups of 16.

**OK:** `local_variable[(BANK_COUNT+1)*get_local_id(0)]`
(Single cycle)
**Local Memory: Banking**

OK: `local_variable[ODD_NUMBER*get_local_id(0)]`
(Single cycle)
Local Memory: Banking

- **OK:** `local_variable[get_local_id(0)]` (Single cycle)
- **Bad:** `local_variable[BANK_COUNT*get_local_id(0)]` (BANK_COUNT cycles)
- **OK:** `local_variable[(BANK_COUNT+1)*get_local_id(0)]` (Single cycle)
- **OK:** `local_variable[ODD NUMBER*get_local_id(0)]` (Single cycle)
- **Bad:** `local_variable[2*get_local_id(0)]` (BANK_COUNT/2 cycles)

Example: Nvidia GT200 has 16 banks. Work items access local memory in groups of 16.
Local Memory: Banking

Example: Nvidia GT200 has 16 banks. Work items access local memory in groups of 16.

OK: `local_variable[f(get_group_id(0))]`
(Broadcast–single cycle)
Example: Nvidia GT200 has 16 banks. Work items access local memory in groups of 16.
GPU local Memory

GPU local access patterns demo
GPU local Memory

GPU local access patterns

demo

What does this mean for 2D arrays in local memory? (E.g. matrix transpose?)
What does this mean for 2D arrays in local memory? (E.g. matrix transpose?)

What does this mean for doubles in local memory?
Faster transfers Host ↔ GPU

How about host ↔ device transfers?

• If talking to CPU: Unnecessary

• If talking to GPU:
  • Want asynchronous transfer
  • Want overlapping transfer

What about paging?
How about host ↔ device transfers?

- If talking to CPU: Unnecessary `CL_MEM_ALLOC_HOST_PTR`
- If talking to GPU:
  - Want asynchronous transfer
  - Want overlapping transfer

What about paging?
Faster transfers Host ↔ GPU

How about host ↔ device transfers?

- If talking to CPU: Unnecessary
  CL_MEM_ALLOC_HOST_PTR
- If talking to GPU:
  - Want asynchronous transfer
  - Want overlapping transfer

What about paging?
CL_MEM_ALLOC_HOST_PTR

(‘pinned’ memory—Demo)
Faster transfers Host ↔ GPU

How about host ↔ device transfers?

- If talking to CPU: Unnecessary
  CL_MEM_ALLOC_HOST_PTR
- If talking to GPU:
  - Want asynchronous transfer
  - Want overlapping transfer

What about paging?
CL_MEM_ALLOC_HOST_PTR

(‘pinned’ memory—

Important: Two different mechanisms at work!)
Too little memory?

Efficient code organization for out-of-core calculations?

**Assume:** \(\leftarrow, \rightarrow\) transfers, computation all proceed independently.
Too little memory?

Efficient code organization for out-of-core calculations?

**Assume:** $\leftarrow$, $\rightarrow$ transfers, computation all proceed independently.

“Double buffering”

**Idea:** Just keep everybody busy.
Too little memory?

Efficient code organization for out-of-core calculations?

Assume: \(\leftarrow, \rightarrow\) transfers, computation all proceed independently.

“Double buffering”

Idea: Just keep everybody busy.

Q: Describe that in OpenCL without synchronizing the host to the GPU.
## Entertainment: GPU Memory Zoo

<table>
<thead>
<tr>
<th>Type</th>
<th>Per</th>
<th>Access</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>private</td>
<td>work item</td>
<td>R/W</td>
<td>1 or 1000</td>
</tr>
<tr>
<td>local</td>
<td>group</td>
<td>R/W</td>
<td>2</td>
</tr>
<tr>
<td>global</td>
<td>grid</td>
<td>R/W</td>
<td>1000</td>
</tr>
<tr>
<td>constant</td>
<td>grid</td>
<td>R/O</td>
<td>1-1000</td>
</tr>
<tr>
<td>imagentd_t</td>
<td>grid</td>
<td>R/(W)</td>
<td>1000</td>
</tr>
</tbody>
</table>
## Entertainment: GPU Memory Zoo

