Evaluating Graph Coloring on GPUs

Final Project for the GPU class - Spring 2010
submitted as a poster to PPoPP 2011

Pascal Grosset, Peihong Zhu, Shusen Liu,
Suresh Venkatasubramanian, and Mary Hall

Graph Coloring

- Assignment of colors to vertices of a graph such that connected vertices have different color
  - Planar graphs: 4 colors is enough
  - Non-planar graphs: NP Complete

- Solutions:
  - Brute-force
  - Heuristics

- Use:
  - Assignment of frequencies to wireless access points
  - ...

Existing Algorithms

- Many heuristics exist with different decision criteria
  - First Fit - none
  - LDO - uses degree as decision parameter
  - SDO - uses saturation as decision parameter
  - LDO - uses degree as decision parameter
  - SDO & LDO - uses saturation and then degree

- Trade-offs
  - Speed
    - Fastest: First-Fit
    - Slowest: SDO & LDO
  - Colors
    - Best: SDO & LDO
    - Worst: First-Fit

Existing Parallel Solutions

- We did not find any relevant related works for graph coloring on GPUs
- Main inspiration:
  - Gebremedhin and Manne
  - (G-M) algorithm for shared memory architectures
    - 4 stages:
      - Partitioning
      - Pseudo-coloring
      - Conflict detection
      - Conflict Resolution
Proposed Framework

- Adapt existing framework to GPUs
- Phase 1: Graph Partitioning
  - Decide how the Graph will be partitioned into subgraphs
- Phase 2: Graph Coloring & Conflict Identification
  - Graph coloring using one of the heuristics
    - First Fit, SDO & LDO, Max In, Max Out
  - Conflict Identification
- Phase 3: Sequential Conflict Resolution
  - To definitely remove all conflicts

Max In & Max Out

- Two new heuristics
  - Decision parameter: number of vertices having neighbors outside the subgraph

```c
while Num_Colored < N(Number of vertices in subgraph) do
  max = -1
  for i = 1 to N do
    if !colored(Vi) then
      no = Number of neighbors outside partition
      if no > max then
        max = no
        index = i
      if no == max then
        if d(Vi) > d(Vindex) then
          index = i
        Color Vindex
        Num_Colored ++
```

Phase 2: Graph Coloring & Conflict Identification

Main part

- Transfer data from CPU to GPU
  1. Graph Coloring
    - Run Graph Coloring: 1 thread per subgraph
      - cudaEventSynchronize
  2. Conflicts Indentification
    - sets color of conflicted nodes to 0: 1 thread per node
      - cudaEventSynchronize
- Transfer Conflicts to CPU
- Count Conflicts
  - If conflicts < threshold
    - exit
  - Else
    - Repeat from 1

Data Storage

- Adjacency Matrix (Initial)
  - Too big
- Adjacency List
  - Very compact
  - Bad memory access pattern
    - bad performance
- "Sort of" Adjacency List
  - Size of each list: max degree
  - Good balance between performance and size
    - can still be optimized
Phase 2: Graph Coloring & Conflict Identification

Main part

- Transfer data from CPU to GPU

1. Graph Coloring
   - Run Graph Coloring: 1 thread per subgraph
   - cudaEventSynchronize

2. Conflicts Identification
   - Set color of conflicted nodes to 0: 1 thread per node
   - cudaEventSynchronize

- Transfer Conflicts to CPU

- Count Conflicts
  - If conflicts < threshold
    - Exit
  - Else
    - Repeat from 1

