Exploiting in-Hub Temporal Locality in SpMV-based Graph Processing

ICPP 2021 / August 9-12
DOI: 10.1145/3472456.3472462
https://blogs.qub.ac.uk/GraphProcessing/

Mohsen Koohi Esfahani
mkoohiesfahani01@qub.ac.uk

Peter Kilpatrick
p.kilpatrick@qub.ac.uk

Hans Vandierendonck
h.vandierendonck@qub.ac.uk
Outline

• Introduction
• Pull vs Push
• Is Pull A Suitable Direction?
• iHTL: in-Hub Temporal Locality
• iHTL Graph Structure
• SpMV in iHTL
• Evaluation
• Conclusion
Introduction

• SpMV Graph Analytics
  – Updating data of a vertex by considering data of its in-neighbours
  – Applications: HITS, Graph Neural Networks, Belief Propagation,…

• Power-law Graphs
  – Social networks, Web graphs, …
  – A very small fraction of vertices (known as hubs), are connected to a large fraction of edges
  – 1% of vertices with maximum degrees are incident to 23% - 77% of edges

| Dataset         | Hubs’ Edges (%) |
|-----------------|-----------------|
| Twitter         | 44.4            |
| Friendster      | 23.9            |
| SK-Domain       | 77.2            |
| WebCC           | 60.5            |
| UK-Delis        | 77.6            |
| UK-Union        | 73.0            |
| UK-Domain       | 45.2            |
| ClueWeb09       | 50.4            |
**Pull**

SpMV in pull direction

| Input: $G(V, E), D^{i-1}, D^i$ |
|----------------------------------|
| for $v \in V$ do |
| sum = 0; |
| for $u \in N^-_v$ do |
| sum += $D^{i-1}[u]$; |
| $D^i[v] = sum$; |

- Random memory accesses:
  - Reading old data ($D^{i-1}$) of in-neighbours
  - Cache is dedicated to old data of source vertices
- Sequential accesses:
  - Writing new data ($D^i$) of a vertex
- No race conditions, i.e., easy and fast parallelization

**Push**

SpMV in push direction

| Input: $G(V, E), D^{i-1}, D^i$ |
|----------------------------------|
| for $v \in V$ do |
| for $u \in N^+_v$ do |
| $D^i[u] += D^{i-1}[v]$; |

- Random memory accesses:
  - Updating new data ($D^i$) of out-neighbours
  - Cache is dedicated to new data of destinations
- Sequential accesses:
  - Reading old data of a vertex ($D^{i-1}$)
- Requires protection of the new data ($D^i$) from concurrent updates
Is Pull A Suitable Direction?

- Hubs experience very high cache miss rates
  - Even after relabeling by locality-optimizing reordering algorithms such as SlashBurn\(^1\), GOrder\(^2\), and Rabbit-Order\(^3\)
- A massive amount of vertex data is pulled into the cache by pull processing of an in-hub that
  - Displaces much of the cache contents and
  - Reduces the opportunity for future reuse

---

1. Lim, Kang, Faloutsos. SlashBurn: Graph compression and mining beyond caveman communities, IEEE TKDD, 2014.
2. Wei, Yu, Lu, Lin. 2016. Speedup graph processing by graph ordering, SIGMOD ’16.
3. Arai, Shiokawa, Yamamuro, Onizuka, Iwamura. Rabbit order: Just-in-time parallel reordering for fast graph analysis, IPDPS’16.
iHTL: in-Hub Temporal Locality

• Incoming edges to in-hubs have
  – A very large number of source vertices, and
  – A small number of destination vertices

• Cache cannot satisfy the large number of source vertices in this sub-graph

• iHTL states that for incoming edges to in-hubs:
  – Cache can accommodate the small number of destinations (in-hubs), so
  – Cache should be dedicated to destinations, and not source vertices. In other words,
  – Push is the cache-compatible direction for processing incoming edges to in-hubs
iHTL Graph Structure

- iHTL divides vertices into 3 types:
  - **in-Hubs** that are identified by investigating graph structure
  - Vertices With Edges to in-Hubs (**VWEH**)
  - Fringe Vertices (**FV**) with no edges to in-hubs