<table>
<thead>
<tr>
<th>Type</th>
<th>Per</th>
<th>Access</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>private</td>
<td>work item</td>
<td>R/W</td>
<td>1 or 1000</td>
</tr>
<tr>
<td>local</td>
<td>group</td>
<td>R/W</td>
<td>2</td>
</tr>
<tr>
<td>global</td>
<td>grid</td>
<td>R/W</td>
<td>1000</td>
</tr>
<tr>
<td>constant</td>
<td>grid</td>
<td>R/O</td>
<td>1-1000</td>
</tr>
<tr>
<td>image&lt;nd_t</td>
<td>grid</td>
<td>R(/W)</td>
<td>1000</td>
</tr>
</tbody>
</table>
Outline

GPU performance
  Understanding GPUs
  GPUs and Memory
  Summary

MPI performance

Parallel Patterns
GPU performance summary

- Latency, latency, latency!
  - Various forms: Memory, branches, computation
  - All need to be hidden
- Bandwidth: usually fixable
- Watch your memory access patterns
  - Local mem is somewhat more forgiving
  - ... and lower latency, higher BW
GPU profiler demo
Outline

GPU performance

MPI performance

Parallel Patterns
MPI performance demo
Understanding Computational Cost
3

Concepts, Patterns and Recipes
Outline

GPU performance

MPI performance

Parallel Patterns

  Embarrassingly Parallel

  Partition
Parallel Programming:
  • To what problems does it apply?
  • How?
    • How big of a headache?
  • What mechanism is suitable?

Organize discussion by patterns of **Dependencies**.
Patterns: Overview

Parallel Programming:

- To what problems does it apply?
- How?
  - How big of a headache?
- What mechanism is suitable?

Organize discussion by patterns of Dependencies.

Will move to more of a discussion style
Outline

GPU performance

MPI performance

Parallel Patterns
   Embarrassingly Parallel
   Partition
Embarrassingly Parallel

\[ y_i = f_i(x_i) \]

where \( i \in \{1, \ldots, N\} \).

Notation: (also for rest of this lecture)

- \( x_i \): inputs
- \( y_i \): outputs
- \( f_i \): (pure) functions (i.e. no side effects)

When does a function have a “side effect”? In addition to producing a value, it
- modifies non-local state, or
- has an observable interaction with the outside world.

Often:
\[ f_1 = \cdots = f_N. \] Then
- Lisp/Python function map
- C++ STL std::transform

GPU performance MPI performance Parallel Patterns
Embarrassingly Parallel

When does a function have a “side effect”? In addition to producing a value, it
• modifies non-local state, or
• has an observable interaction with the outside world.

Notation: (also for rest of this lecture)
• \( x_i \): inputs
• \( y_i \): outputs
• \( f_i \): (pure) functions (i.e. no side effects)
Embarrassingly Parallel

\[ y_i = f_i(x_i) \]

where \( i \in \{1, \ldots, N\} \).

Notation: (also for rest of this lecture)

- \( x_i \): inputs
- \( y_i \): outputs
- \( f_i \): (pure) functions (i.e. no side effects)

When does a function have a "side effect"?
In addition to producing a value, it
- modifies non-local state, or
- has an observable interaction with the outside world.

Often:
\[ f_1 = \cdots = f_N \]

Then
- Lisp/Python function `map`
- C++ STL `std::transform`
Embarrassingly Parallel

\[ y_i = f_i(x_i) \]

where \( i \in \{1, \ldots, N\} \).

