グリッドコンピューティング

2011/10/17

福田圭祐 (11M37264 松岡研究室)

紹介論文:
"An Execution Strategy and Optimized Runtime Support for Parallelizing Irregular Reductions on Modern GPUs"

著者:Xing Hue, Vignette T. Ravi, Wending Ma and Gagman Angara

(Ohio State University)

ICS'11

発表者:福田圭祐

Abstract

- Strategy & runtime support for reduction problems on **unstructured grid** (fluid dynamics & molecular dynamics) on NVIDIA GPUs
- Based on **mesh partitioning** in reduction space to achieve effective use of GPU's **shared memory**.
- Achieved up to **11.6x** speedup compared to serial CPU execution

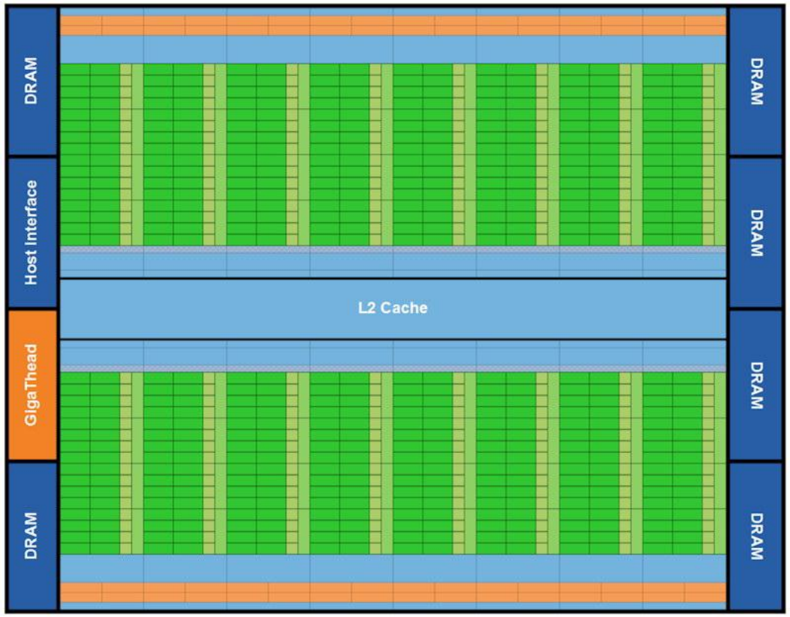
Agenda

- Overview of GPU's architecture and CUDA programing model
- Unstructured grid and Motif
- Background
- Execution Strategy
- Runtime Support
- Evaluation
- Conclusion
- Discussion
- Reference

Agenda

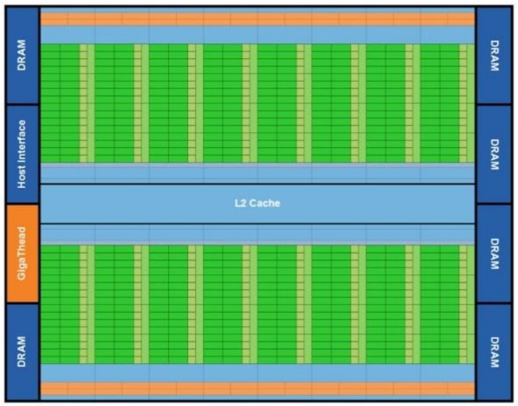
- **Overview of GPU's architecture and CUDA programing model**
- Unstructured grid and Motif
- Background
- Execution Strategy
- Runtime Support
- Evaluation
- Conclusion
- Discussion
- Reference

Overview of GPU's architecture



http://pc.watch.impress.co.jp/docs/news/event/20091001_318660.html

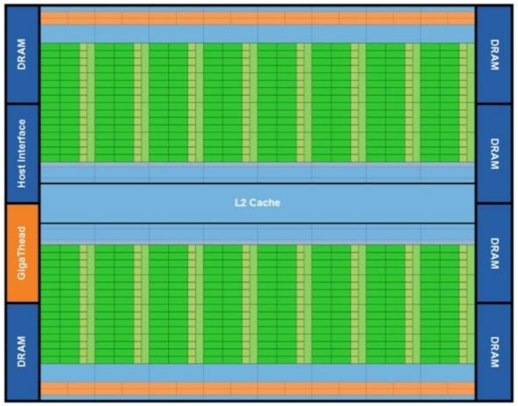
Overview of GPU architecture



- 16(14) SMs (Streaming Multiprocessors)
- 32 CUDA cores / SM
- 512(448) CUDA cores per socket
- 144GBytes/sec memory bandwidth
  (CPU's bandwidth : ex. 36GBytes/s)
- L1 cache & shared memory / SM
- (16kB/48kB configurable)
- Connected to a host through
  PCI express bus

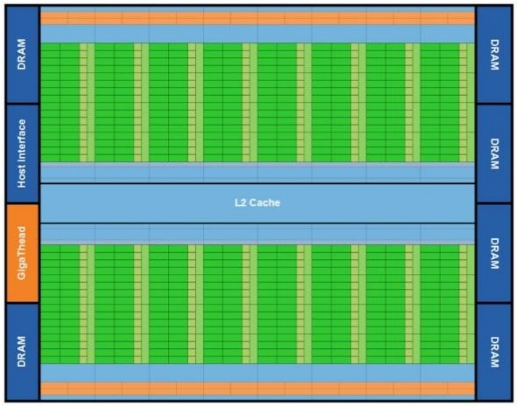
- SIMD execution in a single "Warp"

