Graph-based proxy applications and derivative benchmarking on Ookami

February 11, 2022
Ookami User Group Meeting

Sayan Ghosh
Computer Scientist, PNNL
Graph analytics codes

Vite/Grappolo (Clustering), MEL (Matching), Ripples (Influence Maximization)  
[PARCO’15, IPDPS’18, HPEC’17, HPEC’18, HPEC’19, CLUSTER’19, SC’20, SC’21]

miniVite/Grappolo (Clustering), MEL-UPX (Matching), TriC (Triangle Counting)  
[PMBS’18, HPEC’20]

NEVE (Communication and memory access analysis)  
[TPDS’21]

Derivative benchmarks

Proxy applications

Applications
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
Derivative Benchmark: Analyzing Graph Memory Accesses via simple kernels

- Scanning the neighborhood of a vertex in a graph is common
- Disparity in maximum and average #edges impacts performance (unstructured parallelism)
- Reporting TEPS (higher the better)
- Detect issues with systems and runtimes
- Sandbox for building graph applications

- 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

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!