Exploiting High Bandwidth Memory for Graph Algorithms

George M. Slota[1] Sivasankaran Rajamanickam[2]
Cynthia Phillips[2] Jonathan Berry[2]

[1]Rensselaer Polytechnic Institute, [2]Sandia National Labs
slotag@rpi.edu, srajama@sandia.gov, jberry@sandia.gov, caphill@sandia.gov

SIAM PP 8 March 2018

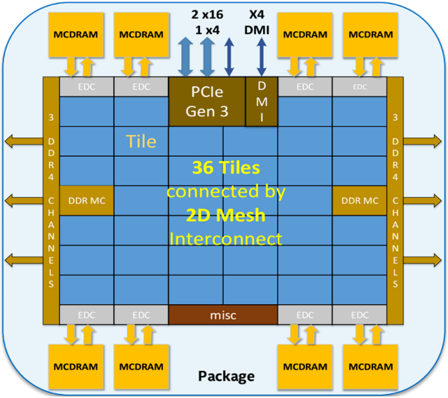
Intro and Overview of Talk

- **Ongoing trend**: expansion of memory hierarchy for increased CPU throughput – e.g., high-bandwidth memory (HBM) layer on current generation Intel Xeon Phis (Knight's Landing)
- Can we explicitly design graph computations to effectively utilize this layer?
- We explore a work chunking approach that iteratively brings in pieces of a large graph to perform local updates in HBM – we specifically look at the *label propagation* algorithm. We find:
  - **Chunking has minimal impact on solution quality**
  - **Chunking can also decrease time to solution**

Primary assumption: the graphs being processed are too large to fit entirely within MCDRAM

Intel Knight's Landing (KNL)

68-72 cores with High Bandwidth Multi-channel DRAM (MCDRAM)