Advantages of GPU architectures



- High memory bandwidth
- Fast context switching
  (hardware thread management)
- Execute large number of threads
  (～ tens of thousands)
- Hiding memory latency
  (switch thread context when a warp is waiting for memory access)

Difficulty of GPU programming



- Needs massive parallelism

  "GPU-friendly" algorithm is required

- Irregular execution path
- Irregular memory access
- Efficient use of fast memory (shared memory)
- Relatively small memory (ex. 3GB / 6GB)
- No global synchronization

CUDA programming model

- Developed by nVidia
- Extension of C++ language & library functions
- Hierarchical grouping of threads (Grid & Thread Block)

    Grid : 2-dim, Thread Block : 3-dim

- 16kB shared memory per Thread Block
- Synchronization can be done only within a thread block

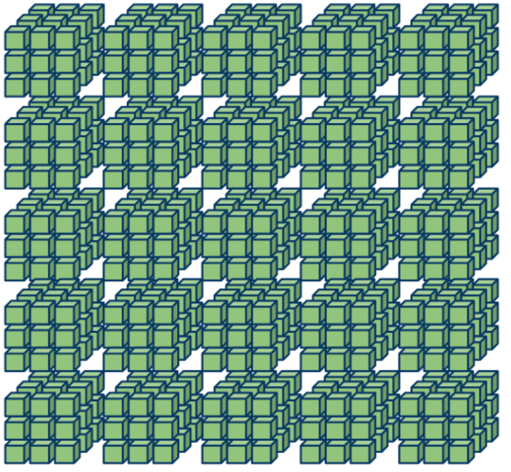


Image from [1]

Agenda

- Overview of GPU's architecture and CUDA programing model
- **Unstructured grid and "Motif"**
- Background
- Execution Strategy
- Runtime Support
- Evaluation
- Conclusion
- Discussion
- Reference

Motif

- A models of typical parallel program structures (formally called “Dwarfs”)
- Types of parallelism and memory access patterns
- Proposed in “Berkley View Report” [2]

Dense Linear Algebra	BLAS level1,2,3 VxV, MxV, MxM
Sparse Linear Algebra	SpMV
Spectral Methods	FFT, All-to-all communication
N-body Methods	$O(N^2)$ calculation
Structured Grids	
Unstructured Grids	Often includes indirect memory reference
Monte Carlo	Repeated random trials

Motif

- A models of typical parallel program structures (formally called “Dwarfs”)
- Types of parallelism and memory access patterns
- Proposed in “Berkley View Report” [2]

Dense Linear Algebra	BLAS level1,2,3 VxV, MxV, MxM
Sparse Linear Algebra	SpMV
Spectral Methods	FFT, All-to-all communication
N-body Methods	$O(N^2)$ calculation
Structured Grids	
Unstructured Grids	Often includes indirect memory reference
Monte Carlo	Repeated random trials

Agenda

- Overview of GPU's architecture and CUDA programing model
- Unstructured grid and "Motif"
- **Background**
- Execution Strategy
- Runtime Support
- Evaluation
- Conclusion
- Discussion
- Reference

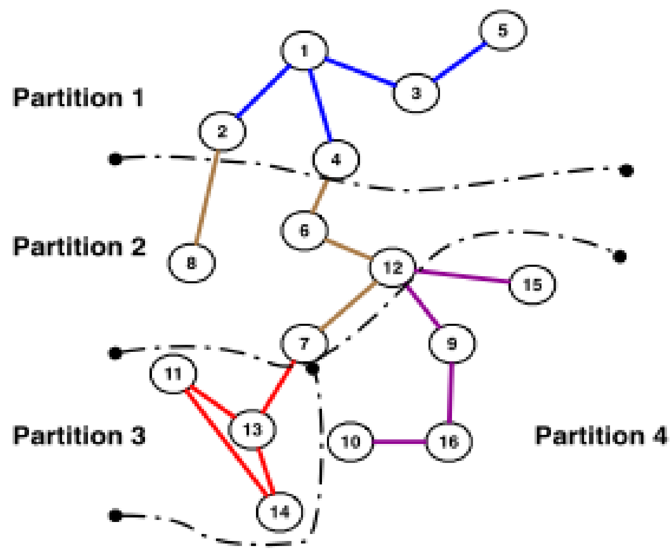
Background: Irregular reduction

- Irregular problems : **unstructured gird**
- **Irregular reduction**

```
Real X(numNodes), Y(numEdges);  ! Data arrays
Integer IA(numEdges,2);          ! Indirection array
Real RA(numNodes);               ! Reduction array

for (i=1; i<numEdges; i++) {
   RA(IA(i,1)) = RA(IA(i,1)) op (Y(i) op X(IA(i,1) op X(IA(i,2))))
   RA(IA(i,2)) = RA(IA(i,2)) op (Y(i) op X(IA(i,1) op X(IA(i,2))));
}
```