Test Data

- Data source: University of Florida Sparse Matrix Collection

<table>
<thead>
<tr>
<th>Name</th>
<th>n</th>
<th>m</th>
<th>Density</th>
<th>Sparsity</th>
<th>Avg. Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>cscid</td>
<td>52,139</td>
<td>1,375,586</td>
<td>0.000086</td>
<td>206</td>
<td>50</td>
</tr>
<tr>
<td>acsrdb</td>
<td>54,670</td>
<td>1,361,697</td>
<td>0.000099</td>
<td>270</td>
<td>47</td>
</tr>
<tr>
<td>acsrdb</td>
<td>217,018</td>
<td>5,830,171</td>
<td>0.00025</td>
<td>179</td>
<td>52</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>n</th>
<th>m</th>
<th>Density</th>
<th>Sparsity</th>
<th>Avg. Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phred13</td>
<td>81,920</td>
<td>2,114,134</td>
<td>0.000083</td>
<td>88</td>
<td>52</td>
</tr>
<tr>
<td>Pexprid13</td>
<td>89,800</td>
<td>2,352,056</td>
<td>0.000097</td>
<td>155</td>
<td>35</td>
</tr>
<tr>
<td>Phred13</td>
<td>94,905</td>
<td>5,280,575</td>
<td>0.00072</td>
<td>299</td>
<td>68</td>
</tr>
</tbody>
</table>

- AMMendrix:
  - n: 220,542; m: 5,277,347; Density: 0.00002; Sparsity: 97; Avg. Degree: 47

- Map undir:
  - n: 958,694; m: 22,785,513; Density: 0.00005; Sparsity: 94; Avg. Degree: 46

- Map undir:
  - n: 617,792; m: 8,912,378; Density: 0.00001; Sparsity: 76; Avg. Degree: 46

- Linear Car Analysis:
  - n: 227,362; m: 5,376,634; Density: 0.00023; Sparsity: 33; Avg. Degree: 48

Benchmarks

- Sequential Algorithms
  - First-Fit
  - SDO & LDO
    - Implementation direct from H. Al-Omari and K. E. Sabri, "New Graph Coloring Algorithms"
    - \(O(n^3)\)
    - up to 1000x speedups!!!
    - Optimized (our) implementation (as a red black Tree)
      - \(O(m \log n)\)
      - 20x - 40x speedup

Implementation

- Some Details:
  - Tests were carried out on a Tesla S1070 and Fermi GTX 480 with Caching
  - Memory transfer time included in all results
  - All results are the average of 10 runs

- Detailed times:
  - Graph = hood; Algo = First Fit
  - Memory transfer Transfer in: 20.9275 ms; Transfer out: 0.874112 ms
  - n: number of vertices, m: number of edges

- Getting boundary list: 12.243 ms; Final CPU steps time: 1.63 ms:
  - Total GPU = 96 ms vs Total CPU = 130 ms
Results: Classes

• Interesting pattern in test results; 3 classes identified
  o Class 1:
    • pkustk10, pkustk11, shipsec1, shipsec5, shipsec8, msdoor & hood
    • Speedup steadily increases initially and eventually plateaus;
      Coloring improves the more threads we use
    • Ratio of maximum to average degree is between 1.6 and 2.5

• Class 2
  o pwtk, bmw3_2 and idoor
  o Speedup steadily increases but then drops off at a certain point;
    best coloring is found before the dip
  o All the graphs in this class are quite large; pwtk & bmw3_2 have similar
    densities and size; there is a larger ratio of maximum to average degree
    than in class 1; 3.4 for pwtk and 6.8 for bmw3_2

• Class 3
  o ct20stif, nasasrb & pkustk13
  • Speedup steadily increase
  • Best color is in the middle of range
  • Ratio of maximum degree is approximately 4 times the average degree

Results: Subgraph Size

• Small subgraph size produce better speedup and colors
**Results: Subgraph Size**

- **Results: Tesla vs Fermi**
  - Obviously Fermi is faster!

**Results: Metis vs Non-Metis**

- Naive partitioning is most of the time faster and more efficient than using Metis
  - Metis is not effective for such small partitions
  - Yields unbalanced partitions at such small graph sizes
  - Unbalanced is bad for GPU
- Metis was slower despite not taking into account the time for Metis to run!

**Conclusion**

- Set of guidelines for graph coloring
  - Fastest: Parallel First Fit
    - Much better colors than Sequential First Fit
    - Slightly slower than Sequential First Fit
  - Best Results (if you do not care about speed)
    - Sequential SDO & LDO implemented as a Red-Black Tree
    - Balance of Speed and Colors
      - Parallel Max Out or Min Out - average of 20X speedup over sequential SDO & LDO
- Use small subgraph Sizes
- Naive partitioning is good
- CUDA does not only makes calculations faster but can also be used to improve results - First Fit!
Questions?