Source:Intel

Stream Triad Bandwidths

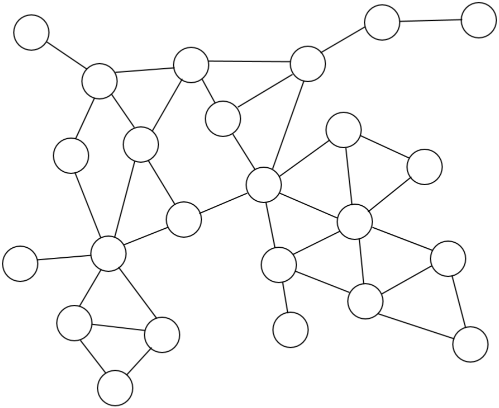
- DDR: 90 GB/s
- MCDRAM: 450 GB/s

Multiple MCDRAM modes

- Cache Mode
- Flat Mode
- Hybrid Mode

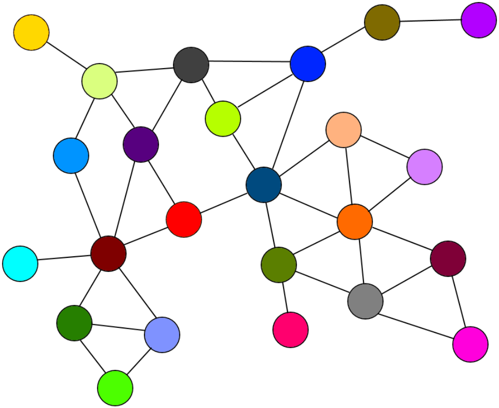
Label Propagation

- Randomly label with $n = \#verts$ labels



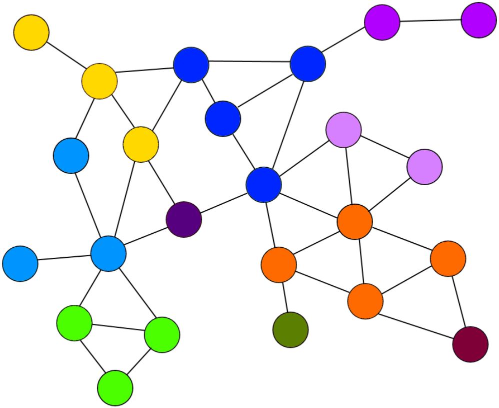
Label Propagation

- Randomly label with $n = \#verts$ labels



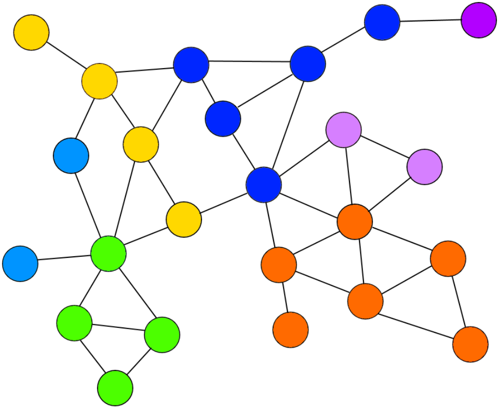
Label Propagation

- Randomly label with $n = \#verts$ labels
- Iteratively update each $v \in V(G)$ with max per-label count over neighbors with ties broken randomly



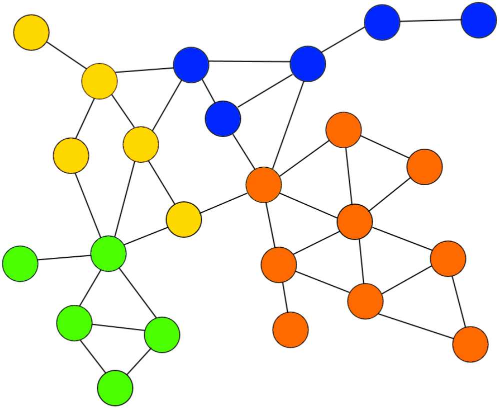
Label Propagation

- Randomly label with $n = \#verts$ labels
- Iteratively update each $v \in V(G)$ with max per-label count over neighbors with ties broken randomly



Label Propagation

- Randomly label with $n = \#verts$ labels
- Iteratively update each $v \in V(G)$ with max per-label count over neighbors with ties broken randomly
- Algorithm completes when no new updates possible; in large graphs, fixed iteration count



Why Label Propagation?

- Iterative vertex updates – prototypical of many other graph computations
- Wide usage – community detection, partitioning, other unsupervised learning problems
- Nondeterministic algorithm by design – solution quality can vary based on processing methodology
- Straightforward to implement via work chunking

Multilevel Memory Label Propagation

via work chunking

$L \leftarrow$ LPChunking($G(V, E)$, $\mathbf{C}_{num}$, $\mathbf{C}_{iter}$)
for all $v \in V$: $L(v) \leftarrow id(v)$ ▷ Initialize labels as vertex ids
while *at least one L(v) updates* **do**
  for $c = 0 \cdots (C_{num} - 1)$ **do**
    $V_c \leftarrow$ **Chunk**(c, V), $E_c \leftarrow \langle v, u \rangle \in E : v$ **or** $u \in V_c$
    for $iter = 1 \ldots C_{iter}$ **while** *one L(v)* : $v \in V_c$ *updates* **do**
      for all $v \in V_c$ **do in parallel** ▷ Random order
        $Counts \leftarrow \emptyset$ ▷ Hash table
        for all $\langle v, u \rangle \in E_c$ **do**
          $Counts(L(u)) \leftarrow Counts(L(u)) + 1$
        $NewLabel \leftarrow$ Max($Counts(\ldots)$)
        if $NewLabel \neq L(v)$ **then**
          $L(v) \leftarrow NewLabel$

Chunking Considerations

Primary chunking variables

- Number of total chunks (C_{num})
- Work iterations performed on each chunk (C_{iter})

How to determine data per chunk?

- Block methods (vertex block, edge block)
- Randomization
- Explicit partitioning

How to transfer chunked data?

