Reusing Data Reorganization for Efficient SIMD Parallelization of Dynamic Irregular Applications

Peng Jiang, Linchuan Chen, and Gagan Agrawal

Department of Computer Science and Engineering
The Ohio State University, USA
Motivation - Applications

• Irregular Data Access is Very Common
  • Static Irregular
    • Bellman-Ford
    • Sparse matrix operations …
  • Dynamic Irregular
    • Vertex-centric BFS, SSSP…
  • Adaptive Dynamic Irregular
    • Particle Simulation …
Motivation - SIMD Hardware Evolution

- More Powerful SIMD features
  - Wider SIMD lanes
  - Gather/scatter operations enable irregular memory accesses
  - Mask data type and operations enable computation on specified SIMD lanes

- Challenging to Exploit for Applications with Irregular Memory Access
  - Memory access locality
  - Write conflict
Intel Xeon Phi

- Large-scale Parallelism
  - 61 cores
  - Each supports 4 hyper-threads

- Wide SIMD Lanes
  - 512 bit lane width = 16 floats / 8 doubles

- More Flexible SIMD Programming with Gather/Scatter
  - Enables the programming for irregular memory access
Intel Xeon Phi

• **Efficient Utilization is Critical**
  • Data Access Locality
    • SIMD level
      • Irregular access with Gather/Scatter
  • SIMD Utilization Ratio
    • Need to keep as many as possible SIMD lanes busy
  • Synchronization Overhead
    • Need to avoid *write conflicts* among SIMD lanes
    • Use of locking among threads should also be minimized or avoided
Our Previous Work

• Exploiting Recent SIMD Architectural Advances for Irregular Applications (CGO’16)
  • A General Optimization Methodology
    • Based on a sparse matrix view
    • Data access patterns identification made easy
  • Three Steps (tiling and grouping)
    • Data locality enhancement through matrix tiling
    • Data access pattern identification
    • Write conflict removal at both SIMD and MIMD levels
  • Aims at Static Irregular Applications
    • Bellman-Ford, PageRank, SPMM
Challenges

- **Dynamic Irregular Applications**
  - In some vertex-centric graph applications (SSSP, BFS)
    - The topology of the graph is determined
    - But the memory access patterns differ among iterations

- **Adaptive Irregular Applications**
  - In particle simulation (Molecular Dynamics)
    - Edges can be removed, new edges can be added
    - The edges changes adaptively

- **Pre-reorganizing Data is Not Effective**
  - Memory access patterns change
Contributions

• **An Approach to Efficiently Vectorizing (Adaptive) Dynamic Applications**
  - Based on the tiling and grouping method
  - Reuse the data reorganization

• **The Steps**
  - Pre-reorganize the data with tiling and grouping
  - Manage the active memory accesses in the reorganized data during execution
  - Apply SIMD to the computation with managed data

• **Subsets Studied**
  - Vertex-centric graph algorithms with dynamic memory access
  - Particle simulation with adaptive dynamic memory access
Utilizing SIMD in Vertex-Centric Graph Processing

- **Dynamic Memory Access Pattern**
  - Vertex-centric SSSP
    - Only the *frontier* vertices are active in each iteration
  - The memory access pattern changes over iteration

```cpp
class SSSPVertex: public Vertex {
  public:
    void Compute(MessageIterator* msgs) {
      if (superstep() == 0) *MutableValue() = INF;
      float mindist = IsSource(vertex_id()) ? 0 : INF;
      while (!msgs->Done()) {
        mindist = mindist < msgs->Value() ? mindist : msgs->Value();
        msgs->Next();
      }
      if (mindist < GetValue()) {
        *MutableValue() = mindist;
        OutEdgeIterator iter = GetOutEdgeIterator();
        for (; !iter.Done(); iter.Next()) {
          SendMessageTo(iter.Target(), mindist + iter.GetValue());
        }
      } else {
        VoteToHalt();
      }
    }
};
```
Utilizing SIMD in Vertex-Centric Graph Processing

