An Accelerated Procedure for Hypergraph Coarsening on the GPU

Lin Cheng, Hyunsu Cho, and Peter Yoon
Trinity College
Hartford, CT, USA
Outline

• Hypergraph coarsening
• Implementation challenges
• Runtime task planning
• Results
Hypergraph

- Nodes
- **Hyperedges** (nets)
  - Subsets of nodes
Hypergraph

• Hypergraph partitioning
  • Minimize **edge cut**
  • Balance constraint
• NP-complete
Hypergraph coarsening

• Heuristic: reduce # nodes by fusing

6 nodes
Hypergraph coarsening

- Heuristic: reduce # nodes by fusing
Mondriaan algorithm

• Given a node, find most “similar” neighbor
• Similarity = # hyperedges containing both

\[ \text{Affinity} = 2 \]
Mondriaan algorithm

• Given a node, find most “similar” neighbor
• Similarity = # hyperedges containing both

\[
v_0 \ v_1 \ v_2 \ v_3 \ v_4 \ v_5 \ v_6 \ v_7 \quad \leftarrow \text{Nodes}
\]

\[
\begin{array}{cccccccc}
  e_0 & & & & & & & \\
  & e_1 & & & & & & \\
  & & e_2 & & & & & \\
  & & & e_3 & & & & \\
  & & & & e_4 & & & \\
  & & & & & e_5 & & \\
  & & & & & & e_6 & \\
  & & & & & & & e_7
\end{array}
\]

Binary sparse matrix

Member? (T/F)
Mondriaan algorithm

• Given a node, find most “similar” neighbor
• Similarity = # hyperedges containing both

1. Find edges containing $v_3$
Mondriaan algorithm

- Given a node, find most "similar" neighbor
- Similarity = # hyperedges containing both

1. Find edges containing $v_3$
2. Collect nonzeros
Mondriaan algorithm

- Given a node, find most “similar” neighbor
- Similarity = # hyperedges containing both

3. Inspect column index of each nonzero
Mondriaan algorithm

• Given a node, find most “similar” neighbor
• Similarity = # hyperedges containing both

![Diagram showing Mondriaan algorithm](image)

3. Inspect column index of each nonzero

**Similarity scores**
Mondriaan algorithm

• Parallel algorithm:
  Inspect edges in parallel
GPU as parallel accelerator

- General-purpose computation
- Massively parallel; many-core
GPU as parallel accelerator

- **SIMD** (Single-Instruction-Multiple-Data)
- NVIDIA GPUs: organized in **warps**
  32 threads share one instruction counter

\[
\text{LOAD A[*] INTO X}
\]

\[
\begin{array}{cccccccc}
\text{X} & & & & & & & \\
\uparrow & & & & & & & \\
\text{A[*]} & & & & & & & \\
\end{array}
\]

X (register)
A[*] (memory)
Warp divergence

Warp divergence

Thread 5-9: NONE

X

A[*]
Warp divergence

Thread 5-9: NONE

stalled
Warp divergence

- Serializes execution
- Caused by **load imbalance**
  - Sparse/irregular data
Mondriaan algorithm

- A naïve strategy results in load imbalance
SHFL to the rescue

- Compiler primitive
- Shuffles content of adjacent registers

\[
x = \text{shfl\_down}(x, 2)
\]
SHFL to the rescue

- Compiler primitive
- Shuffles content of adjacent registers
- Single machine instruction
- Warp-synchronous; no sync. needed after

```
x = shfl_down(x, 2)
```
SHFL to the rescue

• Compiler primitive
• Shuffles content of adjacent registers
• Single machine instruction
• Warp-synchronous; no sync. needed after

• CUB Library: http://nvlabs.github.io/cub/
  warp-wide reduction, prefix sum
Runtime planning with SHFL

Thread 0
- $e_0$
- $e_1$
- $e_3$

Thread 1
- $e_5$

Thread 2
- $e_7$

Nodes: $v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7$

Values: 6, 3, 2
Runtime planning with SHFL

Thread 0
- $e_0$
- $e_1$
- $e_3$

Thread 1
- $e_5$

Thread 2
- $e_7$

Prefix sum:
- 0
- 3
- 6
- 2
- 9
Runtime planning with SHFL

Thread 0

Thread 1

Thread 2

Prefix sum

\[ \text{ceil}(11 / 3) = 4 \]
Runtime planning with SHFL

\[ \text{ceil}(11 / 3) = 4 \]
Runtime planning with SHFL

Range

0 – 3
4 – 7
8 – 10

Equal # of nonzeros
Runtime planning with SHFL

Thread 0

Thread 1

Thread 2

Range

0 – 3

4 – 7

8 – 10
Results: long-tailed distribution

Y-axis: # of nonzeros in columns, descending, log-scale

wikipedia
GPU: 50 s
Seq. : 811 s
16 x speed-up

flickr
GPU: 59 s
Seq. : 877 s
15 x speed-up
Results: non-long-tailed distribution

Y-axis: # of nonzeros in columns, descending, log-scale
Analysis of results

• Good speedup for data with long-tailed distribution of nonzeros

• Synthetic data
  • First 1,000 columns : 100,000 nonzeros each
  • Next 699,000 columns : all zero
  • Speedup: 123 x
Analysis of results

• Recall:

```
Thread 0
  e_0
  e_1
  v_2
  v_3
  v_4
Thread 1
  e_3
  v_2
  v_3
  v_4
Thread 2
  e_5
  v_2
  v_3
  v_4
```

What if this column has < 32 nonzeros?
Conclusion

• Novel technique for task planning using SHFL
• Implementation of hypergraph algorithm handling arbitrary connectivity patterns
Acknowledgment

• Trinity College: Student Research Program
• NVIDIA: CUDA Teaching Center