Part II: Vectorisation
MOTIVATION

Graph processing - Irregular, data-dependent memory accesses
Vectorization a.k.a. Single Instruction Multiple Data (SIMD) parallelism
Push vs pull – open question which one to use
Require vectorized push and pull style traversal; no generic solution for push in literature

1. CleanCut: conflict-free and load-balanced partitions
2. VectorFast: compact graph representation to ease memory pressure
   Fast-forwarding for frontiers or convergence optimization
3. Graptor Domain-Specific Language (DSL) and Compiler
   Ease of use (e.g., auto-vectorization) and efficiency (compiler optimizations)

BACKGROUND: GRAPH DATA STRUCTURES (PULL)

Adjacency matrix
12 vertices -> 12x12 matrix
31 edges -> 31 non-zero elements

Compressed Sparse (scalar)
Ghost cell in index array shows number of edges

Padded Compressed Sparse
Padding ensures alignment
L-to-1 reductions required

Colors show distinct vector lanes
Assuming L=4 lanes
BACKGROUND: GRAPH DATA STRUCTURES (PULL)

Data-Parallel Vectorization
- Lanes process distinct neighbor lists
- Many disabled lanes

SELL-C-sigma
- Apply degree sorting [Kreutzer SIAM’14]
- Load imbalance for power-law graphs [Besta IPDPS’17]
BACKGROUND: GRAPH DATA STRUCTURES (PULL)

SELL-C-Sigma [Kreutzer SIAM’14]
Load imbalance for power-law graphs [Besta IPDPS’17]

For each edge u → v:
new_pr[v] += old_pr[u]/deg[u];

Load imbalance:
20 vs 11 edges

Graph with
12 vertices, 31 edges
CHALLENGES OF PUSH STYLE VECTORIZATION

Push style traversal interchanges read/write property arrays

*Edges* array shows array indices being updated

Sequential reads of old_pr

Data-dependent, irregular access to new_pr

For each edge \( u \rightarrow v \):
\[
\text{new}_\text{pr}[v] += \frac{\text{old}_\text{pr}[u]}{\text{deg}[u]};
\]

**Load imbalance:**
20 vs 11 edges

**Inter-thread conflicts in push**
CLEANCUT: RACE-FREE GRAPH PARTITIONS: PUSH

Load imbalance: 20 vs 11 edges

Inter-lane conflicts in push

Inter-thread conflicts in push

Approach:
Partition destinations
No inter-thread races
Load balanced (number of edges)

Thread 0
Thread 1

Use owner-computes and VEBO

No longer degree-sorted Padding overhead

QUEEN'S UNIVERSITY BELFAST
CLEANCUT: RACE-FREE GRAPH PARTITIONS: PUSH

**Approach:**
Compact and sort differently in each partition
Additional array to indicate sort order (source vertex IDs)

**Thread 0**

<table>
<thead>
<tr>
<th>Source Index</th>
<th>Edges (Destinations)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 2 3</td>
</tr>
<tr>
<td>1</td>
<td>0 2 3</td>
</tr>
<tr>
<td>2</td>
<td>0 1 2</td>
</tr>
<tr>
<td>3</td>
<td>1 3 2</td>
</tr>
<tr>
<td>4</td>
<td>1 3 2</td>
</tr>
<tr>
<td>5</td>
<td>6 1 2</td>
</tr>
<tr>
<td>6</td>
<td>7 1 2</td>
</tr>
<tr>
<td>7</td>
<td>7 1 2</td>
</tr>
</tbody>
</table>

**Thread 1**

<table>
<thead>
<tr>
<th>Source Index</th>
<th>Edges (Destinations)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4 5 9</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>7 8</td>
</tr>
<tr>
<td>4</td>
<td>7 8</td>
</tr>
<tr>
<td>5</td>
<td>0 3 2</td>
</tr>
<tr>
<td>6</td>
<td>2 0 1</td>
</tr>
<tr>
<td>7</td>
<td>3 0 1</td>
</tr>
</tbody>
</table>

Load balanced and no inter-thread conflicts
Still has inter-lane conflicts

**Approach:**
Swap destinations or add padding to avoid inter-lane race conditions

**Thread 0**

<table>
<thead>
<tr>
<th>Source Index</th>
<th>Edges (Destinations)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 2 3</td>
</tr>
<tr>
<td>1</td>
<td>0 3 2</td>
</tr>
<tr>
<td>2</td>
<td>0 3 2</td>
</tr>
<tr>
<td>3</td>
<td>1 2 3</td>
</tr>
<tr>
<td>4</td>
<td>0 3 2</td>
</tr>
<tr>
<td>5</td>
<td>3 1 2</td>
</tr>
<tr>
<td>6</td>
<td>3 1 2</td>
</tr>
<tr>
<td>7</td>
<td>0 3 2</td>
</tr>
</tbody>
</table>

May increase padding
EXPERIMENTAL EVALUATION

Hybrid processing with sparse (push)/dense (push or pull) phases based on frontier density
Vectorize dense phases only

