A High-Quality and Fast Maximal Independent Set Algorithm for GPUs

Martin Burtscher and Sindhu Devale
Department of Computer Science

TEXAS STATE UNIVERSITY
The rising STAR of Texas

Efficient Computing Laboratory
Overview

- Introduction
- Serial and parallel algorithms
- Our parallel algorithm
- Optimizations
- Results
- Summary
Maximal Independent Set

- Maximal independent set (**MIS**)
  - **Subset** of vertices of undirected graph
  - Vertices in subset are **independent** (not adjacent)
  - Subset is **maximal** (all other vertices are adjacent)
  - Not unique

- Largest possible MIS
  - **Maximum** independent set
  - NP-hard to compute
Importance of MIS

- Building block of many parallel graph algorithms
  - Graph coloring
  - Maximal matching
  - 2-satisfiability
  - Maximal set packing
  - Odd set cover problem
  - etc.
Importance of MIS (cont.)

- **Parallelization of complex computations**
  - Supports arbitrary and dynamically changing conflicts
    1. Build graph (vertices = computations, edges = conflicts)
    2. Compute MIS
    3. Run computations in MIS in parallel (w/o locks or atomics)
    4. Repeat if necessary
  - E.g., Delaunay mesh refinement

- Approach is only useful if MIS can be computed quickly in parallel and benefits from large sets
Highlights

- ECL-MIS algorithm for massively-parallel devices
  - Fastest MIS runtimes on modern GPUs
- Randomized permutation selection function
  - Largest set sizes among many MIS algorithms
- New optimizations
  - Enhance performance and reduce memory footprint
Serial Algorithm
Serial Algorithm

- Repeating steps
  - Visit unvisited vertex
  - Add vertex to set if no graph neighbors in set

- Example
  - Start with empty set

- Set = {}
Serial Algorithm

- Repeating steps
  - Visit unvisited vertex
  - Add vertex to set if no graph neighbors in set

- Example
  - a has no neighbor in set
  - Add vertex a

- Set = \{a\}
Serial Algorithm

- Repeating steps
  - Visit unvisited vertex
  - Add vertex to set if no graph neighbors in set

- Example
  - b has neighbor in set
  - Discard vertex b

- Set = {a}
Serial Algorithm

- Repeating steps
  - Visit unvisited vertex
  - Add vertex to set if no graph neighbors in set

- Example
  - c has neighbor in set
  - Discard vertex c

- Set = \{a\}
Serial Algorithm

- Repeating steps
  - Visit unvisited vertex
  - Add vertex to set if no graph neighbors in set

- Example
  - $d$ has neighbor in set
  - Discard vertex $d$

- Set = \{a\}
Serial Algorithm

- Repeating steps
  - Visit unvisited vertex
  - Add vertex to set if no graph neighbors in set

- Example
  - $e$ has no neighbor in set
  - Add vertex $e$

- Set = \{a, e\}
Serial Algorithm

- Repeating steps
  - Visit unvisited vertex
  - Add vertex to set if no graph neighbors in set

- Example
  - $f$ has neighbor in set
  - Discard vertex $f$

- Set = \{a, e\}
Serial Algorithm

- Repeating steps
  - Visit unvisited vertex
  - Add vertex to set if no graph neighbors in set

- Example
  - $g$ has neighbor in set
  - Discard vertex $g$

- Set = \{a, e\}
Serial Algorithm

- Repeating steps
  - Visit unvisited vertex
  - Add vertex to set if no graph neighbors in set

- Example
  - h has no neighbor in set
  - Add vertex h

- Set = \{a, e, h\}
Serial Algorithm

- Repeating steps
  - Visit unvisited vertex
  - Add vertex to set if no graph neighbors in set

- Example
  - \(i\) has neighbor in set
  - Discard vertex \(i\)

- \(\text{MIS} = \{a, e, h\}\)
Luby’s Random-Priority Parallel MIS Algorithm
Random-Priority Algorithm (Luby)

- Repeating steps
  - Assign random priorities
  - Add vertices with highest local priority to set
  - Remove their neighbors from graph

- Set = {}
Random-Priority Algorithm (Luby)

- Repeating steps
  - Assign random priorities
  - Add vertices with highest local priority to set
  - Remove their neighbors from graph