Figure 1: Typical Structure of *Irregular Reduction*



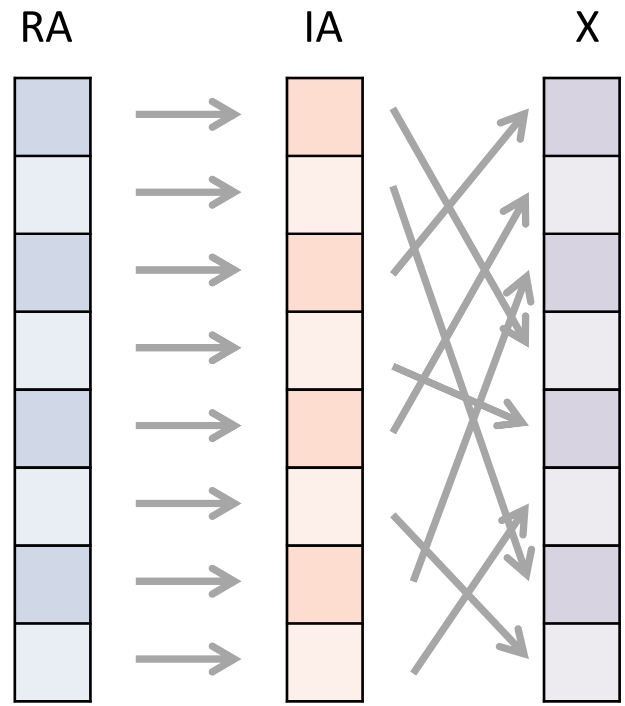
Background: Irregular reduction

- Irregular problems : **unstructured gird**
- **Irregular reduction**

```
Real X(numNodes), Y(numEdges);  ! Data arrays
Integer IA(numEdges,2);         ! Indirection array
Real RA(numNodes);              ! Reduction array

for (i=1; i<numEdges; i++) {
   RA(IA(i,1)) = RA(IA(i,1)) op (Y(i) op X(IA(i,1) op X(IA(i,2))))
   RA(IA(i,2)) = RA(IA(i,2)) op (Y(i) op X(IA(i,1) op X(IA(i,2))));
}
```

Figure 1: Typical Structure of *Irregular Reduction*



Background: Irregular reduction

- Irregular problems : **unstructured gird**
- **Irregular reduction**

Computation Space : Read-only
X : accessed via Indirect array "IA"
Y : accessed via loop counter "i"

```
Real X(numNodes), Y(numEdges);  ! Data arrays
Integer IA(numEdges,2);         ! Indirection array
Real RA(numNodes);              ! Reduction array

for (i=1; i<numEdges; i++) {
   RA(IA(i,1)) = RA(IA(i,1)) op (Y(i) op X(IA(i,1) op X(IA(i,2))))
   RA(IA(i,2)) = RA(IA(i,2)) op (Y(i) op X(IA(i,1) op X(IA(i,2))));
}
```

Figure 1: Typical Structure of *Irregular Reduction*

Reduction Space : Output

Background: Irregular reduction

- Irregular problems : **unstructured gird**
- **Irregular reduction**

Computation Space : Read-only
X : accessed via Indirect array "IA"
Y : accessed via loop counter "i"

```
Real X(numNodes), Y(numEdges);  ! Data arrays
Integer IA(numEdges,2);         ! Indirection array
Real RA(numNodes);              ! Reduction array

for (i=1; i<numEdges; i++) {
   RA(IA(i,1)) = RA(IA(i,1)) op (Y(i) op X(IA(i,1) op X(IA(i,2))))
   RA(IA(i,2)) = RA(IA(i,2)) op (Y(i) op X(IA(i,1) op X(IA(i,2))));
}
```

Figure 1: Typical Structure of *Irregular Reduction*

Reduction Space : Output

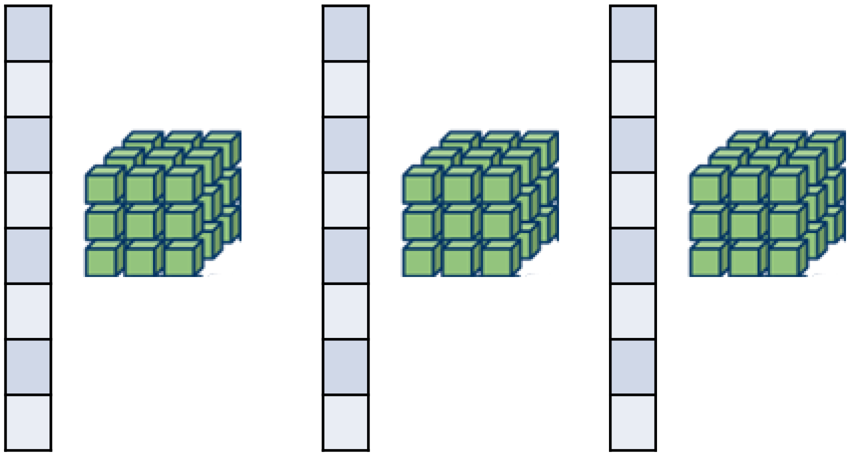
Challenges:
Datasets are typically extremely large,
And global memory is very slow
→ Needs to utilize shared memory

Agenda

- Overview of GPU's architecture and CUDA programing model
- Unstructured grid and "Motif"
- Background
- **Execution Strategy**
- Runtime Support
- Evaluation
- Conclusion
- Discussion
- Reference

Naïve use of shared memory

- Copy the reduction array for each thread block



- Summation of reduction array is required
- What if RA is larger than shared memory (16KB) ?
- Significant memory overhead

Solution:
Partitioning-based Locking Scheme

- **Partition the reduction space** such that each portion fits shared memory
- Reorder the RA and X to achieve **coalescing access** to global memory
- Completely eliminate the requirement for RA array reduction