Knights Landing (KNL): Xeon Phi 7210, 1.3 GHz
64 threads, 16-way SIMD with AVX512F
gcc 7.2.0
Intel Cilkplus parallel runtime

<table>
<thead>
<tr>
<th>Graph</th>
<th>Vertices</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>orkut</td>
<td>3.07M</td>
<td>234M</td>
</tr>
<tr>
<td>twitter</td>
<td>41.7M</td>
<td>2.41G</td>
</tr>
<tr>
<td>friendster</td>
<td>125M</td>
<td>5.17G</td>
</tr>
<tr>
<td>RMAT27</td>
<td>134M</td>
<td>2.21G</td>
</tr>
<tr>
<td>USAroad</td>
<td>23.9M</td>
<td>57.7M</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>BFS</td>
<td>Breadth-first-search</td>
</tr>
<tr>
<td>CC</td>
<td>Connected components</td>
</tr>
<tr>
<td>LVL</td>
<td>BFS variant</td>
</tr>
<tr>
<td>FM</td>
<td>Radius estimation</td>
</tr>
<tr>
<td>MIS</td>
<td>Maximal Independent Set</td>
</tr>
<tr>
<td>PR</td>
<td>PageRank</td>
</tr>
<tr>
<td>APR</td>
<td>Accelerated PageRank</td>
</tr>
</tbody>
</table>
Some algorithms prefer push, others prefer pull.

Good vectorized configurations are faster than scalar.

Caching generally helps in pull, not always in push.

Twitter graph
Xeon Phi 7210, 1.3 GHz
64 threads, 16-way SIMD
Part III: Reduced Precision
REDUCED PRECISION COMPUTATION

Motivation: reduce memory pressure related to “random” memory accesses

Fewer bits/vertex, so more vertex data in cache

Half-precision floating point – also used for compact neural networks

Need software emulation!

<table>
<thead>
<tr>
<th>Format</th>
<th>Origin</th>
<th>Size</th>
<th>Exponent</th>
<th>Mantissa</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP16</td>
<td>IEEE</td>
<td>16</td>
<td>5</td>
<td>10</td>
<td>General</td>
</tr>
<tr>
<td>BFloat</td>
<td>Intel</td>
<td>16</td>
<td>8</td>
<td>7</td>
<td>Deep Learning</td>
</tr>
<tr>
<td>DLFloat</td>
<td>IBM</td>
<td>16</td>
<td>6</td>
<td>9</td>
<td>Deep Learning</td>
</tr>
<tr>
<td>CPMS</td>
<td>KIT</td>
<td>16</td>
<td>11</td>
<td>4</td>
<td>PageRank</td>
</tr>
<tr>
<td>6+10</td>
<td>QUB</td>
<td>16</td>
<td>6</td>
<td>10</td>
<td>PageRank a.o.</td>
</tr>
</tbody>
</table>

A CUSTOM FORMAT FOR PAGERANK

- All PageRank values are **non-zero** and **less than 1**
- Values are strictly positive – no sign bit needed
- The smallest PageRank value is \((1-d)/|V|\)
  - \(d\) is damping factor, typically 0.85
- Exponents take between \(-\log |V|-1\) bits
- 6 exponent bits cover graphs with \(\sim 2^{64}\) vertices
- 5 exponent bits cover graphs with \(\sim 2^{32}\) vertices
CONVERSION

- Trick: store compact numbers in memory, convert to hardware-supported format when computing
- No range checking, rounding.
- No zero, NaN, Inf, denormalized numbers
- Reinterpret integer values as floating-point values
  - Note: scalar execution on x86-64 adds 3-cycle overhead for move between register files
- Two 1-cycle assembly operations to convert “6+10” format to IEEE-754 FP32 or FP64
  - Performance very sensitive to details of code!
CONVERGENCE RATE
PAGERANK

Twitter graph
“6+10” reduced precision format supports convergence to within 1e-4

Need to switch precision when convergences stalls:
• Residual error < 1e-4
• or improved by less than 10%
PERFORMANCE RESULTS
PAGERANK

Data formats for PageRank values in memory and accumulator in register
8-way parallelism
• does not converge

End-to-end speedup when switching "6+10" to FP32 is 8.4% to 16.4%

Intel Xeon Gold 6130, 2 sockets, 16 cores per socket, AVX-512

![Graph showing execution time per power iteration for different data formats and datasets]
CONCLUSION

Graph analytics performance strongly impacted by memory issues
Data layout matters, need to differentiate by memory access patterns
Load-balanced partitions can be obtained by low-overhead vertex relabeling
Vectorisation requires handling of intra- and inter-thread conflicts
Memory pressure can be relieved by reduced-precision computation
Graptor provides high-level map abstractions and optimizes performance ”under the hood”
WITH THANKS TO

- Jiawen Sun
- Amir Sabbagh Molahosseini
- Mohsen Koohi Esfahani
- Giorgis Georgakoudis
- Mahwish Arif
- Dimitrios S. Nikolopoulos