- All threads transfer, then all threads work
- Overlap transfer of C_{i+1} with work on C_i
- Vary number of work/transfer threads to ensure balance

Algorithmic Variants

Baseline Cache

- Baseline implementation running in cache mode

Baseline Hybrid

- Baseline implementation with hash table allocated in MCDRAM
- Graph structure and other data handled by MCDRAM cache

Chunk-HBM

- All data explicitly allocated in MCDRAM
- Per-chunk graph structure transfered into MCDRAM
- All vertex labels static in MCDRAM

Experimental Setup

Test System and test graphs

Test System: *Bowman* at Sandia Labs – each node has a KNL with 68 cores, 96 GB DDR, and 16 GB MCDRAM

Test Graphs:

Network	n	m	$davg$	$dmax$	D̃
LiveJournal	4.8 M	69 M	18	20 K	18
Friendster	66 M	1.8 B	27	5.2 K	34
Twitter	52 M	2.0 B	37	3.7 M	19
Host	89 M	2.0 B	22	3.4 M	23
uk-2007	105 M	3.3 B	31	975 K	82
wBTER_50	50 M	1.2 B	24	110 K	12
wBTER_100	100 M	2.4 B	24	135 K	12

How does chunking impact solution quality?

Convergence and solution quality

For label propagation and community detection algorithms in general

Defining convergence

- *True convergence*: no more label updates can occur
- *Looser criteria*: fixed iterations, some modularity gain or change, number of labels, others

We run to true convergence when possible, but fix iterations to enable a parametric study of chunking variables.

Defining solution quality

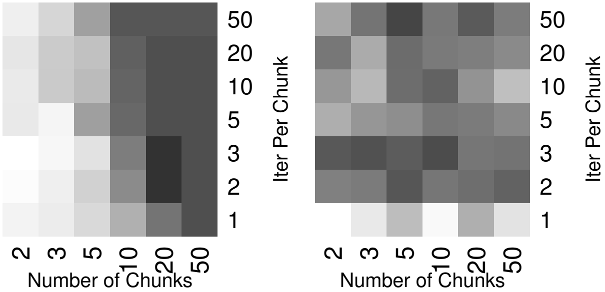
- Standard metrics when no ground truth exists: *modularity*, *conductance*, among many others
- When ground truth exists: *normalized mutual information* (NMI) and related measurements

Despite some observed flaws with their usage, we select the standard measurements of *modularity* and *NMI*.

Chunking parameters

Evaluating impact of number of chunks and iterations per chunk

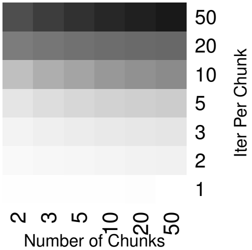
- Heatmaps of iterations to convergence (left) and impact on final modularity (right) – lighter is better
- About 5$\times$ increase in iterations captured in left plot and 2% total modularity change in right plot
- While chunking increases iterations to convergence, it has minimal impact on final solution quality (and actually improves it in several instances – LiveJournal, Host, wBTER)

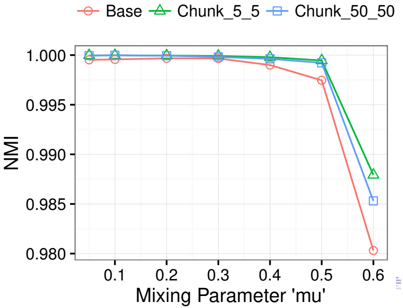


Chunking parameters

Lancichinetti-Fortunato-Radicchi (LFR) benchmark

- Ran same parametric tests on LFR benchmark ($n = 10,000$, $k = 15$, $maxk = 500$, $t1 = 2$, $t2 = 1$, $\mu = 0.05 \ldots 0.6$)
- Heatmap of iterations to convergence (left) and NMI versus baseline (right)
- Similar takeaways to real-world test instances



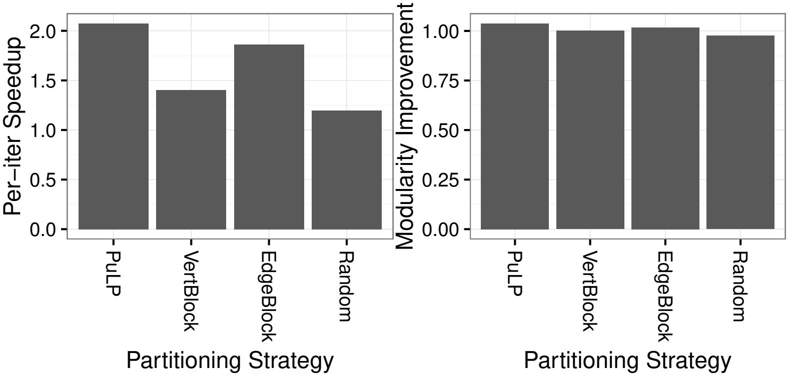


Can HBM chunking improve time to solution?