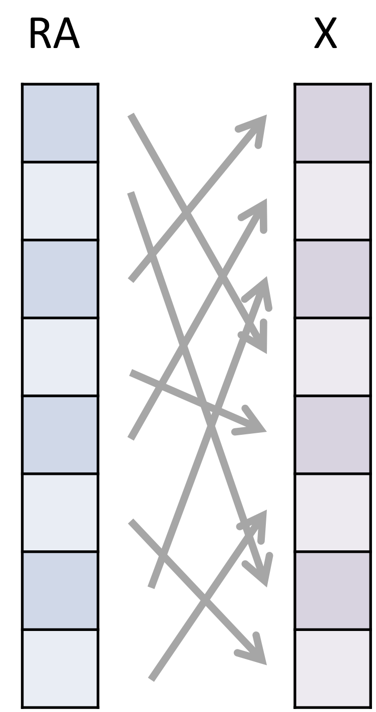
Two strategies for partitioning

- (1) Computation Space-based Partitioning

```
Real X(numNodes), Y(numEdges);  ! Data arrays
Integer IA(numEdges,2);         ! Indirection array
Real RA(numNodes);              ! Reduction array

for (i=1; i<numEdges; i++) {
   RA(IA(i,1)) = RA(IA(i,1)) op (Y(i) op X(IA(i,1) op X(IA(i,2))))
   RA(IA(i,2)) = RA(IA(i,2)) op (Y(i) op X(IA(i,1) op X(IA(i,2))));
}
```

Figure 1: Typical Structure of *Irregular Reduction*



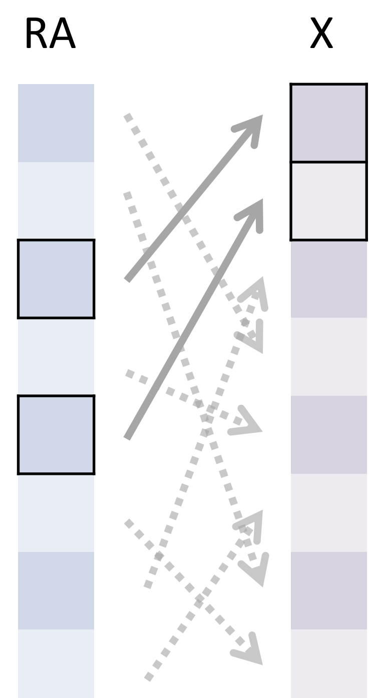
Two strategies for partitioning

- (1) Computation Space-based Partitioning

```
Real X(numNodes), Y(numEdges);   ! Data arrays
Integer IA(numEdges,2);          ! Indirection array
Real RA(numNodes);               ! Reduction array

for (i=1; i<numEdges; i++) {
   RA(IA(i,1)) = RA(IA(i,1)) op (Y(i) op X(IA(i,1) op X(IA(i,2))))
   RA(IA(i,2)) = RA(IA(i,2)) op (Y(i) op X(IA(i,1) op X(IA(i,2))));
}
```

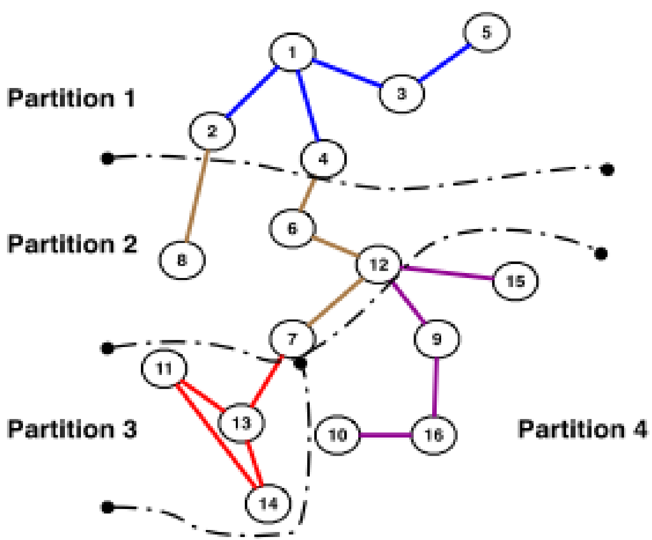
Figure 1: Typical Structure of *Irregular Reduction*



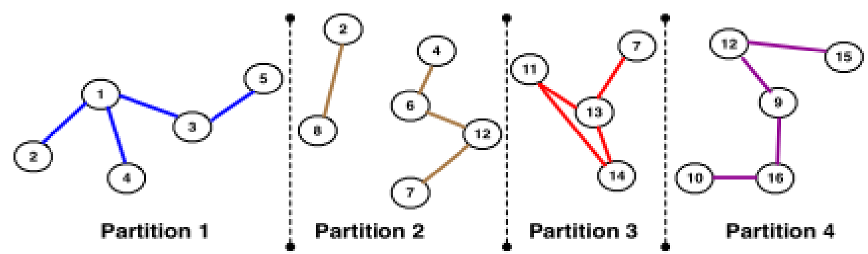
  - A reduction point can belong to several partitions
  - A # of reduction points that corresponds to a partition varies,
    → difficult to optimally use shared memory

Two strategies for partitioning

- (1) Computation Space-based Partitioning



(a) Partitioning on Computation Space



(b) Reduction Size Increase in Each Partition

Figure 2: Computation Space partitioning and reduction size in each partition

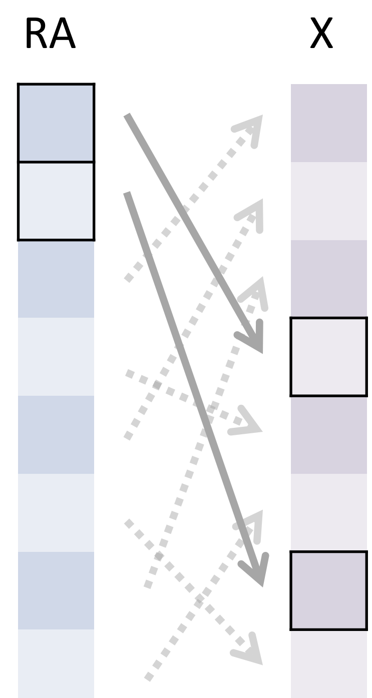
Two strategies for partitioning

- (2) Reduction Space-based Partitioning

```
Real X(numNodes), Y(numEdges);  ! Data arrays
Integer IA(numEdges,2);         ! Indirection array
Real RA(numNodes);              ! Reduction array

for (i=1; i<numEdges; i++) {
  RA(IA(i,1)) = RA(IA(i,1)) op (Y(i) op X(IA(i,1) op X(IA(i,2))))
  RA(IA(i,2)) = RA(IA(i,2)) op (Y(i) op X(IA(i,1) op X(IA(i,2))));
}
```

