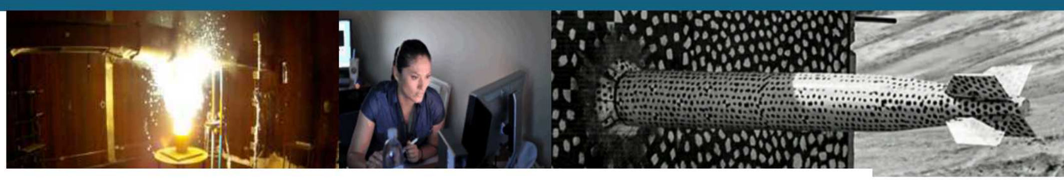
SAND2020-1664C

# Scalable Triangle Counting on Distributed-Memory Systems



**Seher Acer** ✦
Abdurrahman Yasar △
Sivasankaran Rajamanickam ✦
Jonathan Berry ✦
Michael Wolf ✦
Umit Catalyurek △

✦ : Sandia National Labs △ : Georgia Tech

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

# Outline

- Triangle Counting Problem
- Proposed Hybrid-Parallel Algorithm
  - Distributed-Memory Level with 2D Partitioning
  - Shared-Memory Level with 1D Partitioning
- Experimental Results
- Conclusion

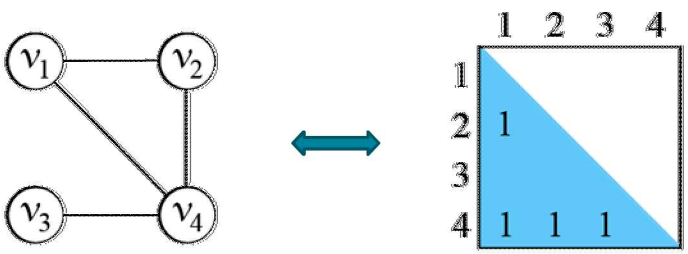
[1] **S. Acer et al.**, "Scalable Triangle Counting on Distributed-Memory Systems", *IEEE HPEC 2019.*

# Triangle Counting Problem

- What is the number of edge triplets $\langle(v_i, v_j); (v_j, v_k); (v_k, v_i)\rangle$ in a given graph?
- Arises in
  - spam detection
  - link recommendation
  - dense neighborhood graph discovery
- An IEEE HPEC Graph Challenge problem
  - https://graphchallenge.mit.edu/
- Graph is undirected

# Triangle Counting Problem

- Problem formulation using linear algebra
- SpGEMM: Sparse matrix-matrix multiplication
- Various methods use
  - adjacency matrix $A$
  - lower and/or upper triangular matrices $L$ and/or $U$
  - incidence matrix $I$
- We only use the lower-triangular matrix $L$ as in [1]



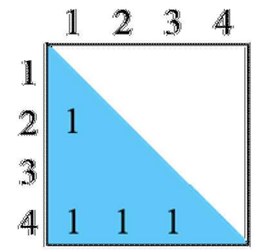

- Number of triangles = sum( $(L \times L)$ .* $L$ )

SpGEMM mask

[1] **M. M. Wolf et al.**, "Fast linear algebra-based triangle counting with KokkosKernels", *IEEE HPEC 2017.*

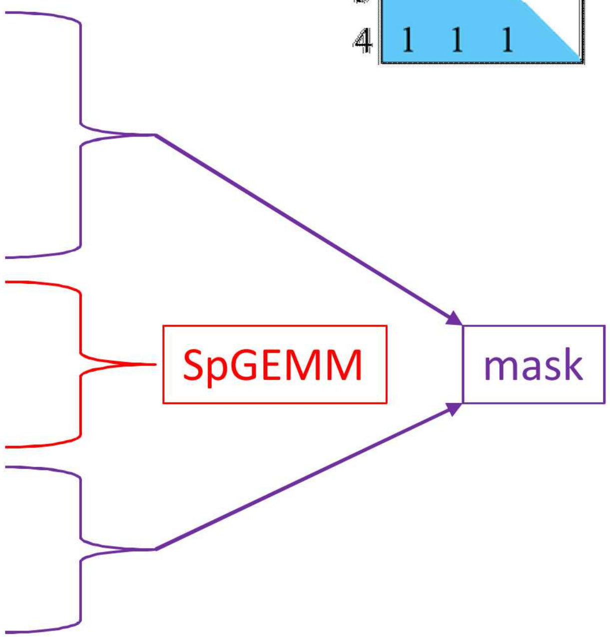
# Triangle Counting

No need for intermediate matrices in $\mathrm{sum}((L \times L) .* L)$




- For each row $i$ of $L$
  - Initialize $c_i$ to 0
  - Create a hashmap $H$
  - For each nonzero $l_{i,j}$ in row $i$
    - Insert $j$ into $H$
  - For each nonzero $l_{i,j}$ in row $i$
    - For each nonzero $l_{j,k}$ in row $j$
      - If $k$ exists in $H$
        - Increment $c_i$

