High-Performance Scientific Computing
Lecture 12: GPU Performance, Applications

MATH-GA 2011 / CSCI-GA 2945 · November 28, 2012

Today

GPU performance

MPI performance

Parallel Patterns

Outline

GPU performance

- Understanding GPUs
- GPUs and Memory
- Summary

MPI performance

Parallel Patterns

Outline

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

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

David Kanter / Realworldtech.com

Comparing architectures

	Nvidia	Nvidia	Nvidia	AMD	Units
	GF100	GF104	GK104	GCN	Units
# Warps/core	48	48	64	40	
Warp Size	32	32	32	64	W.Item
SP FPUs/core	32	48	192	64	
Cores	15	7	8	32	
Core clock	1400	1300	823	925	MHz
Reg File	128	128	256	256	kiB
Lmem/core	64	64	64	64	kiB
Lmem BW/core	64	64	128	128	B/clock
GMem Bus	3				
GMem Clock	3				

What are the main limits for programs?

What happens if you exceed them?

David Kanter / Realworldte

Occupancy calculator

Performance in three sentences

Flops are cheap
Bandwidth is money
Latency is physics

[M. Hoemmen]

Outline

GPU performance

- Understanding GPUs
- GPUs and Memory
- Summary

MPI performance

Parallel Patterns

Parallel Memories: Different Approaches

Problem

Digital memories have only one data 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?

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

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

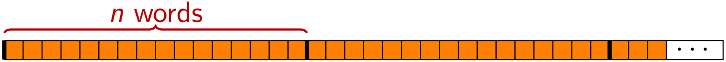
- Split a really wide data bus, but have only one address bus
- Have many "small memories" ("*banks*") with separate address busses. Pick bank by LSB of address.

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.

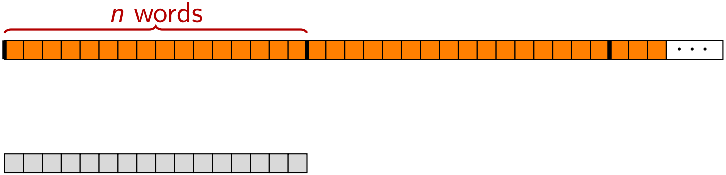


Global Memory

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.

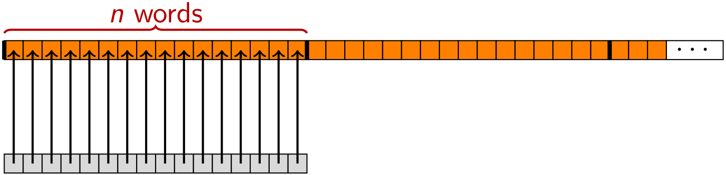


Global Memory

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.



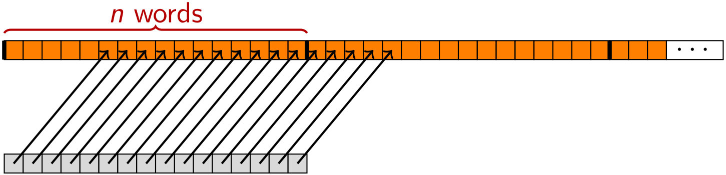
OK: `global_variable[get_global_id(0)]`
(Single transaction)

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.



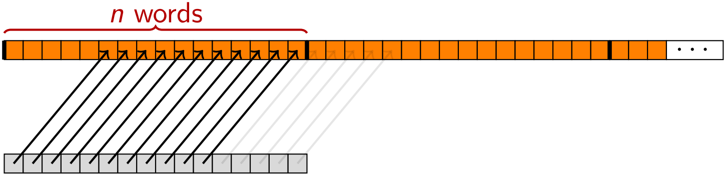
Bad: `global_variable[5+get_global_id(0)]`
(Two transactions)

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.

