Pacific Northwest NATIONAL LABORATORY

Graph-based proxy applications and derivative benchmarking on Ookami

February 11, 2022
Ookami User Group Meeting

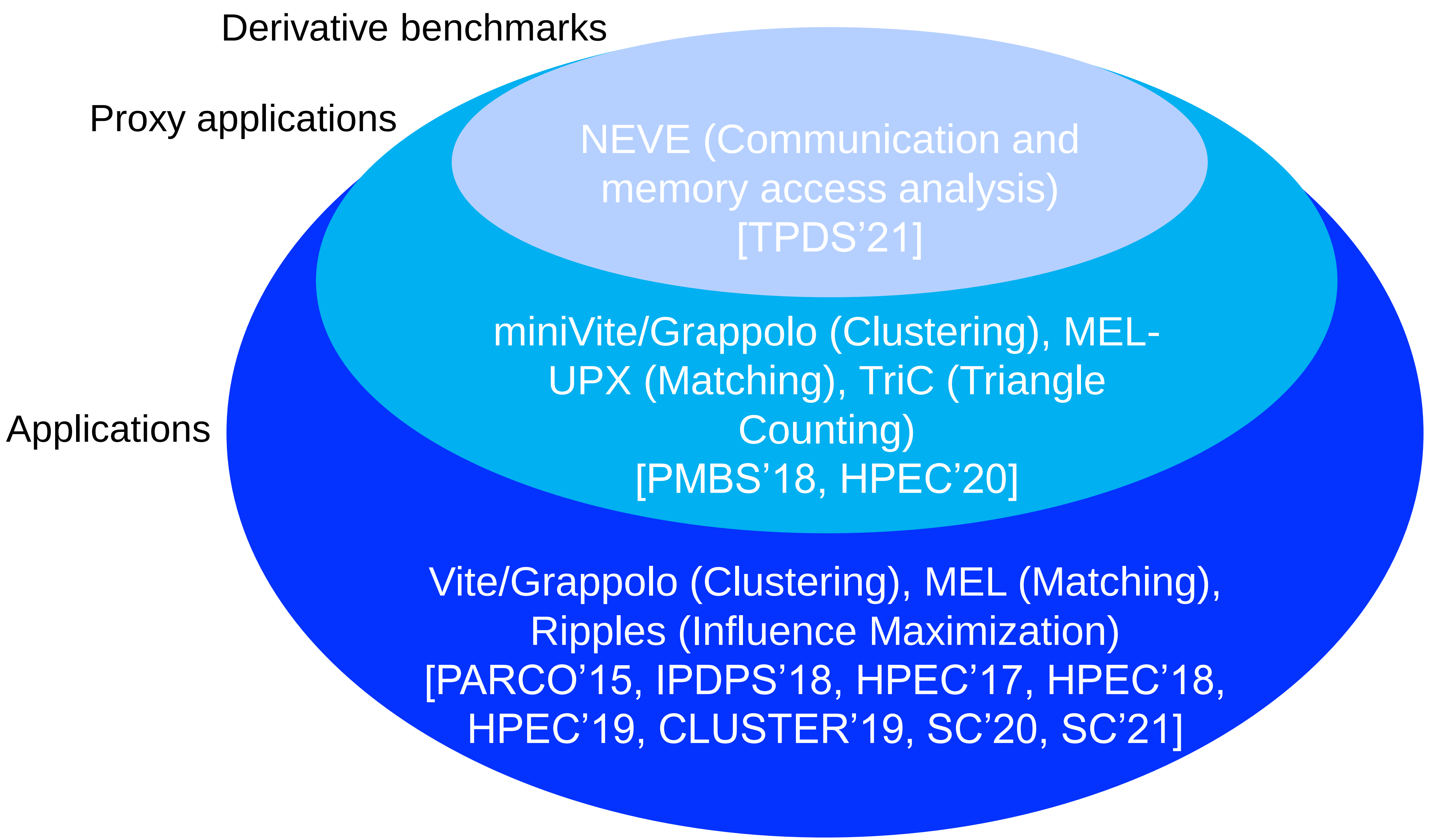
Sayan Ghosh
Computer Scientist, PNNL

U.S. DEPARTMENT OF ENERGY BATTELLE

PNNL is operated by Battelle for the U.S. Department of Energy

Pacific Northwest NATIONAL LABORATORY

Graph analytics codes

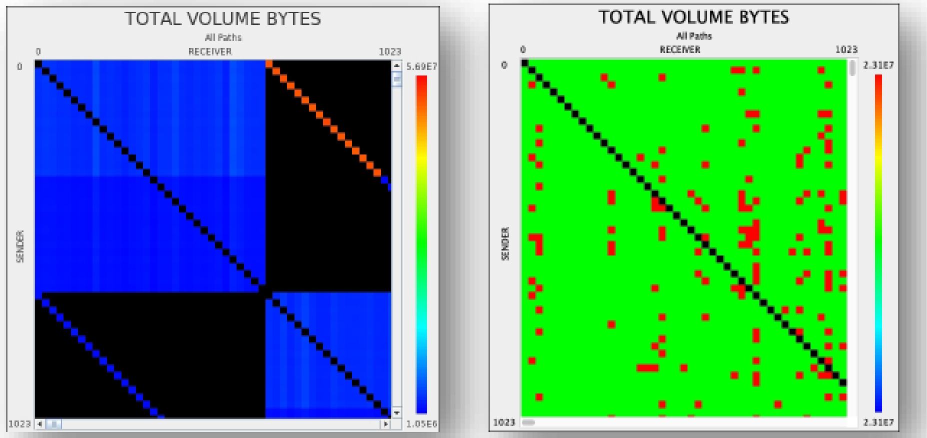


Graph algorithms

- Combinatorial (graph) algorithms are key enablers in data analytics
  - Graph coloring, matching, community detection, pattern, centralities, traversals, etc.
- Relatively *less* computation and *more* memory accesses
  - Graph codes on accelerators mainly exploits the b/w, ALUs are relatively underutilized (many algorithms have 0 FLOPS)
  - Limited vectorization advantage
- Graphs are multifarious, distributed-memory poses challenges
  - Asynchronous, irregular and adversarial communication patterns
  - Network contention

```
Input: G(V,E), s ∈ V