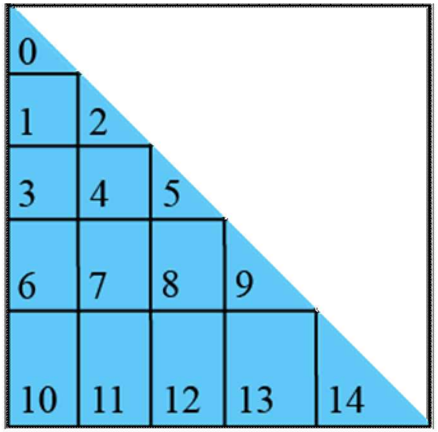
wedge
$v_i - v_j - v_k$
$(i > j > k)$

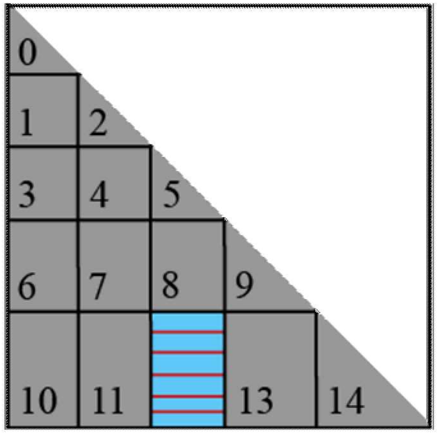



($c_i$: the number of triangles in which $v_i$ is the largest indexed vertex)

# Proposed Hybrid-Parallel Algorithm

- Distributed-memory level
  - MPI
  - 2D Cartesian partition of matrix $L$
  - Each MPI rank gets one block of $L$
- Shared-memory level
  - Cilk/OpenMP tasking
  - 1D partition of the block
  - Each thread gets a subset of consecutive rows in the block

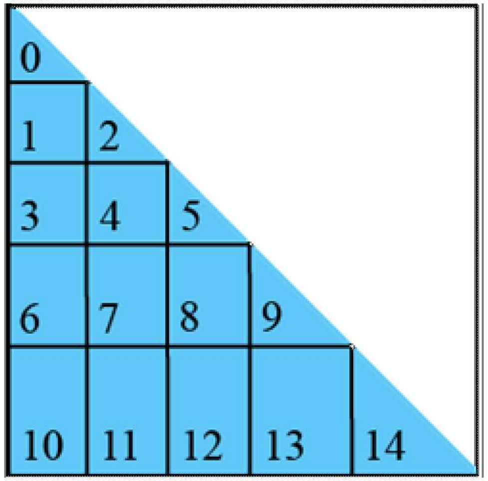


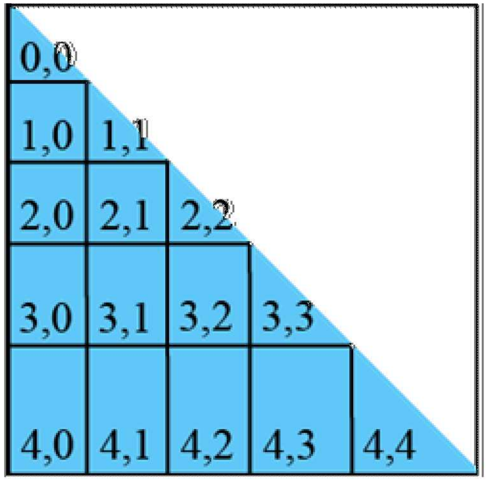



# Proposed Hybrid-Parallel Algorithm

2D Cartesian partitioning [1] of $L$

- $P = Q(Q+1)/2$ MPI ranks
- Balanced number of nonzeros in blocks
  - $avg = nnz(L)/P$
  - $q^{\text{th}}$ row chunk has $q$ blocks
  - $q^{\text{th}}$ chunk has $q \cdot avg$ nonzeros
  - Use the same chunks for columns
- MPI rank at $(q, r)$ owns $L_{q,r}$