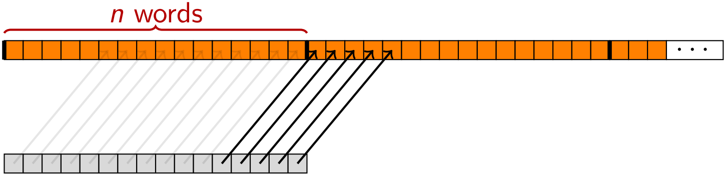


Global Memory

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.

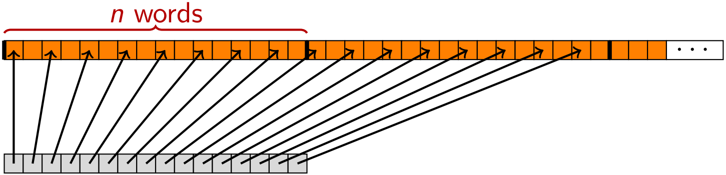


Global Memory

Rule of thumb

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

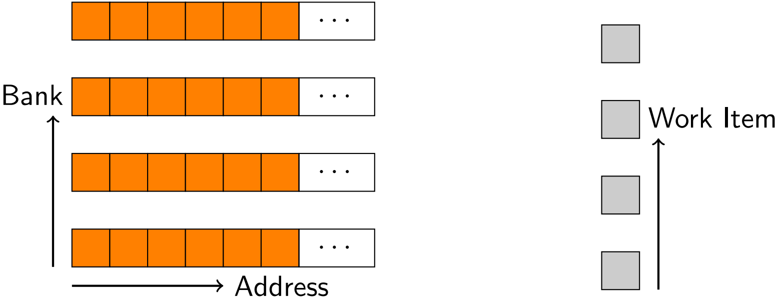
work items access global memory simultaneously. Full utilization only if all bits in bus transaction are useful.



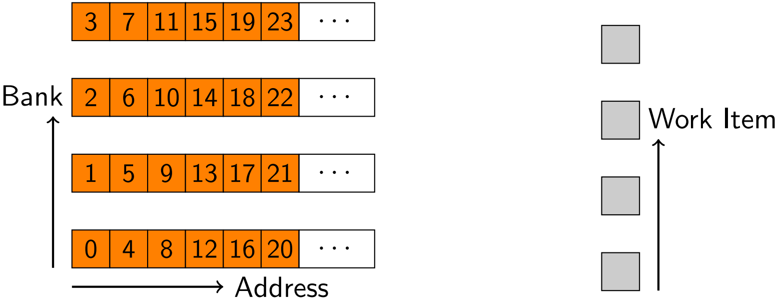
Bad: `global_variable[2*get_global_id(0)]`
(Two transactions)

GPU global access patterns demo

Local Memory: Banking

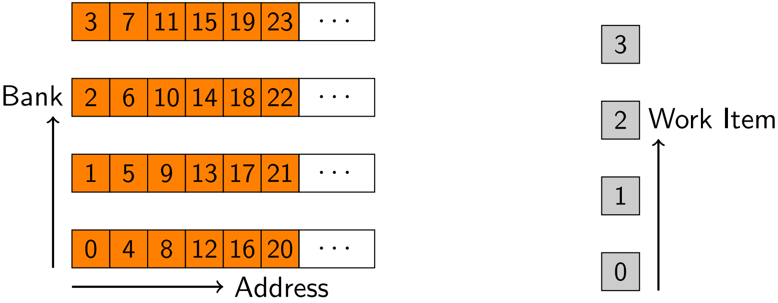


Local Memory: Banking



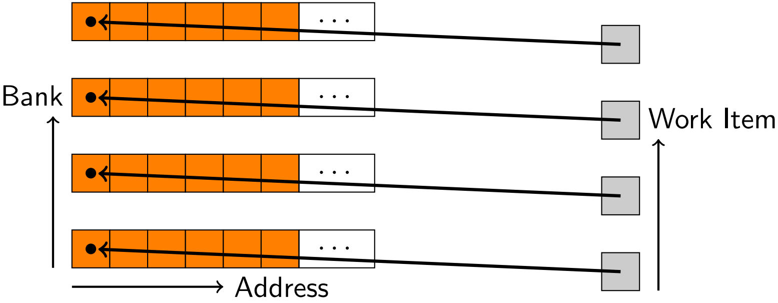
3	7	11	15	19	23	· · ·
2	6	10	14	18	22	· · ·
1	5	9	13	17	21	· · ·
0	4	8	12	16	20	· · ·