Q.enqueue((visit(s))
while (!Q.empty()):
  u = Q.dequeue()
  for v in neighbor(u):
    if (!visited(v)):
      Q.enqueue((visit(v))
```

```
Input: G(V,E)
for u in V:
  for v in neighbor(u):
    visit(v)
```



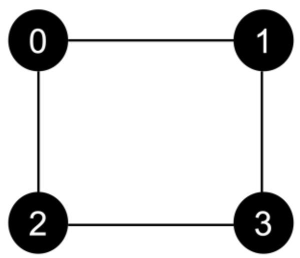
Pair-wise communication volume for BFS (left) and Graph neighborhood (right) for same graph

Communication is shown as a heat map of bytes/process pair, black is 0 bytes

Graph500 or traversal-based algorithms are not necessarily representative use cases!

Derivative Benchmark: Analyzing Graph Memory Accesses via simple kernels

Graph

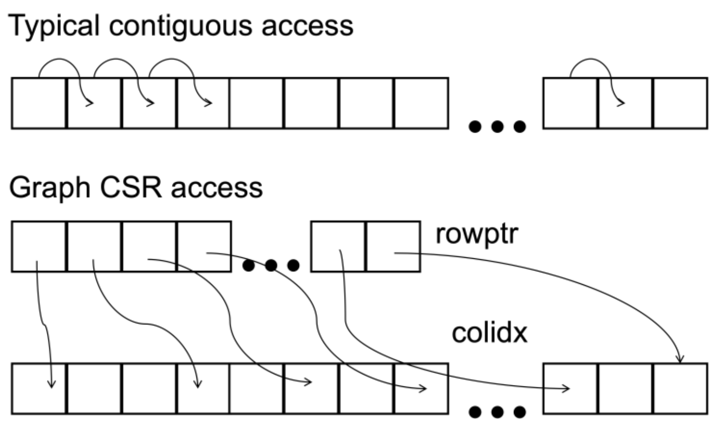


Adjacency matrix

	0	1	2	3
0	0	1	1	0
1	1	0	0	1
2	1	0	0	1
3	0	1	1	0

CSR

rowptr: 0 2 4 6 9
colidx : 1 2 0 3 0 3 1 2



Input: $G = (V, E)$, (undirected) graph G.

1: **for** $v \in V$ **do**
2: **for** $u \in \text{adj}(v)$ **do** {Neighbors of v}
3: {Perform some work with $\{v, u\}$}

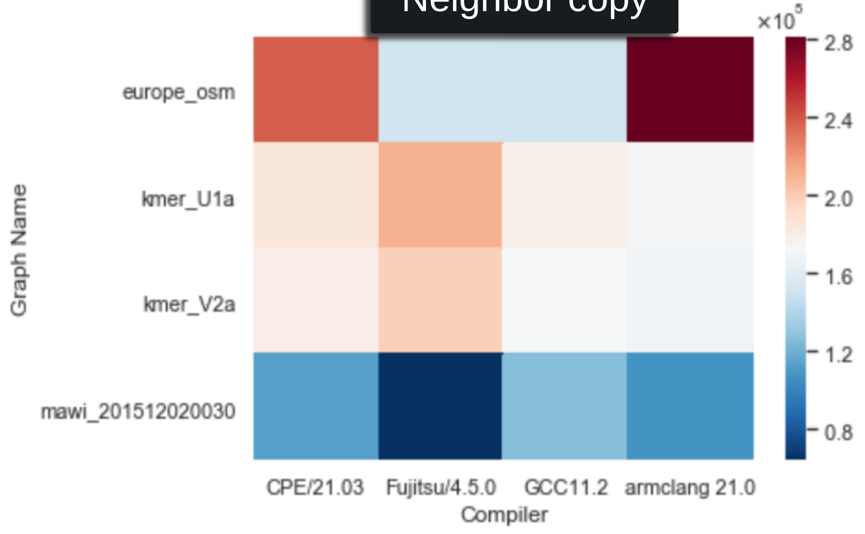
Scanning the neighborhood of a vertex in a graph is common