Set = {}
Random-Priority Algorithm (Luby)

- Repeating steps
  - Assign random priorities
  - Add vertices with highest local priority to set
  - Remove their neighbors from graph

Set = \{b, e\}
Random-Priority Algorithm (Luby)

- Repeating steps
  - Assign random priorities
  - Add vertices with highest local priority to set
  - Remove their neighbors from graph

- Set = \{b, e\}
Random-Priority Algorithm (Luby)

- Repeating steps
  - Assign random priorities
  - Add vertices with highest local priority to set
  - Remove their neighbors from graph

Set = \{b, e\}
Random-Priority Algorithm (Luby)

- Repeating steps
  - Assign random priorities
  - Add vertices with highest local priority to set
  - Remove their neighbors from graph

- Set = \{b, c, e, i\}
Random-Priority Algorithm (Luby)

- Repeating steps
  - Assign random priorities
  - Add vertices with highest local priority to set
  - Remove their neighbors from graph

- MIS = \{b, c, e, i\}
Random-Permutation Parallel MIS Algorithm
Random-Permutation Algorithm

- Initialization
  - Assign random priorities
- Repeating steps
  - Add vertices with highest local priority to set
  - Remove neighbors and their edges from graph
- Set = {}
Random-Permutation Algorithm

- Initialization
  - Assign random priorities
- Repeating steps
  - Add vertices with highest local priority to set
  - Remove neighbors and their edges from graph
- Set = {}
Random-Permutation Algorithm

- Initialization
  - Assign random priorities

- Repeating steps
  - Add vertices with highest local priority to set
  - Remove neighbors and their edges from graph

- Set = \{b, e\}
Random-Permutation Algorithm

- Initialization
  - Assign random priorities
- Repeating steps
  - Add vertices with highest local priority to set
  - Remove neighbors and their edges from graph
- Set = \{b, e\}
Random-Permutation Algorithm

- Initialization
  - Assign random priorities

- Repeating steps
  - Add vertices with highest local priority to set
  - Remove neighbors and their edges from graph

- Set = \{b, c, e, h\}
Random-Permutation Algorithm

- **Initialization**
  - Assign random priorities

- **Repeating steps**
  - Add vertices with highest local priority to set
  - Remove neighbors and their edges from graph

- **MIS** = \{b, c, e, h\}
Luby’s Random-Selection Parallel MIS Algorithm
Random-Selection Algorithm (Luby)

- Repeating steps
  - Mark vertices with probability $0.5/\text{degree}$
  - Add marked vertices to set if no marked neighbors
  - Remove their neighbors from graph

- Set = $\{\}$
Random-Selection Algorithm (Luby)

- Repeating steps
  - Mark vertices with probability $0.5/\text{degree}$
  - Add marked vertices to set if no marked neighbors
  - Remove their neighbors from graph
- Set = {}
Random-Selection Algorithm (Luby)

- Repeating steps
  - Mark vertices with probability 0.5/degree
  - Add marked vertices to set if no marked neighbors
  - Remove their neighbors from graph
- Set = \{b, d\}
Random-Selection Algorithm (Luby)

- Repeating steps
  - Mark vertices with probability $0.5/\text{degree}(v)$
  - Add marked vertices to set if no marked neighbors
  - Remove their neighbors from graph

- Set = \{b, d\}
Random-Selection Algorithm (Luby)

- Repeating steps
  - Mark vertices with probability $0.5/\text{degree}$
  - Add marked vertices to set if no marked neighbors
  - Remove their neighbors from graph

- Set $= \{b, d\}$
Random-Selection Algorithm (Luby)

- Repeating steps
  - Mark vertices with probability $0.5/\text{degree}$
  - Add marked vertices to set if no marked neighbors
  - Remove their neighbors from graph

Set = \{b, c, d, f, i\}
Random-Selection Algorithm (Luby)

- Repeating steps
  - Mark vertices with probability $0.5/\text{degree}$
  - Add marked vertices to set if no marked neighbors
  - Remove their neighbors from graph

- MIS = \{b, c, d, f, i\}
ECL-MIS
Our Permutation-Selection Parallel MIS Algorithm
Our Permutation-Selection Algorithm

- **Initialization**
  - Assign priorities \( \sim 1/\text{deg} \)
  - Randomize within level