Local Memory: Banking



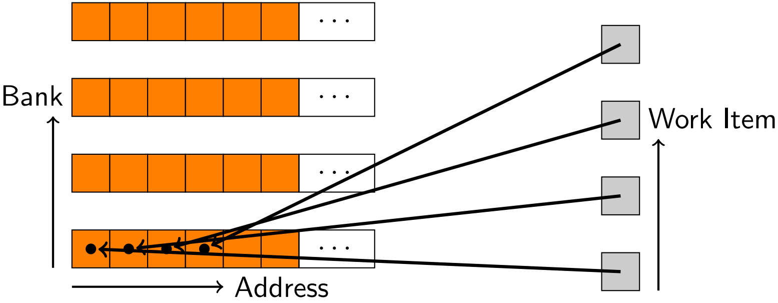
3	7	11	15	19	23	· · ·
2	6	10	14	18	22	· · ·
1	5	9	13	17	21	· · ·
0	4	8	12	16	20	· · ·

Local Memory: Banking



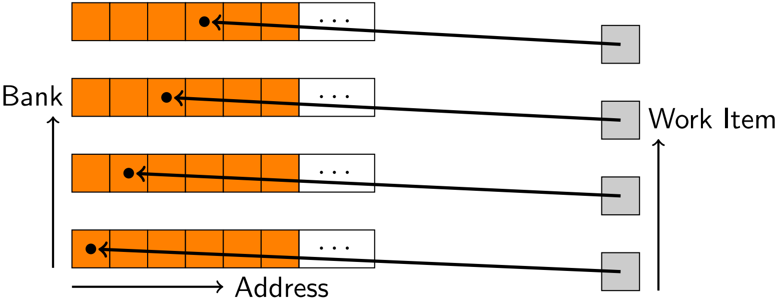
OK: `local_variable[get_local_id(0)]`,
(Single cycle)

Local Memory: Banking



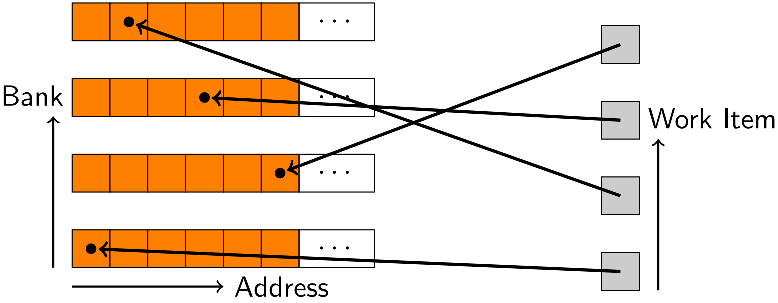
<span>Bad:</span> `local_variable[BANK_COUNT*get_local_id(0)]`
(`BANK_COUNT` cycles)

Local Memory: Banking



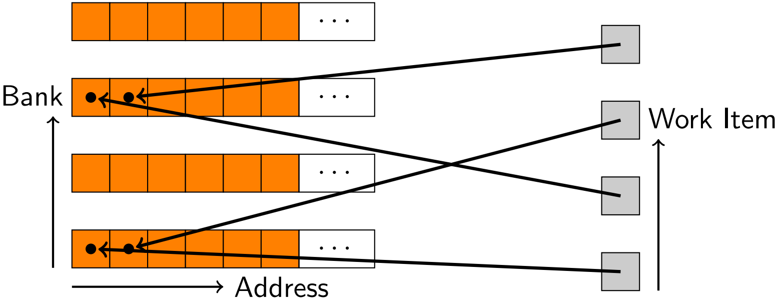
OK: `local_variable[(BANK_COUNT+1)*get_local_id(0)]`
(Single cycle)