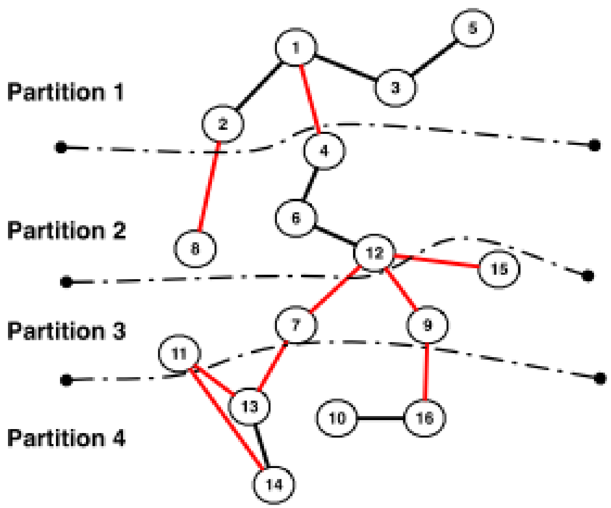
Figure 1: Typical Structure of *Irregular Reduction*



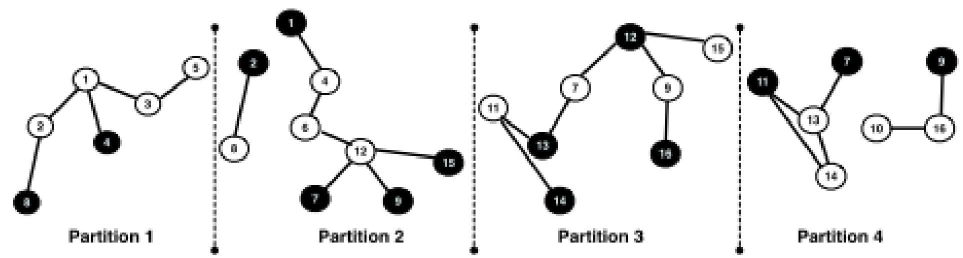
  - Only one copy of the reduction array
  - More edges than method (1)
  - → **We employ this method here**

Two strategies for partitioning

- (2) Reduction Space-based Partitioning



(a) Partitioning on Reduction Space



(b) Workload Increase in Each Partition

Figure 3: Reduction Space partitioning and computation size in each partition

  - → **We employ this method here**
  - Computation size increases, but memory is more precious

Agenda

- Overview of GPU's architecture and CUDA programing model
- Unstructured grid and "Motif"
- Background
- Execution Strategy
- **Runtime Support**
- Evaluation
- Conclusion
- Discussion
- Reference

3 Runtime supports for partitioning

- METIS partitioning (MP)
- GPU-based (trivial) partitioning (GP)
- Multi-dimensional partitioning (MD)

3 Runtime supports for partitioning

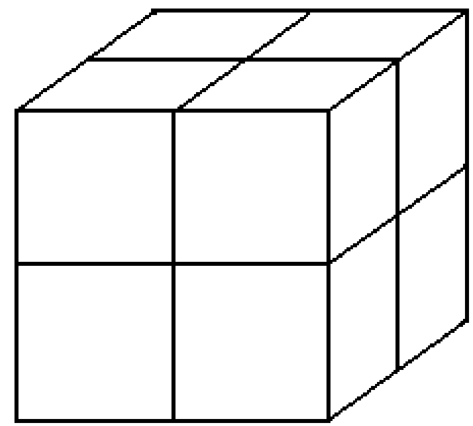
- METIS partitioning (MP)
  - Widely used partitioner for graphs and finite element meshes
  - Executes serially on a CPU
  - Initialization cost is very high

3 Runtime supports for partitioning

- GPU-based (trivial) partitioning (GP)
  - Very simple and implemented on a GPU
  - Divide reduction space simply on the order of inputs
  - Has a significantly larger number of edges
  - O(n) , n = # of particles

3 Runtime supports for partitioning

- Multi-dimensional partitioning (MD)
  - Based on node coordinates and finding the k-th smallest value



  - O(np) , p = # of partitions
  - Practically the # of particles is much smaller than # of particles
  - Implemented on CPU

Agenda

- Overview of GPU's architecture and CUDA programing model
- Unstructured grid and "Motif"
- Background
- Execution Strategy
- Runtime Support
- **Evaluation**
- Conclusion
- Discussion
- Reference

Evaluation settings & applications

- Evaluation settings
  - CPU : Xeon E5520, 48GB memory
  - GPU : NVIDIA C2050, 2.68GB memory

- Applications
  - Fluid dynamics application “Euler”
    - 20,000 nodes, 120,000 edges, 12,000 faces
    - 50,000 nodes, 300,000 edges, 29,000 faces
    - 10,000 time step iterations
  - Molecular Dynamics application
    - 37,000 molecules, 4,600,000 interactions
    - 131,000 molecules, 16,200,000 interactions
    - 100 time step
    - In adaptive version, indirection array is modified every 20 interactions

