Jaccard Coefficients

The College of Engineering
at the University of Notre Dame

What is a Jaccard Coefficient?

- Similarity between neighborhoods of two nodes (V, U):
  - Intersection(u,v) $= |N(V) \cup N(U)|$
  - Union(u, v) $= |N(V) \cap N(U)|$
  - $Jaccard(V, U) = \frac{\text{Intersection(u, v)}}{\text{Union(u,v)}}$
  - $N(V)$ 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(Mlog(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*A=C
  - Values in C correspond to the intersection size
  - Complexity: O(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

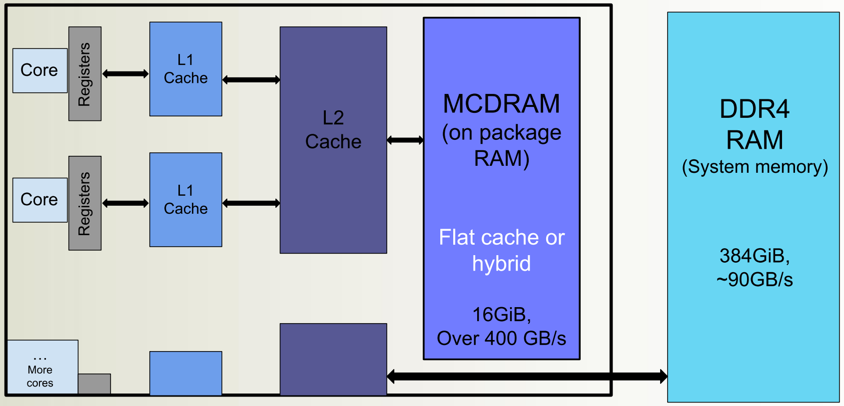
1. Compute each Jaccard twice (U, W) and (W, U)
   – Can be solved by checking ordering
   – Only count if U > W (based on arbitrary ordering)
2. N^2 storage required
   – Only need the number of *unique two-hop paths*
   – Could store results in BST but will add to computational complexity

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

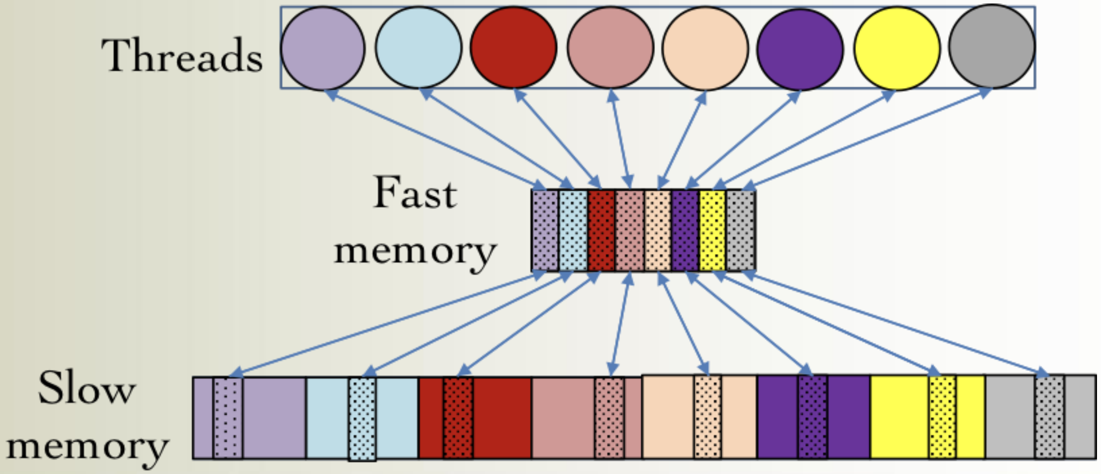
KNL

- MCDRAM can be configured:
  - Cache
  - Flat
  - Hybrid
- Which mode do we want to use if the problem will not fit in MCDRAM?