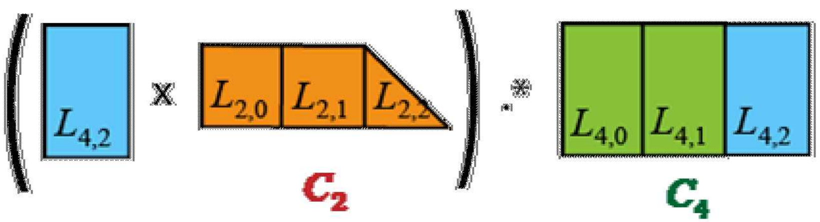



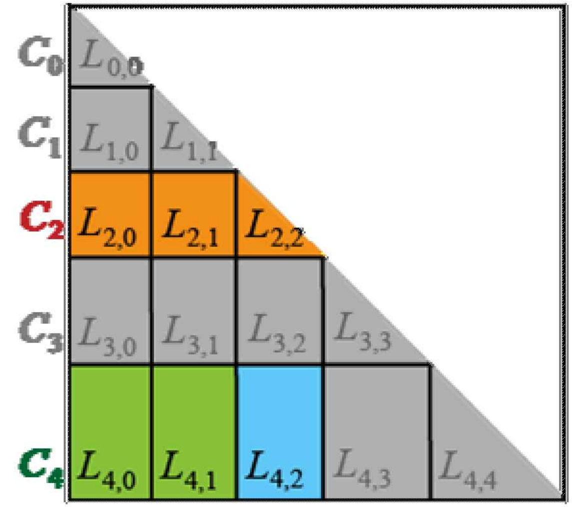

[1] **B. Hendrickson et al.**, "An efficient parallel algorithm for matrix-vector multiplication ", *IJHCS, 1995.*

# Proposed Hybrid-Parallel Algorithm

MPI rank at $(q, r)$

- performs $\left(L_{q,r} \times C_r\right) .* C_q$



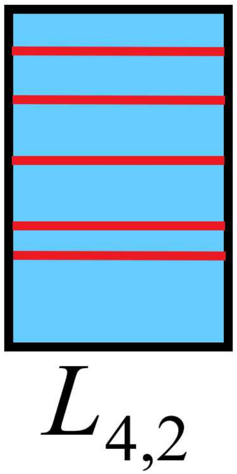

- For $k = 0$ to $r$
  - compute $\text{sum}\left(\left(L_{q,r} \times L_{r,k}\right) .* L_{q,k}\right)$
- needs to receive nonzeros from $L_{r,k}$ and $L_{q,k}$

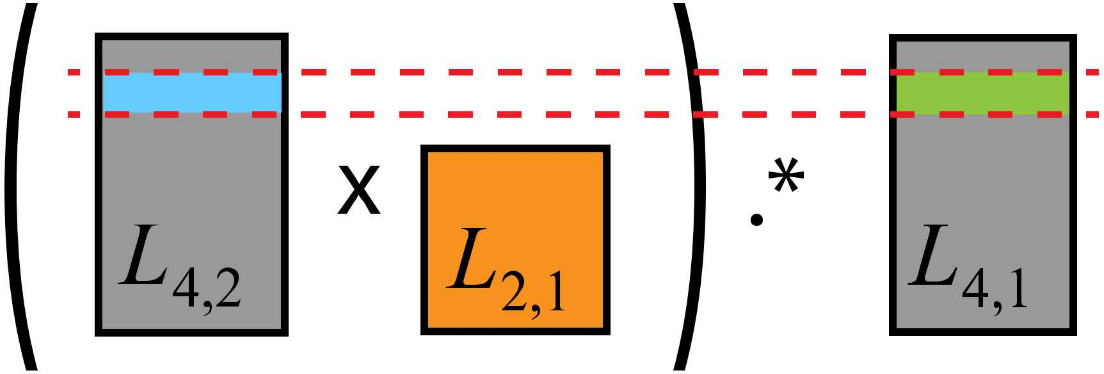
Number of messages per MPI rank = $O(Q) = O(\sqrt{P})$

# Proposed Hybrid-Parallel Algorithm

Shared-memory level [1]:

- 1D row partitioning inside the block
- Balanced number of nonzeros in stripes
- $\#stripes = \#threads \times \alpha$
- $\alpha$ denotes decomposition rate
  - Empirically, $\alpha = 4$ gives the best runtime
- Each thread gets one stripe and computes:




[1] **A. Yasar et al.**, "Fast Triangle Counting Using Cilk", *IEEE HPEC 2018.*

# Experimental Results – Part I

Framework

- Vertices sorted in decreasing order of their degrees
- C++ code with Intel compiler and OpenMPI
- Tested on
  - 28, 36, 45, 55, 66, 78, 91, 105, 120, 136, 153, 171, 190 ranks
- Cilk for shared-memory parallelism
- Two clusters
  - with Skylake nodes: 2 Intel Xeon Platinum 8160 CPUs
    - (4x12): 4 MPI ranks per node, 12 threads per MPI rank
  - with Broadwell nodes: 2 Intel Xeon E5-2695 CPUs
    - (2x18): 2 MPI ranks per node, 18 threads per MPI rank

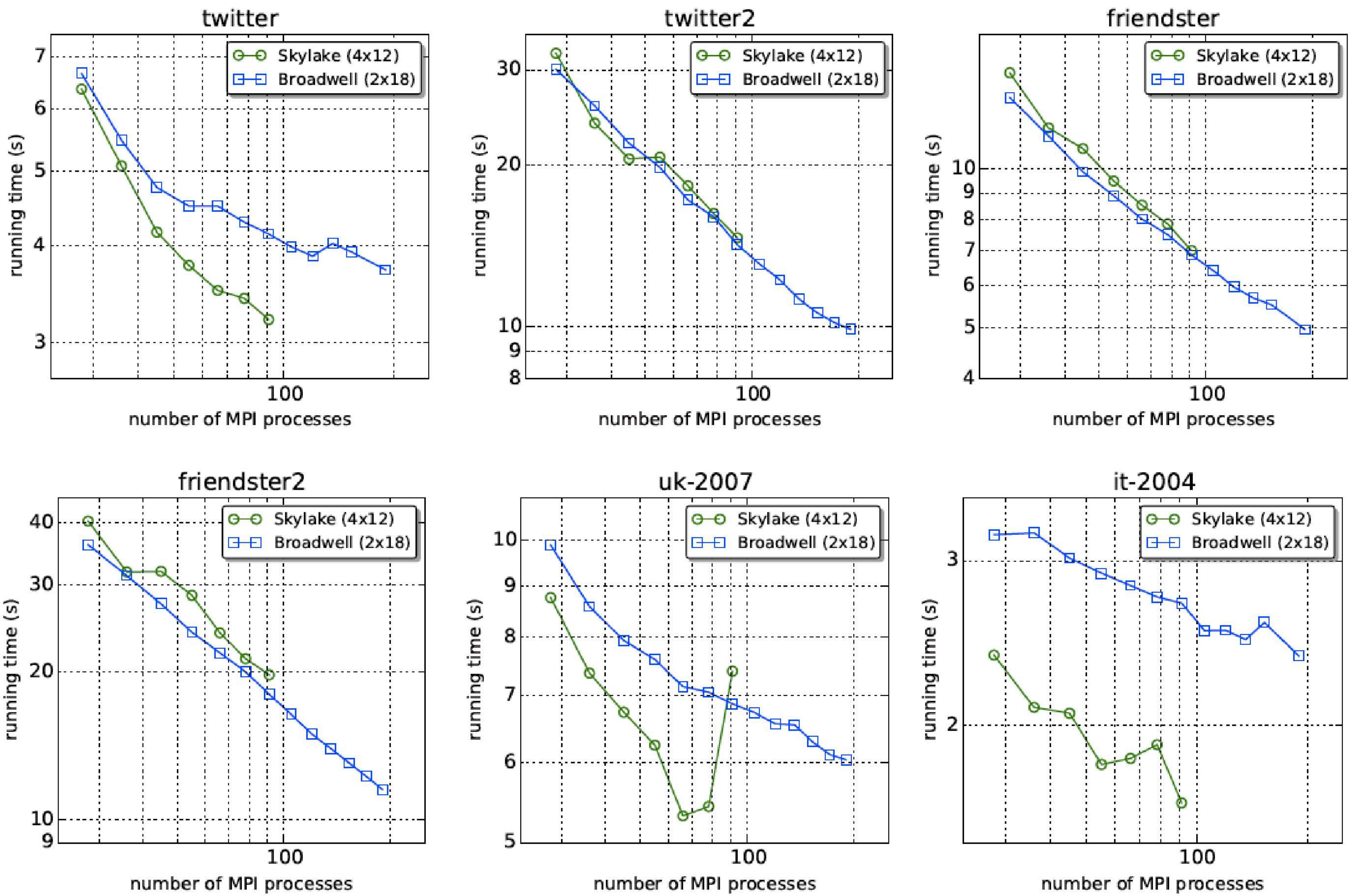
# Experimental Results - Part I

## Dataset – Part I

| graph | #vertices | #edges | #triangles |
|---|---|---|---|
| it-2004 [1] | 41,291,594 | 1,027,474,947 | 48,374,551,054 |
| twitter [1] | 61,578,414 | 1,202,513,046 | 34,824,916,864 |
| Twitter2 [2] | 103,809,266 | 3,107,433,379 | 151,582,758,659 |
| Friendster [1] | 65,608,366 | 1,806,067,135 | 4,173,724,142 |
| Friendster2 [2] | 131,216,732 | 3,604,811,068 | 16,803,555,478 |
| uk-2007 [3] | 105,896,555 | 3,301,876,564 | 286,701,284,103 |

