Jaccard Coefficients
What is a Jaccard Coefficient?

• Similarity between neighborhoods of two nodes (V, U):
  – Intersection(u, v) = |N(V) ∪ N(U)|
  – Union(u, v) = |N(V) ∩ N(U)|
  – Jaccard(V, U) = \frac{\text{Intersection}(u, v)}{\text{Union}(u, v)}
  – N(V) is the Neighborhood of V
Complexity of Computing Jaccard

• To compute Intersection(U, V)
  – If lists of neighbors are sorted:
    • $O(M)$ – $M$ is max of outdegree of U or V
  – If lists of neighbors are sorted first
    • $O(M \log(M))$
  – Otherwise perform repeated searches:
    • $O(M^2)$
Compute Jaccard With GraphBLAS

- GraphBLAS
  - Linear Algebra package to perform graph operations
  - Can be used to compute Jaccard efficiently
  - Represent graph G as matrix A, compute $A \times A = C$
  - Values in C correspond to the intersection size
  - Complexity: $O(\text{nnz}(A))$
Jaccard – Compute all pairs

• Can determine 0 value Jaccards to reduce work
• Intersect[N, N] array
• For each vertex V
  – For each vertex U in Neighborhood(V)
    • For each W in Neighborhood(U)
      – Intersect[V, W]++;
• Any pairs without a value have no shared neighborhood (intersection is empty)
Problems With This Algorithm

1. Compute each Jaccard twice \((U, W)\) and \((W, U)\)
   - Can be solved by checking ordering
   - Only count if \(U > W\) (based on arbitrary ordering)

2. \(N^2\) storage required
   - Only need the number of unique two-hop paths
   - Could store results in BST but will add to computational complexity
Goal of Project: Utilize High Bandwidth Memory (HBM)

- Compared to DDR HBM provides:
  - Equivalent Latency
  - Higher Bandwidth
  - Smaller capacity

- HBM is becoming Ubiquitous
  - GPU
  - KNL
  - Taihui Light
• MCDRAM can be configured:
  – Cache
  – Flat
  – Hybrid

• Which mode do we want to use if the problem will not fit in MCDRAM?
Previous Work: Cache-Oblivious Sorting

- 1.9x speedup over state-of-the-art
- For \( k \) threads, each thread sorts \( 1/k \) of the input data
- Each thread runs a divide and conquer sequential sort
  - Aggregation of all threads’ working sets fits in MCDRAM
- Once the \( k \) sorts complete, GNU multiway merge the results
- Can we adapt this concept to Jaccard?
Chunking With Jaccard

• Run in Flat mode
• Bring portion of data in, operate on it, move next portion in
• Could operate like producer/consumer problem

<table>
<thead>
<tr>
<th>Copy-in: Block 0</th>
<th>Copy-in: Block 1</th>
<th>Copy-in: Block 2</th>
<th>Copy-in: Block N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute: Block 0</td>
<td>Compute: Block 1</td>
<td>Compute: Block N-1</td>
<td>Compute: Block N</td>
</tr>
<tr>
<td>Copy-out: Block 0</td>
<td>Copy-out: Block N-2</td>
<td>Copy-out: Block N-1</td>
<td>Copy-out: Block N</td>
</tr>
</tbody>
</table>
Two Parallel Algorithms

1. Vertex Level
   - Each vertex is a task, threads compute two hop paths and Jaccard values
   - Accounts for imbalance fairly well, since there are many vertices

2. Spread pairs of vertices among threads
   - Easier to ensure no duplicate values are computed
   - Less parallelism during creation of problems
## Results

<table>
<thead>
<tr>
<th>Cache</th>
<th>Pair of vertices</th>
<th>Vertex-Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flat</td>
<td>Flat</td>
<td>Flat</td>
</tr>
<tr>
<td></td>
<td>RMAT 50k edges 400k vertices</td>
<td>RMAT 150k edges 100k vertices</td>
</tr>
<tr>
<td></td>
<td>272</td>
<td>204</td>
</tr>
<tr>
<td></td>
<td>136</td>
<td>68</td>
</tr>
</tbody>
</table>

### Bar Graphs

- **Flat**
  - Pair of vertices: 272
  - Vertex-Level: 204
- **Chunked**
  - Pair of vertices: 136
  - Vertex-Level: 68

- **Cache**
  - Pair of vertices: 272
  - Vertex-Level: 204
- **Flat**
  - Pair of vertices: 136
  - Vertex-Level: 68
Conclusion

• Jaccard is not a bandwidth bound problem
• Poor candidate for MCDRAM
• We can scale fairly efficiently to make use of hyperthreads
Next Steps

• Adapt State of the art Triangle Counting algorithm to compute Jaccard (uses GraphBLAS)
• Develop a MPI based strong scaling Jaccard algorithm
• Streaming algorithms