Comparison between partitioning schemes

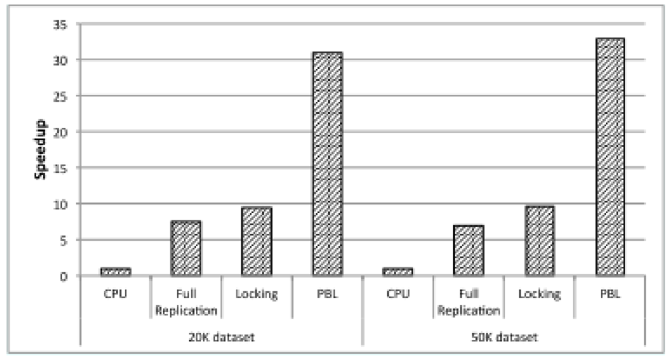


Figure 6: Euler: Comparison of PBL Scheme Over Conventional Strategies and Sequential CPU Execution

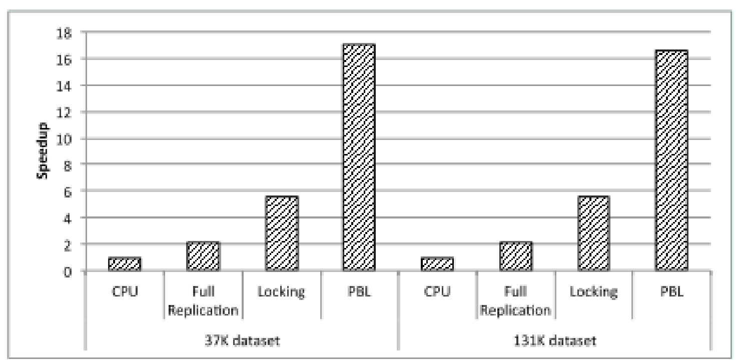


Figure 7: Molecular Dynamics: Comparison of PBL Over Conventional Strategies and Sequential CPU Execution

Impact of number of partitions on partitioning efficiency

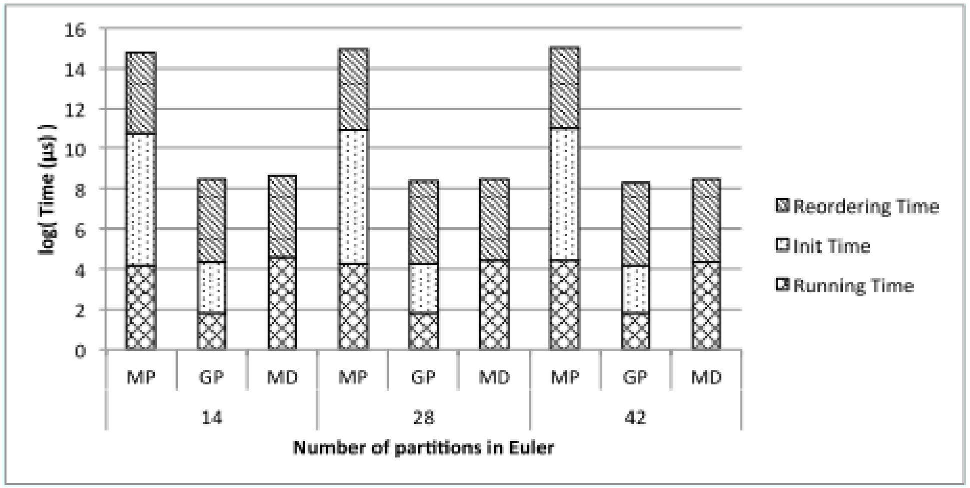


Figure 8: Cost Components of Partitioners (Euler)

- "GP has the shortest running time, across varying number of partitions"
- "MP is around 2.8 times faster than MD when 14 partitions are desired. However, MP increases sharply with the increasing number of partitions"
- "MD is not influenced by the number of partitions significantly"