[1] **T. A. Davis and Y. Hu**, "The University of Florida Sparse Matrix Collection", *ACM TOMS, 2011*.
[2] **G. Slota et al.,** "Scalable Generation of Graphs for Benchmarking HPC Community-Detection Algorithms", *SC19*.
[3] **P. Boldi and S. Vigna**, "WebGraph Datasets: Laboratory for algorithmics", 2018.

# Experimental Results – Part I



# Experimental Results – Part I

Comparison against other MPI and Cilk-based approaches

| | *GC'17 Champion* | *GC'18 Champion* | |
|---|---|---|---|
| | **Our method** MPI+Cilk | **Pearce [1]** MPI | **Yasar et al [2]** Cilk | **Tom&Karypis [3]** MPI |
| twitter | **3.21 s.** 1092 cores | 8.52 s. x2.6 6144 cores | 28.35 s. x8.8 48 cores +2HT | 18.52 s. x5.8 169 cores |
| friendster | **4.95 s.** 3420 cores | - | 18.55 s. x3.7 48 cores +2HT | 27.51 s. x5.5 169 cores |

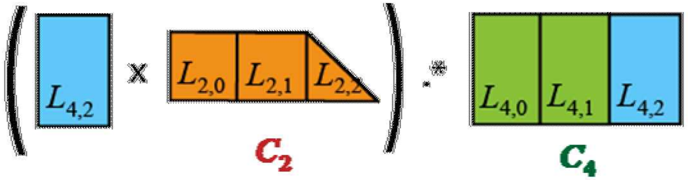
[1] **R. Pearce**, "Triangle counting for scale-free graphs at scale in distributed memory", *IEEE HPEC 2017.*
[2] **A. Yasar et al.,** "Fast triangle counting using Cilk", *IEEE HPEC 2018.*
[3] **A. S. Tom and G. Karypis**, "A 2D Parallel Triangle Counting Algorithm for Distributed-Memory Architectures", *ICPP, 2019.*

# Experimental Results – Part II

- Largest public graph: WDC [1]
  - 3.5B vertices, 112B edges, 9.6T triangles
- Tested on
  - 105, 136, and 171 MPI ranks
- Shared-memory parallelism
  - Cilk
  - OpenMP
- The cluster with Broadwell nodes
  - (1x36): 1 MPI rank per node, 36 threads per MPI rank

[1] Web Data Commons webgraph. http://webdatacommons.org, 2012.

# Experimental Results – Part II




- For small graphs
  1. computation starts after receiving $C_r$ and $C_q$
  2. use dense hashmap (the fastest)
- For the large graph, memory is a problem
  1. interleaved computation & communication
     - For $k = 0$ to $r$
       - allocate memory and receive $L_{r,k}$ and $L_{q,k}$
       - compute $\text{sum}\left(\left(L_{q,r} \times L_{r,k}\right) .* L_{q,k}\right)$
       - deallocate memory used for $L_{r,k}$ and $L_{q,k}$
  2. use sparse hashmap (the most memory-efficient)

# Experimental Results – Part II

- Largest public graph WDC
- Baseline algorithm [1]: 808 s. on 256 (x24) nodes

| #MPI ranks | Runtime (s) | |
|---|---|---|
| | OpenMP | Cilk |
| 105 | 582 | 559 |
| 136 | 522 | 492 |
| 171 | 497 | 481 |

- %40 faster using almost same number of cores
  - 256x24=6144 vs 171x36=6156

[1] **R. Pearce**, "Triangle counting for scale-free graphs at scale in distributed memory", *IEEE HPEC 2017*.

# Conclusion

- A hybrid-parallel algorithm for triangle counting
- 2D Cartesian partitioning among MPI ranks
- 1D row partitioning among Cilk/OpenMP threads
- Fastest known runtime on twitter: 3.21 seconds
  - 2.6x faster than the baseline algorithm
- Fastest known runtime on WDC: 481 seconds
  - %40 faster than the baseline algorithm

✓ 2D Cartesian partitioning + hybrid approach