Consideration 1: Partitioning Methodology

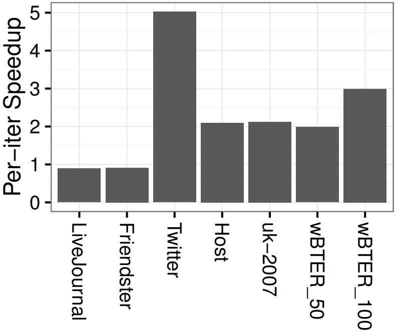
5 iterations per chunk, minimum number of chunks possible (∼5), 40 iterations

- Effects of partitioning method on per-iteration speedup vs. baseline (left) and modularity (right)
- Explicit partitioning demonstrates largest improvements, but at the obvious cost of computing the partition



Consideration 2: Overlapping Communication

- Average speedups across all partitioning methods while overlapping communication
- Note: when overlapping, we double the number of chunks; this can lead to greater than 2× relative speedup due to cache effects on graph data and hash tables



Overall: Cache vs. Hybrid vs. Flat modes

Best times (in seconds) in each mode for each graph for 40 iter or convergence

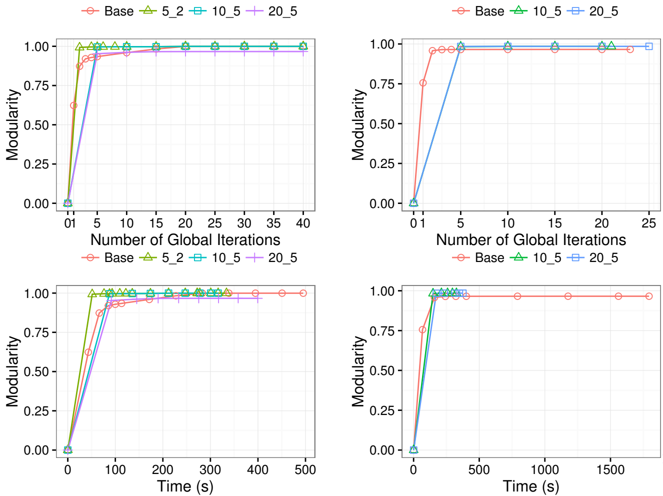
Network	Cache	Hybrid	Flat	Method
LiveJournal	33	29	25	P-OL
Friendster	495	337	333	VB
Twitter	1,793	871	242	P-OL
Host	2,447	2,086	712	EB-OL
uk-2007	1,981	1,241	783	P-OL
wBTER_50	577	474	225	VB-OL
wBTER_100	1,602	491	435	EB-OL

Methods: **VB**: Vertex Block; **EB**: Edge Block; **P**: PuLP Partitioning; **-OL** with overlapping communication

Time and modularity vs. iterations

Per-iteration time and total time doesn't tell the whole story

Friendster (left) and Twitter (right) for modularity vs. iterations (top) and time per iteration (bottom). Baseline and $C_{num}\_C_{iter}$.



Discussion: Generalization

To other vertex programs on KNLs with HBM

- Tested chunked versions of PageRanks and K-cores
- Speedups still there but much less – under 25%
  - Hash table for label propagation is likely just extremely ill-performant in cache mode; benefits most from memory considerations
- Minimal impact on solution quality for PR (for K-cores, we run to true convergence)

GPU and SSD-based graph processing

- Note: biggest general takeaway is running multiple *local* iterations doesn't impact solution quality
- So limited-memory GPUS and large-scale processing with SSD arrays might consider similar approaches

Distributed processing

- Equivalence to only communicating every *nth* iteration

Conclusions
and future work

- Chunking minimally affects solution quality of label propagation, but can increase the number of iterations required for a given "quality"
- Explicit handling of HBM generally improves per-iteration timing and can improve time-to-solution in select instances
- Future work:
  - Further explore generalizations to other vertex programs
  - Multi-tiered chunking – hold key vertices in HBM and update every iteration

Paper to appear in IPDPS 2018

`www.gmslota.com`, `slotag@rpi.edu`