Computation time components

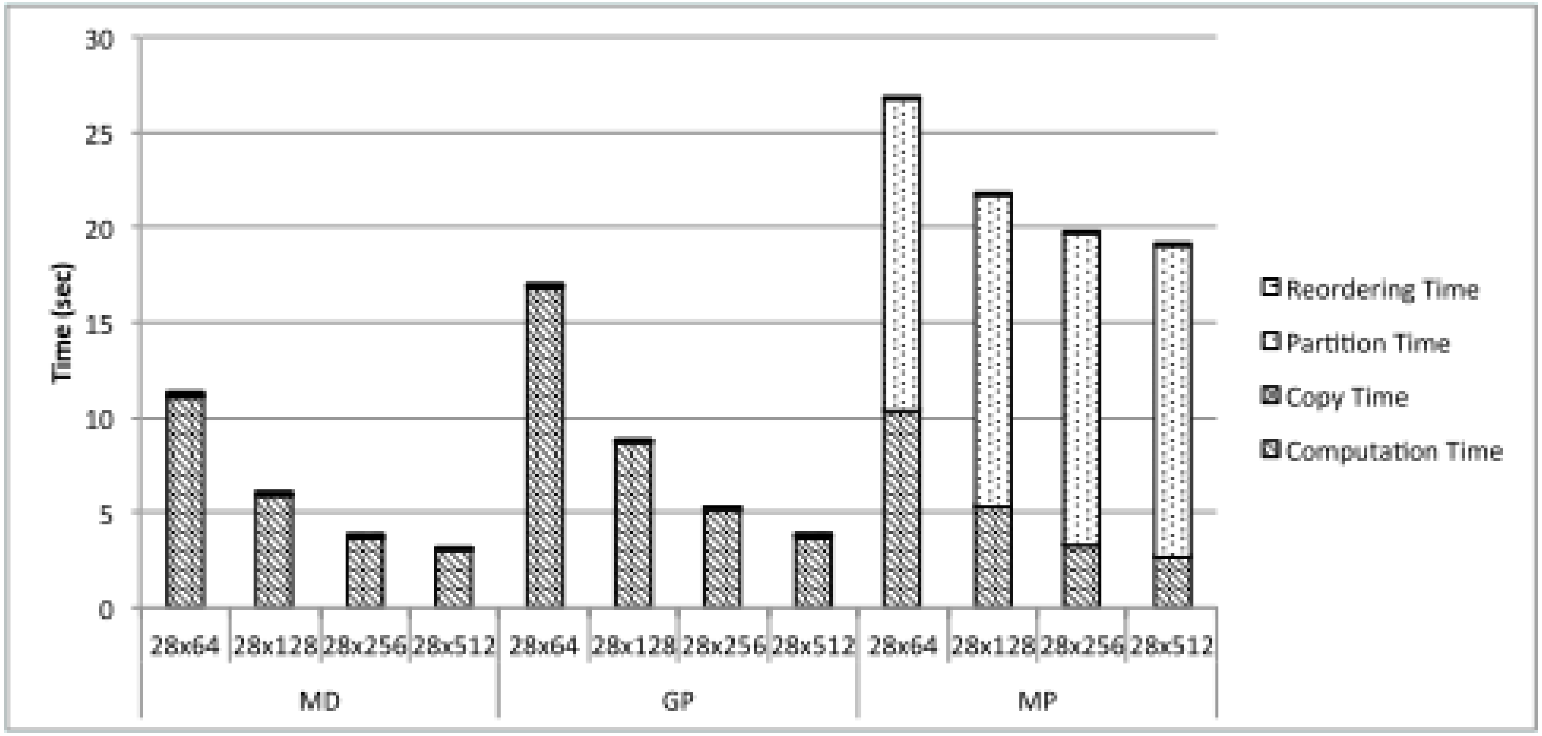


Figure 11: Comparison of Metis, GPU and Multi-dimensional using 28 Partitions for Euler (20K)

Impact of shared memory preference

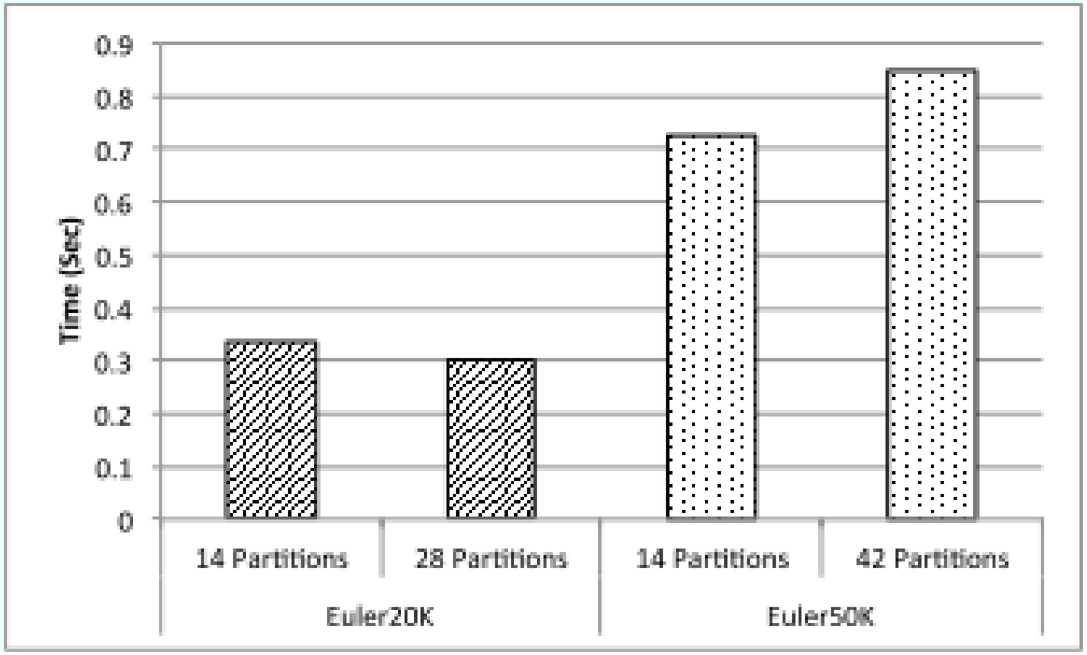


Figure 13: Shared Memory Preferred (14 Partitions) Vs. Cache Preferred (28 Partitions) - (left) Shared Memory Preferred (14 Partitions) Vs. Cache Preferred (42 Partitions) - (right)

Comparison of MP,GP,MD (adaptive)