Local Memory: Banking



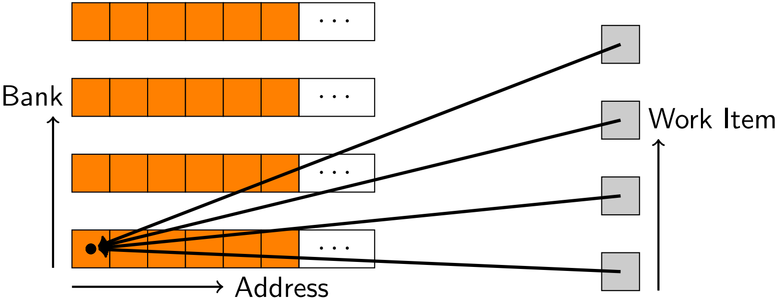
OK: `local_variable[ODD_NUMBER*get_local_id(0)]`
(Single cycle)

Local Memory: Banking



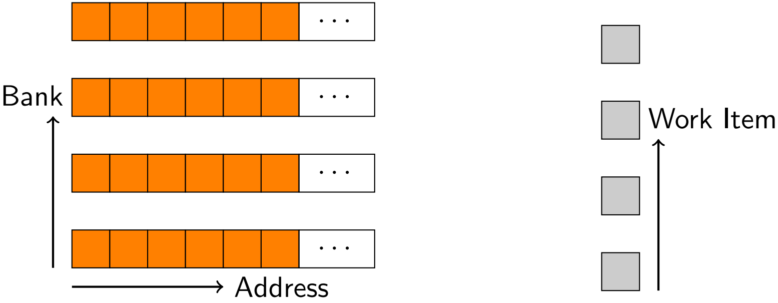
Bad: `local_variable[2*get_local_id(0)]`
(`BANK_COUNT`/2 cycles)

Local Memory: Banking



OK: `local_variable[f(get_group_id(0))]`
(Broadcast–single cycle)

Local Memory: Banking



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?)

GPU local Memory

GPU local access patterns demo

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

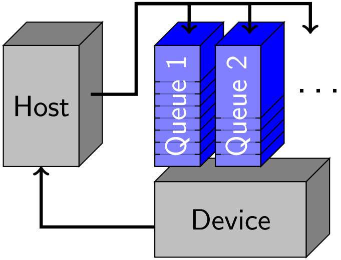
What does this mean for `doubles` in local memory?

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?

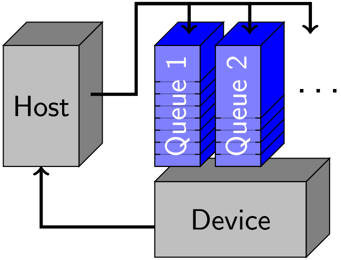


Faster transfers Host ↔ GPU

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?



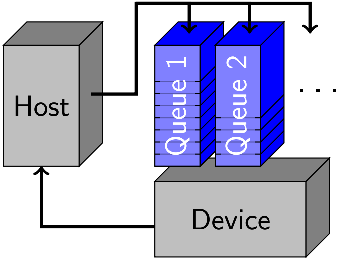
Faster transfers Host ↔ GPU

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)



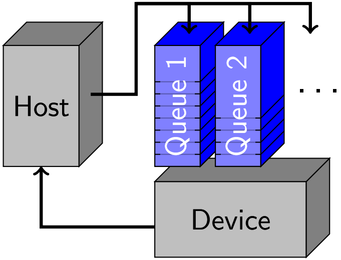
Faster transfers Host ↔ GPU

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: ←, → 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