- **Repeating steps**
  - Add vertices with highest local priority to set
  - Remove their neighbors from graph

- **Set** = \( \{\} \)
Our Permutation-Selection Algorithm

- Initialization
  - Assign priorities \(\sim 1/\text{deg}\)
  - Randomize within level

- Repeating steps
  - Add vertices with highest local priority to set
  - Remove their neighbors from graph

- Set = {}
Our Permutation-Selection Algorithm

- Initialization
  - Assign priorities \( \sim 1/\text{deg} \)
  - Randomize within level

- Repeating steps
  - Add vertices with highest local priority to set
  - Remove their neighbors from graph

- Set = \( \{ \} \)
Our Permutation-Selection Algorithm

- **Initialization**
  - Assign priorities \( \sim 1/\text{deg} \)
  - Randomize within level

- **Repeating steps**
  - Add vertices with highest local priority to set
  - Remove their neighbors from graph

- **Set** = \( \{b, c, d, h\} \)
Our Permutation-Selection Algorithm

- **Initialization**
  - Assign priorities $\sim 1/\text{deg}$
  - Randomize within level

- **Repeating steps**
  - Add vertices with highest local priority to set
  - Remove their neighbors from graph

- **Set** = \{b, c, d, h\}
Our Permutation-Selection Algorithm

- **Initialization**
  - Assign priorities $\sim \frac{1}{\text{deg}}$
  - Randomize within level

- **Repeating steps**
  - Add vertices with highest local priority to set
  - Remove their neighbors from graph

- **Set** = \{b, c, d, f, h\}
Our Permutation-Selection Algorithm

- Initialization
  - Assign priorities $\sim 1/\text{deg}$
  - Randomize within level

- Repeating steps
  - Add vertices with highest local priority to set
  - Remove their neighbors from graph

- MIS = \{b, c, d, f, h\}
ECL-MIS Features

- Single initialization
  - Requires less work (faster)
  - Enables *asynchronous* implementation (faster)
- Permutation-selection function
  - *Boosts* set size (higher-quality result)
  - Requires only a few bits (lower memory footprint)
- Combined priority and status information
  - Reduces storage (lower memory footprint)
  - Minimizes memory accesses (faster)
Permutation-Selection Function

- Requirements
  - Has to work for all graphs
  - Needs to be proportional to $1/\text{degree}$
  - Do not know highest degree (but know average)

- Our solution
  \[ \text{priority}(v) = \frac{\text{avg\_degree}}{\text{avg\_degree} + \text{degree}(v)} \]
  - Degree includes random fraction (e.g., 3.xyz)
  - Scaled to small integer (e.g., a byte)
Permutation-Selection Function

Wide range at low degrees

50% of range is below avg degree

Makes ties unlikely

Narrow range at high degrees

Ties are likely but unimportant
Combining Information

- Standard implementation (2 arrays)
  - 1st array: vertex state (undecided, in set, out of set)
  - 2nd array: vertex priority (random number)

- Our implementation (1 array)
  - 7 MSBs hold combined status and priority
    - Reserved highest value: in set (= higher than its neighbors)
    - Reserved lowest value: out of set (= removed from graph)
    - Remaining values = priority
  - LSB = decided/undecided (to boost performance)
Results
Methodology

- **System**
  - GPUs: *Titan X* and *K40*, nvcc 8.0
  - CPUs: 2 Xeon E5-2687W v3 (20 cores, 3.1GHz), gcc 5.3

- **GPU MIS codes**
  - CUSP, ECL, IrGL, and Pannotia

- **CPU MIS codes**
  - Ligra, Ligra+, and PBBS (Cilk and OpenMP, incremental and non-deterministic)
  - PBBS (serial)
## Input Graphs