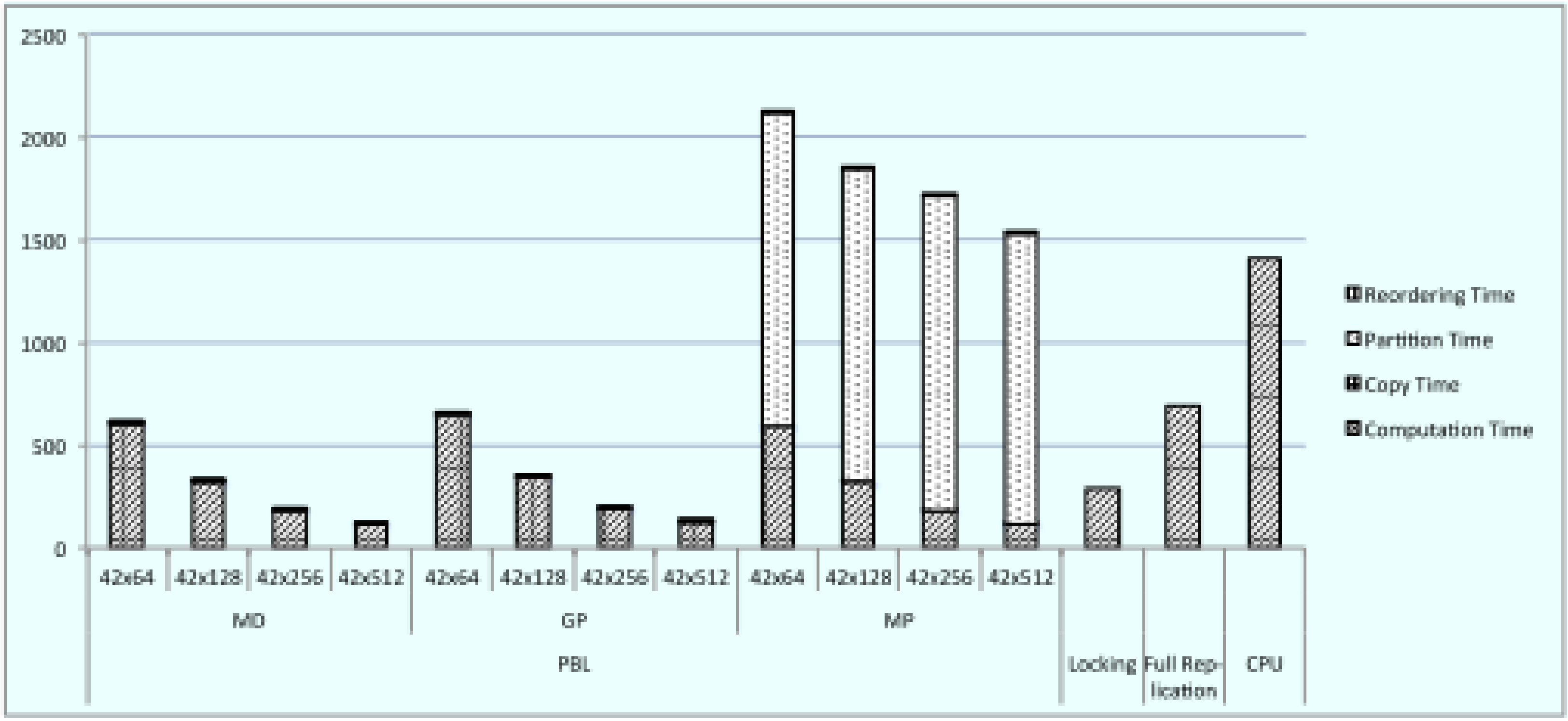


Figure 14: Comparison of MP, GP, and MD for Adaptive Molecular Dynamics (37K dataset, 42 partitions)

Agenda

- Overview of GPU's architecture and CUDA programing model
- Unstructured grid and "Motif"
- Background
- Execution Strategy
- Runtime Support
- Evaluation
- **Conclusion**
- Discussion
- Reference

Conclusion

- To execute irregular reduction problem on GPU, shared-memory-aware partitioning scheme is proposed
- For efficient mesh partitioning, METIS partitioning, GPU simple partitioning and multi dimension partitioning are compared.
- METIS achieves good partitioning, but has significant initialization overhead.
- GPU simple partitioning has the shortest partitioning time, but the partitioning quality is low.
- Multi dimension partitioning achieves good time-quality balance and best overall performance.

Agenda

- Overview of GPU's architecture and CUDA programing model
- Unstructured grid and "Motif"
- Background
- Execution Strategy
- Runtime Support
- Evaluation
- Conclusion
- **Discussion**
- Reference

Discussion

- Other multi-threaded parallel graph partitioning software ?
  - (ParMETIS is MPI-based parallelization)
  - SCOTCH[3], PARTY[4], Chaco[5], JOSTLE[6],…
- GPU ‘trivial’ partitioning can be faster ?
  - Partitioning times of GP(on GPU) and MP(on CPU) are nearly equal
  - GP seems to have room to further optimization
  - Any challenge on implementing GP on GPU?
- What about if the total amount of data > 3GB ?
  - “Partition-aware” data management is required ?

Agenda

- Overview of GPU's architecture and CUDA programing model
- Unstructured grid and "Motif"
- Background
- Execution Strategy
- Runtime Support
- Evaluation
- Conclusion
- **Reference**

References

- [1] http://www.resultsovercoffee.com/2011/02/cuda-blocks-and-grids.html
- [2] "The Landscape of Parallel Computing Research: A View from Berkeley" Electrical Engineering and Computer Sciences University of California at Berkeley, Technical Report No. UCB/EECS-2006-183
- [3] SCOTCH http://www.labri.fr/perso/pelegrin/scotch
- [4] PARTY http://www2.cs.uni-paderborn.de/cs/ag-monien/PERSONAL/ROBSY/party.html
- [5] Chaco http://www.sandia.gov/~bahendr/chaco.html
- [6] JOSTLE http://www.cs.sunysb.edu/~algorithm/implement/jostle/implement.shtml

Thank you.