Disparity in maximum and average #edges impacts performance (unstructured parallelism)

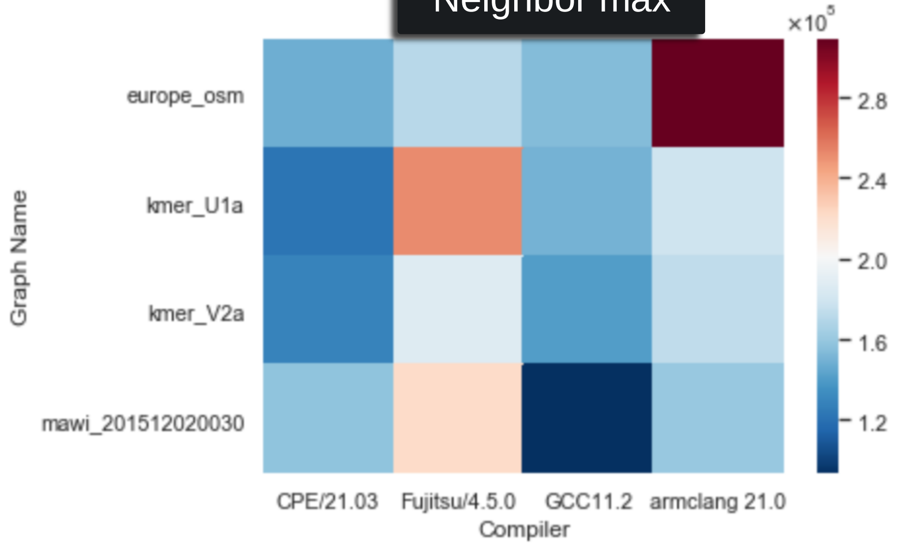
Different graphs (100M+ edges) on different compilers

Red spectrum is good, blue/white not so much

Neighbor copy

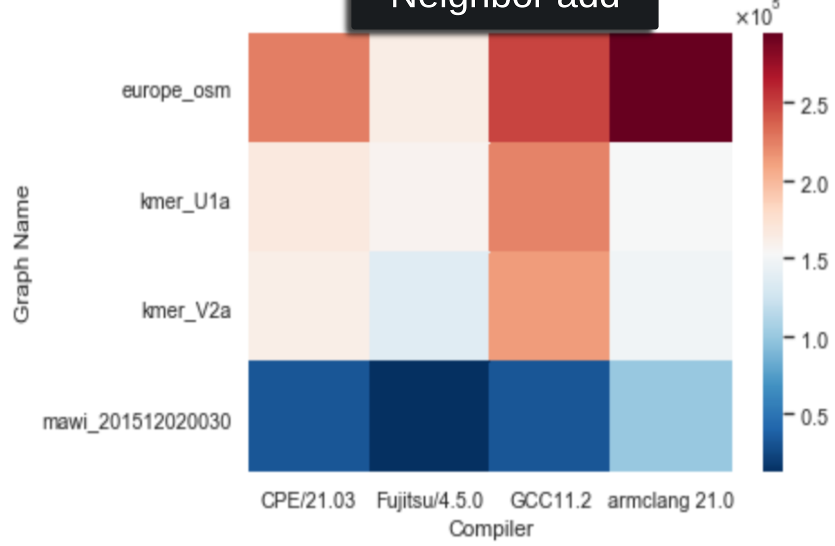


Neighbor max



- Reporting TEPS (higher the better)
- Detect issues with systems and runtimes
- Sandbox for building graph applications