Type	**Per**	**Access**	**Latency**	
private	work item	R/W	1 or 1000	
local	group	R/W	2	
global	grid	R/W	1000	Cached?
`constant`	grid	R/O	1-1000	Cached
image*n*d_t	grid	R(/W)	1000	Spatially cached

Entertainment: GPU Memory Zoo

Type	**Per**	**Access**	**Latency**	
private	work item	R/W	1 or 1000	
local	group	R/W	2	
global	grid	R/W	1000	Cached?
`constant`	grid	R/O	1-1000	Cached
image*n*d_t	grid	R(/W)	1000	Spatially cached

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

Demo

GPU profiler demo

Outline

GPU performance

MPI performance

Parallel Patterns

MPI performance demo

</2>

Understanding Computational Cost

< 3 >

Concepts, Patterns and Recipes

Outline

GPU performance

MPI performance

Parallel Patterns

- Embarrassingly Parallel
- Partition

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**.

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*)

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.

where $i \in \{1$

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*)

Embarrassingly Parallel

$$y_i = f_i(x_i)$$

where $i \in \{1, \dots, N\}$.

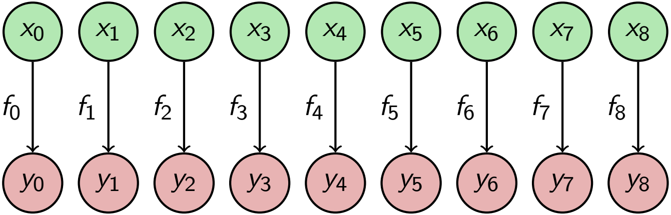
Notation: (also for rest of this lecture)

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

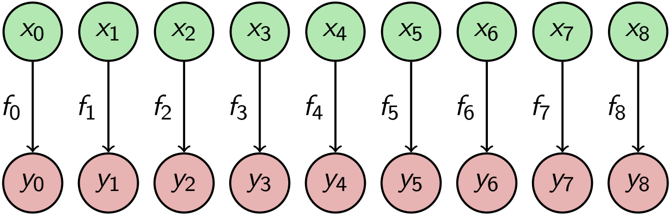
Often: $f_1 = \cdots = f_N$. Then

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

Embarrassingly Parallel: Graph Representation



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?

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

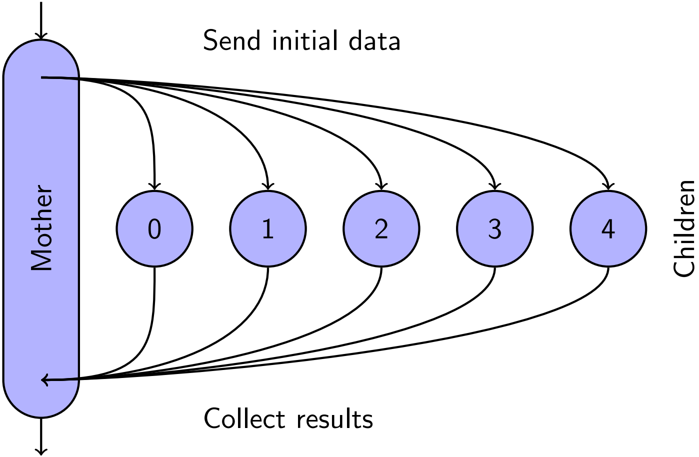
Embarrassingly Parallel: Issues

- 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:



(formerly called “Master-Slave”)

Outline

GPU performance

MPI performance

Parallel Patterns

- Embarrassingly Parallel
- Partition

Partition

$$y_i = f_i(x_{i-1}, x_i, x_{i+1})$$

where $i \in \{1, \ldots, N\}$.

Partition

$$y_i = f_i(x_{i-1}, x_i, x_{i+1})$$

where $i \in \{1, \ldots, N\}$.