Notation: (also for rest of this lecture)

- \( x_i \): inputs
- \( y_i \): outputs
- \( f_i \): (pure) functions (i.e. no side effects)

Often: \( f_1 = \cdots = f_N \). Then

- Lisp/Python function map
- C++ STL std::transform

When does a function have a “side effect”?
In addition to producing a value, it
- modifies non-local state, or
- has an observable interaction with the outside world.
Embarrassingly Parallel: Graph Representation

Trivial? Often: no.

GPU performance  MPI performance  Parallel Patterns
Embarrassingly Parallel: Graph Representation

Trivial? Often: no.
Embarrassingly Parallel: Examples

Surprisingly useful:

- Element-wise linear algebra: Addition, scalar multiplication (not inner product)
- Image Processing: Shift, rotate, clip, scale, . . .
- Monte Carlo simulation
- (Brute-force) Optimization
- Random Number Generation
- Encryption, Compression (after blocking)
Embarrassingly Parallel: Examples

Surprisingly useful:

- Element-wise linear algebra:
  Addition, scalar multiplication (not inner product)
- Image Processing: Shift, rotate, clip, scale, . . .
- Monte Carlo simulation
- (Brute-force) Optimization
- Random Number Generation
- Encryption, Compression (after blocking)

But: Still needs a minimum of coordination. How can that be achieved?
Mapping to Mechanisms

- Single threads?
Mapping to Mechanisms

- Single threads?
- OpenMP?
Mapping to Mechanisms

- Single threads?
- OpenMP?
- MPI?

GPU performance  MPI performance  Parallel Patterns
Mapping to Mechanisms

- Single threads?
- OpenMP?
- MPI?
- MPI: Larger than # ranks?
Mapping to Mechanisms

- Single threads?
- OpenMP?
- MPI?
- MPI: Larger than \# ranks?
- GPU?
Embarrassingly Parallel: Issues

- Process Creation: Dynamic/Static?
  - MPI 2 supports dynamic process creation
- Job Assignment (‘Scheduling’): Dynamic/Static?
- Operations/data light- or heavy-weight?
- Variable-size data?
- Load Balancing:
  - Here: easy
Embarrassingly Parallel: Issues

- Process Creation: Dynamic/Static?
  - MPI 2 supports dynamic process creation
- Job Assignment (‘Scheduling’): Dynamic/Static?
- Operations/data light- or heavy-weight?
- Variable-size data?
- Load Balancing:
  - Can you think of a load balancing recipe?
Mother-Child Parallelism

Mother-Child parallelism:

Send initial data

Collect results

(formerly called “Master-Slave”)

(formerly called “Master-Slave”)
Outline

- GPU performance
- MPI performance
- Parallel Patterns
  - Embarrassingly Parallel
  - Partition
\[ y_i = f_i(x_{i-1}, x_i, x_{i+1}) \]

where \( i \in \{1, \ldots, N\} \).
\[ y_i = f_i(x_{i-1}, x_i, x_{i+1}) \]

where \( i \in \{1, \ldots, N\} \).

Includes straightforward generalizations to dependencies on a larger (but not \( O(P) \)-sized!) set of neighbor inputs.
\[ y_i = f_i(x_{i-1}, x_i, x_{i+1}) \]

where \( i \in \{1, \ldots, N\} \).

Includes straightforward generalizations to dependencies on a larger (but not \( O(P) \)-sized!) set of neighbor inputs.

**Point:** Processor \( i \) owns \( x_i \). ("owns" = is "responsible for updating")
Partition: Graph

GPU performance  MPI performance  Parallel Patterns
Mapping to Mechanisms

- OpenMP?
Mapping to Mechanisms

- OpenMP?
- MPI?
Mapping to Mechanisms

- OpenMP?
- MPI?
- MPI: Larger than \# ranks?
Mapping to Mechanisms

- OpenMP?
- MPI?
- MPI: Larger than \# ranks?
- GPU?
Partitioning for neighbor communication
Partitioning for neighbor communication

How can I chop up a domain?
Questions?
Image Credits

- Field: sxc.hu/mzacha