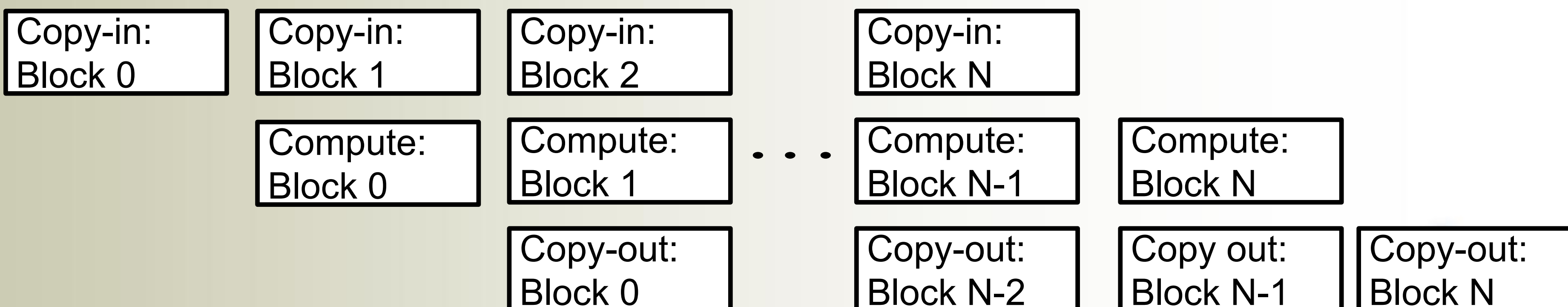
Previous Work: Cache-Oblivious Sorting

- 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

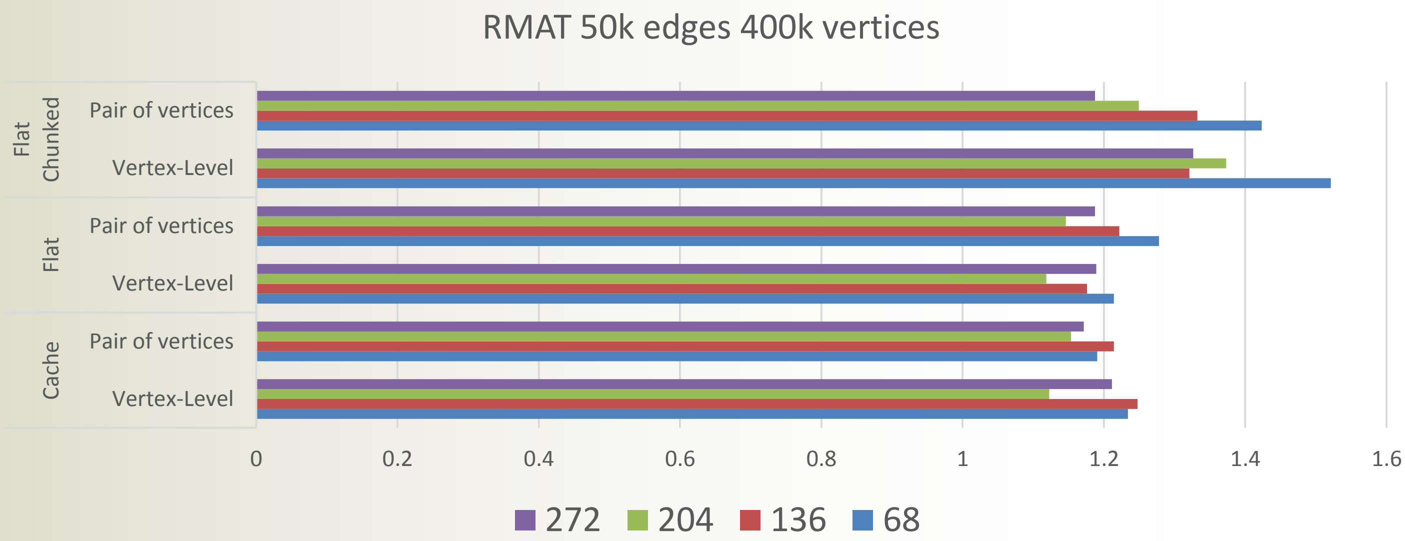
- Run in Flat mode
- Bring portion of data in, operate on it, move next portion in
- Could operate like producer/consumer problem

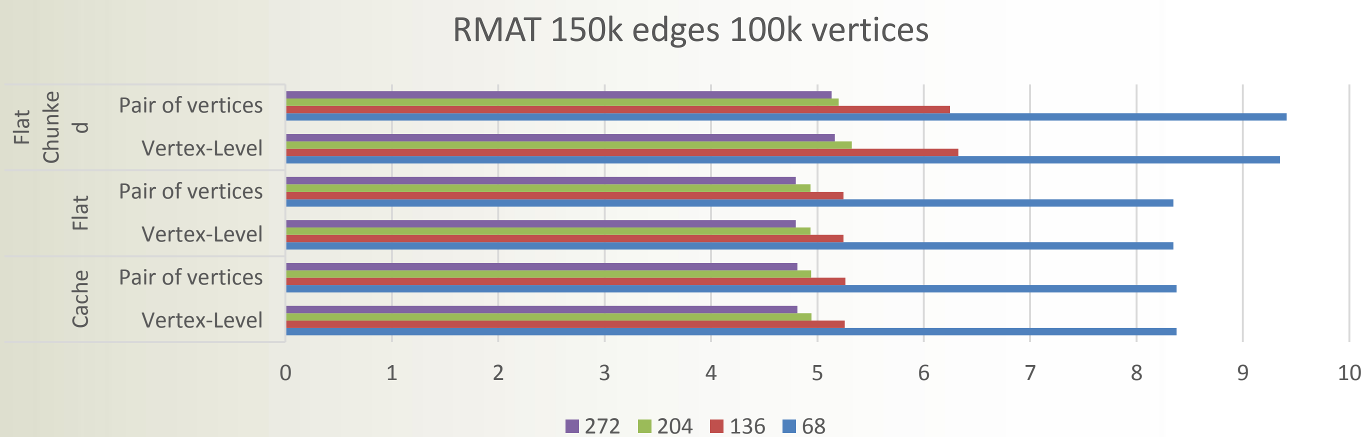


Two Parallel Algorithms

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


RMAT 50k edges 400k vertices
Flat Chunked
Pair of vertices
Vertex-Level
Flat
Pair of vertices
Vertex-Level
Cache
Pair of vertices
Vertex-Level
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
272 204 136 68


RMAT 150k edges 100k vertices
Flat Chunke d
Pair of vertices
Vertex-Level
Flat
Pair of vertices
Vertex-Level
Cache
Pair of vertices
Vertex-Level
0
1
2
3
4
5
6
7
8
9
10
272 204 136 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