- **16 graphs**
  - Real-world + synth.
  - All made undirected

### Sizes
- 66k – 24M vertices
- 387k – 524M edges
- 0 – 214k degrees

<table>
<thead>
<tr>
<th>Graph</th>
<th>Vertices</th>
<th>Edges</th>
<th>Min Degree</th>
<th>Max Degree</th>
<th>Avg Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>2d-2e20.sym</td>
<td>1,048,576</td>
<td>4,190,208</td>
<td>2</td>
<td>4</td>
<td>4.0</td>
</tr>
<tr>
<td>amazon0601</td>
<td>403,394</td>
<td>4,886,816</td>
<td>1</td>
<td>2,752</td>
<td>12.1</td>
</tr>
<tr>
<td>as-skitter</td>
<td>1,696,415</td>
<td>22,190,596</td>
<td>1</td>
<td>35,455</td>
<td>13.1</td>
</tr>
<tr>
<td>citationCiteSeer</td>
<td>268,495</td>
<td>2,313,294</td>
<td>1</td>
<td>1,318</td>
<td>8.6</td>
</tr>
<tr>
<td>cit-Patents</td>
<td>3,774,768</td>
<td>33,037,894</td>
<td>1</td>
<td>793</td>
<td>8.8</td>
</tr>
<tr>
<td>coPapersDBLP</td>
<td>540,486</td>
<td>30,491,458</td>
<td>1</td>
<td>3,299</td>
<td>56.4</td>
</tr>
<tr>
<td>delaunay_n24</td>
<td>16,777,216</td>
<td>100,663,202</td>
<td>3</td>
<td>26</td>
<td>6.0</td>
</tr>
<tr>
<td>in-2004</td>
<td>1,382,908</td>
<td>27,182,946</td>
<td>0</td>
<td>21,869</td>
<td>19.7</td>
</tr>
<tr>
<td>internet</td>
<td>124,651</td>
<td>387,240</td>
<td>1</td>
<td>151</td>
<td>3.1</td>
</tr>
<tr>
<td>kron_g500-logn21</td>
<td>2,097,152</td>
<td>182,081,864</td>
<td>0</td>
<td>213,904</td>
<td>86.8</td>
</tr>
<tr>
<td>r4-2e23.sym</td>
<td>8,388,608</td>
<td>67,108,846</td>
<td>2</td>
<td>26</td>
<td>8.0</td>
</tr>
<tr>
<td>rmat16.sym</td>
<td>65,536</td>
<td>967,866</td>
<td>0</td>
<td>569</td>
<td>14.8</td>
</tr>
<tr>
<td>rmat22.sym</td>
<td>4,194,304</td>
<td>65,660,814</td>
<td>0</td>
<td>3,687</td>
<td>15.7</td>
</tr>
<tr>
<td>uk-2002</td>
<td>18,520,486</td>
<td>523,574,516</td>
<td>0</td>
<td>194,955</td>
<td>28.3</td>
</tr>
<tr>
<td>USA-road-d.NY</td>
<td>264,346</td>
<td>730,100</td>
<td>1</td>
<td>8</td>
<td>2.8</td>
</tr>
<tr>
<td>USA-road-d.USA</td>
<td>23,947,347</td>
<td>57,708,624</td>
<td>1</td>
<td>9</td>
<td>2.4</td>
</tr>
</tbody>
</table>
Titan X Performance (Edges/Second)

ECL-MIS is >3.9x faster on each tested graph

>12x faster than other codes on average

>100x faster than other codes on kron

A High-Quality and Fast MIS Algorithm
ECL-MIS is >3.8x faster on each tested graph

>70x faster than other codes on kron

>9x faster than other codes on average
Set Size (Deterministic)

ECL-MIS yields largest set on all but one graph

10% larger on average
Randomization does not affect ECL-MIS set size
Random selection yields 1.7% smaller sets
Random permutation yields 10% smaller sets
Using 16-bit values yields a 15% slowdown

Using 32-bit values yields a 33% slowdown

Not using randomization yields a 6% slowdown

Synchronous execution yields an 18% slowdown

Using 2 separate arrays yields an 18% slowdown

Visiting all neighbors yields a 59% slowdown

Combination of optimizations is key
Comparison to CPU Codes (Averages)

ECL-MIS is fastest on all but one tested graph:
>2.9x faster than CPU codes on average

ECL-MIS yields largest set on 15 of 16 graphs:
9% to 11% larger on average
Summary

- ECL-MIS maximal independent set algorithm
  - Fastest GPU implementation (due to optimizations)
  - Produces largest sets (due to permutation selection)
- Atomic-free CUDA implementation
  - http://cs.txstate.edu/~burtscher/research/ECL-MIS/

Acknowledgments
- NSF grant 1406304
- Nvidia donations