- **Our Approach**
  - Manage the active memory accesses in an index data structure
  - Reuse the tiled and grouped data to apply SIMD
Utilizing SIMD in Vertex-Centric Graph Processing

- **Index Data Structure**
  - Stores
    - The ordered edges (*Index*)
  
- The grouped edges (*Xcoord*, *Ycoord*)

- A mapping from ordered edges to grouped edges (*Pos*)
Utilizing SIMD in Vertex-Centric Graph Processing

- Search-and-Activate
  - The active edges are searched in ordered index with binary search

  - The positions of these active edges in the grouped edge array are obtained from the mapping

  - These edges are activated in the grouped array (ready for SIMD processing)
SIMD Execution for Adaptive Dynamic Applications

- The active edges in these applications change adaptively (e.g., Molecular Dynamics)

(a) Edges before Rebuilding  (b) Edges Changed in Rebuilding

Only 5% of the total edges are changed
SIMD Execution for Adaptive Dynamic Applications

• **Our Approach**
  • Pre-reorganize the initial edges (by tiling and grouping)

  • Every time the edges are rebuilt, inactivate the removed edges and add newly built edges

  • Incrementally maintain the grouped edges

  • Apply SIMD to the computation
SIMD Execution for Adaptive Dynamic Applications

- **Index Data Structure Collection**
  - Stores *multiple*:
    - Ordered edges (*Index*)
    - Grouped edges (*Xcoord, Ycoord*)
    - And mapping from ordered edges to grouped edges (*Pos*)
  - A new collection is added when edges are rebuilt
• **Search-and-Update**
  - The active edges are searched in multiple ordered indices with binary search

  - If an edge exists, obtain its position from the mapping, and activate it in the grouped edge array

  - Add all newly built edges in a new ordered index, group the new edges, and build mapping for the new edges
Results

• **Platform**
  • Intel Xeon Phi SE10P coprocessor
  • 61 cores, 1.1 GHz, 4 hyper threads per core
  • 8 GB GDDR5
  • Intel ICC 13.1.0, -O3 enabled

• **Applications**
  • Graph Algorithm
    • BFS, SSSP, SSWP, WCC, TC
  • Molecular Dynamics
    • Moldyn, MiniMD
Graph Algorithm

- Single thread performance, speedup by SIMD
Molecular Dynamics

- Overhead of neighbor rebuilding
- Time unit: seconds

<table>
<thead>
<tr>
<th>App</th>
<th>Dataset</th>
<th>Non-Group</th>
<th>Regroup-All</th>
<th>Inc-Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>Moldyn</td>
<td>16-3.0r</td>
<td>4.93</td>
<td>601.14</td>
<td>5.89</td>
</tr>
<tr>
<td></td>
<td>32-3.0r</td>
<td>20.21</td>
<td>950.23</td>
<td>8.43</td>
</tr>
<tr>
<td>MiniMD</td>
<td>16-3.0r</td>
<td>1.21</td>
<td>651.78</td>
<td>5.47</td>
</tr>
<tr>
<td></td>
<td>32-3.0r</td>
<td>8.37</td>
<td>1096.35</td>
<td>20.20</td>
</tr>
</tbody>
</table>

Even faster than Non-Group because the neighbor rebuilding are processed in tiles.
Molecular Dynamics

- Overall performance with single thread

![Graph showing performance comparisons between Serial and Inc-Group+SIMD for Moldyn and MiniMD. The graph displays execution time over 100 iterations in seconds for 16-3.0r and 32-3.0r sizes, with performance improvements noted as 4.43x, 3.44x, 4.35x, and 2.54x.]
Conclusions

- **Effective Approach for SIMDizing (Adaptive) Dynamic Irregular Applications**
  - Vertex-centric graph processing
  - Particle simulation

- **Efficient Management of Dynamic Memory Accesses**
  - Index Data Structure and Search-and-Activate procedure

- **Performance**
  - High efficiency in SIMD utilization
Thanks for your attention!

Q?

• Peng Jiang jiang.952@osu.edu
• Linchuan Chen chen.1996@osu.edu
• Gagan Agrawal agrawal@cse.ohio-state.edu