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

Partition

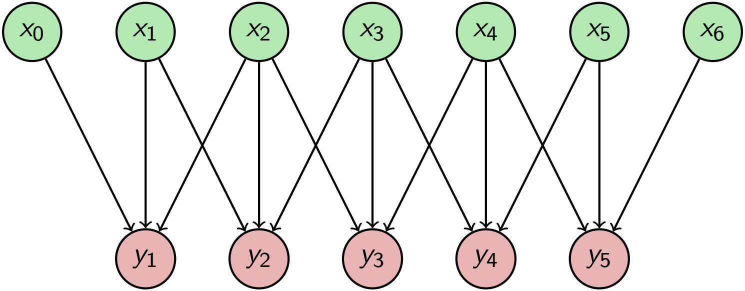
$$y_i = f_i(x_{i-1}, x_i, x_{i+1})$$

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



Mapping to Mechanisms

- OpenMP?

Mapping to Mechanisms

- OpenMP?
- MPI?

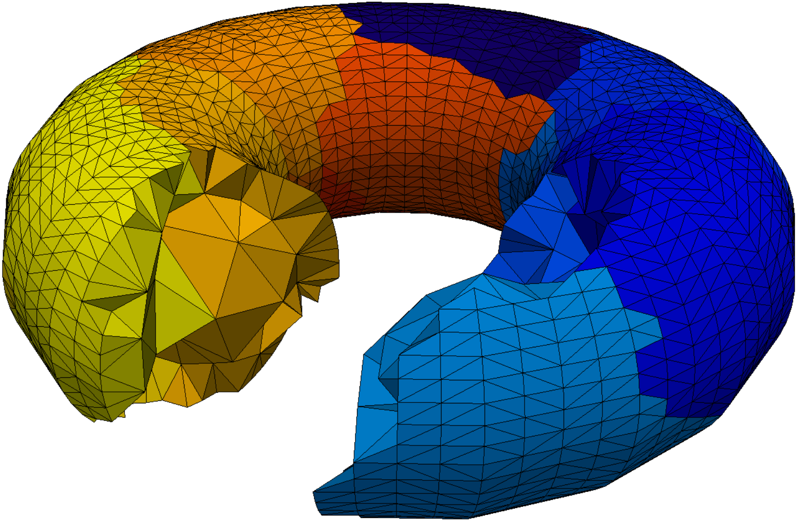
Mapping to Mechanisms

- OpenMP?
- MPI?
- MPI: Larger than # ranks?

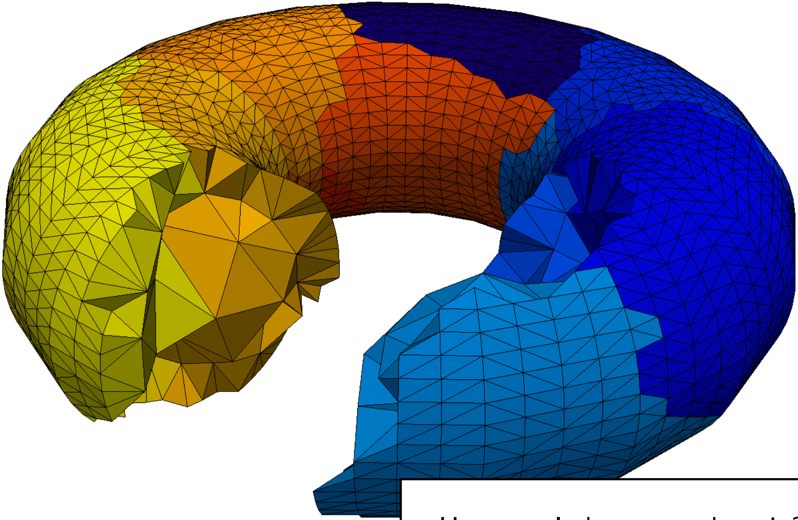
Mapping to Mechanisms

- 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