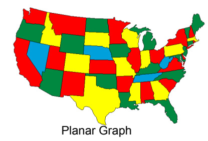
Evaluating Graph Coloring on GPUs

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

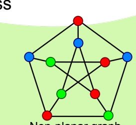
Pascal Grosset, Peihong Zhu, Shusen Liu,
Suresh Venkatasubramanian, and Mary Hall

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
  - ...



Planar Graph



Non-planar graph

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

Benchmarks

Degree: number of neighbors of a vertex, Saturation: number of differently colored neighbors

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

```
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

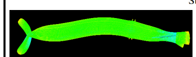
- Transfer data from CPU to GPU

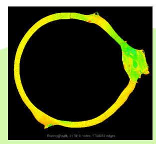
1. Graph Coloring
   - Run Graph Coloring: 1 thread per subgraph
     - cudaEventSynchronize -
2. Conflicts Indentification
   - sets color of conflicted nodes to 0: 1 thread per node
     - cudaEventSynchronize -

- Transfer Conflicts to CPU
- Count Conflicts
  - If conflicts < threshold
    - exit
  - Else
    - Repeat from 1

Test Data

- Data source: University of Florida Sparse Matrix Collection


nasasrb


pwtk

Name	n	m	Density	Δ	Avg Degree
Structural Engineering					
ct20stif	52,329	1,375,396	0.00100	206	50
nasasrb	54,870	1,366,097	0.00091	275	47
pwtk	217,918	5,926,171	0.00025	179	52
Civil Engineering					
pkustk10	81,920	2,114,154	0.00063	89	52
pkustk11	87,804	2,565,054	0.00067	131	55
pkustk13	94,893	3,260,967	0.00072	299	68
Automotive					
hood	220,542	5,273,947	0.00020	76	47
ldoor	958,464	22,785,136	0.00005	76	46
msdoor	417,792	9,912,536	0.00011	76	46
Ship section					
shipsec1	140,874	3,836,265	0.00039	101	54
shipsec5	176,860	4,966,618	0.00032	125	55
shipsec8	114,919	3,269,240	0.00050	101	54
Linear Car Analysis					
bmw3_2	227,362	5,539,634	0.00021	335	49

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

n: number of vertices, m: number of edges

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

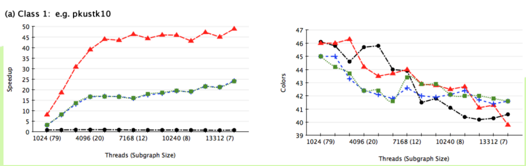
	Pass 1	Pass 2	Pass 3
Coloring	23.569	15.0256	2.19456
Detect	7.1281	6.6528	6.58186
Count Time	0.003488	0.003712	0.00336

Getting boundary list: 12.2434 ms + Final CPU steps time: 1.63 ms:

- Total GPU: ~ 96 ms vs Total CPU: ~ 100 ms

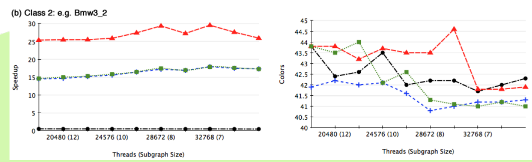
Results: Classes

- Interesting pattern in test results; 3 classes identified
  - 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



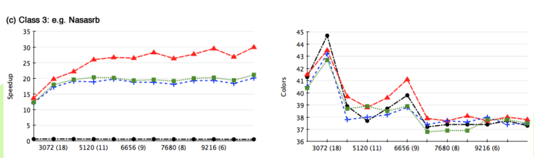
Results: Classes

- Class 2
  - pwtk, bmw3_2 and ldoor
  - Speedup steadily increases but then drops off at a certain point; best coloring is found before the dip
  - 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



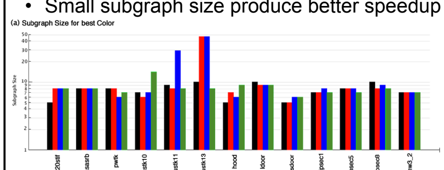
Results: Classes

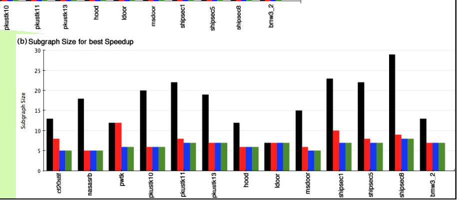
- Class 3
  - ct20stif, nasasrb & pkustk13
    - Speedup steadily increase
      Best color is in the middle of range
    - Ratio of maximum degree is approximately 4 times the average degree



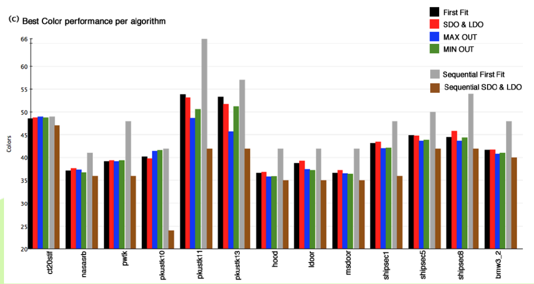
Results: Subgraph Size

- Small subgraph size produce better speedup and colors



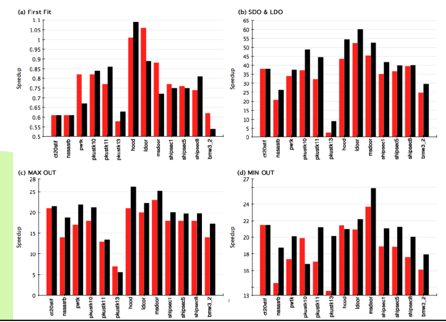


Results: Subgraph Size



Results: Tesla vs Fermi

- Obviously Fermi is faster!

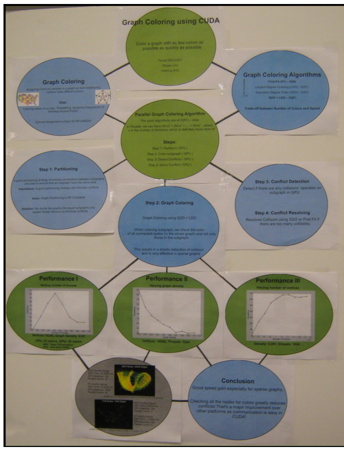


Results: Metis vs Non-Metis

- 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!

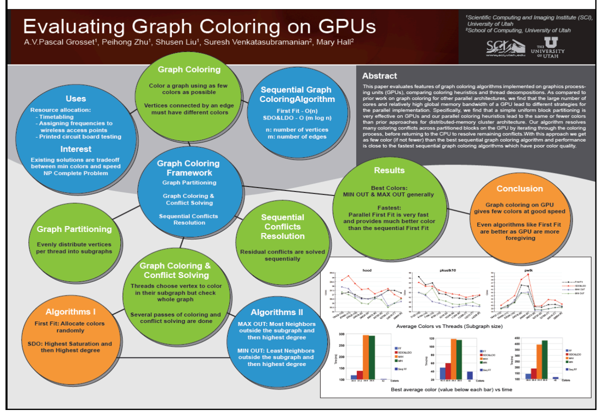
Conclusion

- 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!



Class Poster

PPoPP Final Poster



Questions?