Neighbor add

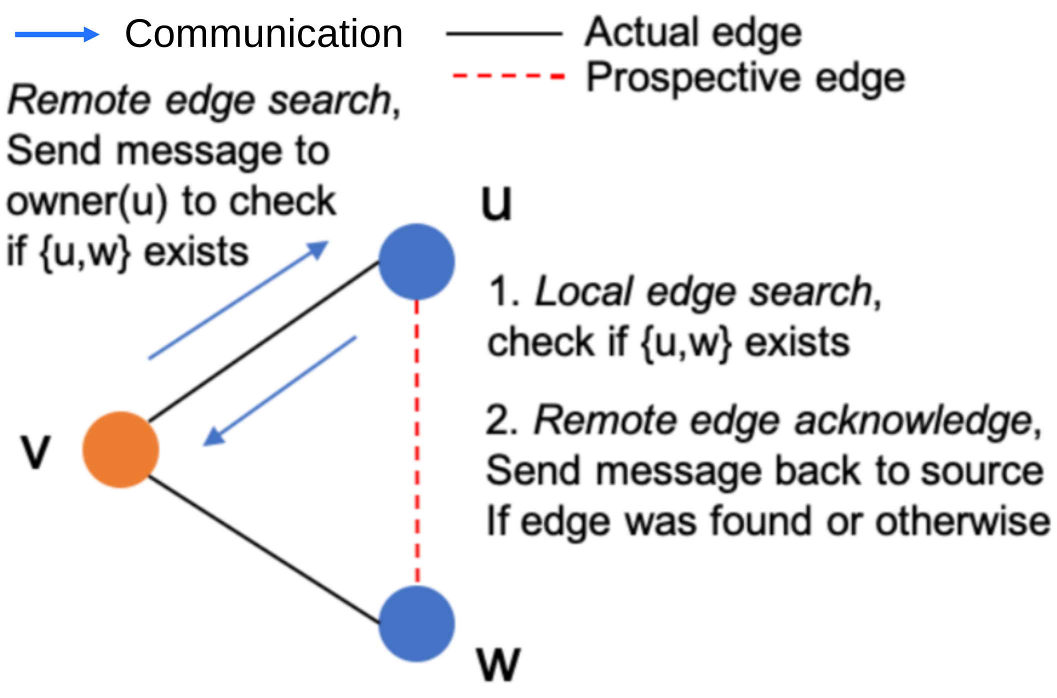


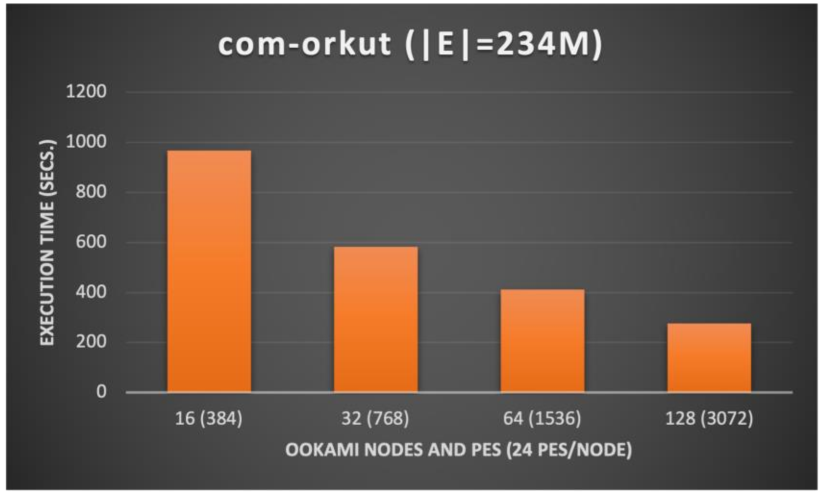
- Column-wise variation depict differences in compilers
- Row-wise variation depict differences across graphs (structure is impeding parallelism, investigate)
- Will try ZFILL

* Results from Yan Kang, PSU

Proxy: Triangle counting by exploiting graph structure

- Developed several variants of distributed exact graph triangle counting
  - Simple formulation – exploits vertex-centric distributed graph structure – process-based
  - Options to suspend and resume work on a vertex based on a customizable buffer
    - ✓ Throttle messages and limit neighborhood size
  - Different communication models: MPI send/recv, RMA, neighborhood collectives
- This code is a pilot before moving on optimizing other apps – clustering/matching, etc
- Preliminary results on Ookami shows about 3.5x speedup relative to 8x nodes





Ghosh S, Halappanavar M. TriC: Distributed-memory Triangle Counting by Exploiting the Graph Structure. In 2020 IEEE High Performance Extreme Computing Conference (HPEC) 2020 Sep 22 (pp. 1-6). IEEE. [code: https://github.com/sg0/tric]

Summary

- Most of our codes are distributed and have a startup problem (more nodes to run – major communication overhead!)
  - Mitigating it with fixed-buffer and suspend-restart mechanism
  - Buffer size is a trade-off (too large OOM, too small more iters)
  - More processes and less threads lead to better results (not much on-node parallelism – could be more) – using 12-24 PEs per node
  - Investigate communication avoiding heuristics and extract more bandwidth from node
  - Impact of SVE?
- armclang (21.0) performance seems to be generally better
- Software setup/building was straightforward – thanks for the support and active Slack channel!