- iHTL divides graph into 3 major parts:
  - A number of **Flipped Blocks** contain incoming edges to in-hubs
  - A **Sparse Block** contains edges to non-hubs
  - A **Zero Block** contains no edges
SpMV in iHTL

- iHTL process graph in 2 steps:
  - Processing **flipped blocks in push** direction
    - Private L2 cache is used as buffer
    - Flipped blocks are processed in parallel
    - Fast **buffer merging** as it should be done only for in-hubs
  - Processing **the sparse block in pull** direction
    - Improved locality by separating in-hubs

4- Roy, Mihailovic, Zwaenepoel, X-Stream: edge-centric graph processing using streaming partitions, SOSP’13.
Evaluation

- Comparison to state-of-the-art graph processing frameworks: GraphGrind(GGrind)$^5$, GraphIt$^6$, Galois$^7$

|            | Push |         |         | Pull |         |         |       | iHTL |
|------------|------|---------|---------|------|---------|---------|-------|------|
|            | GGrind | GraphIt | GGrind | GraphIt | Galois | iHTL |
| LvJrnl     | 91    | 770     | 54     | 106   | 37     | 28     |
| Twtr10     | 176   | 340     | 143    | 76    | 114    | 57     |
| TwtrMpi    | 895   | 1,606   | 693    | 402   | 422    | 268    |
| Frndstr    | 1,352 | 2,023   | 1,149  | 858   | 885    | 627    |
| SK         | 828   | 2,547   | 289    | 187   | 176    | 112    |
| WbCc       | 1,245 | 1,444   | 981    | 606   | 664    | 382    |
| UKDls      | 1,606 | 1,346   | 535    | 312   | 281    | 231    |
| UU         | 2,479 | 3,626   | 757    | 430   | 390    | 320    |
| UKDmn      | 2,637 | 1,827   | 806    | 439   | 407    | 348    |
| CIWb9      | 6,844 | 6,220   | 7,301  | 3,405 | 4,407  | 2,367  |
| Avg. Speedup | 4.8× | 9.5× | 2.4× | 1.7× | 1.5× | 1× |

- Comparison to state-of-the-art locality optimizing relabeling algorithms: SlashBurn(SB), GOrder(GO), Rabbit-Order(RO)

|          | Iteration Time (ms) | Preprocessing Time (s) |
|----------|----------------------|-------------------------|
|          | SB | GO | RO | iHTL | SB | GO | RO | iHTL |
| LvJrnl   | 44 | 45 | 48 | 28   | 4  | 362 | 6  | 0.9 |
| Twtr10   | 63 | 101 | 84 | 57   | 9  | 712 | 15 | 0.9 |
| TwtrMpi  | 345 | 306 | 399 | 268 | 68 | 5,697 | 66 | 4.9 |
| Frndstr  | 841 | 682 | 652 | 627 | 78 | 4,894 | 139 | 5.8 |
| SK       | 212 | 192 | 153 | 112 | 240 | 588 | 35 | 4   |
| WbCc     | 601 | 492 | 410 | 382 | 112 | 6,587 | 72 | 3.5 |
| UKDls    | 356 | 234 | 231 | 1,044 | 67 | 3.3 |
| UU       | 537 | 346 | 320 | 1,736 | 80 | 3.8 |
| UKDmn    | 492 | 399 | 348 | 1,022 | 69 | 5.5 |
| CIWb9    | 3,147 | 2,367 | 916 | 16.9 |
| Avg. Speedup | 1.5× | 1.4× | 1.3× | 1× | 200× | 2000× | 38× | 1× |

5- Sun, Vandierendonck, Nikolopoulos. GraphGrind: Addressing load imbalance of graph partitioning, ICPP’17.
6- Zhang, Yang, Baghdadi, Kamil, Shun, Amarasinghe. GraphIt: A high-performance graph dsl, OOPSLA’18.
7- Nguyen, Lenharth, Pingali, A lightweight infrastructure for graph analytics, SOSP’13.
Conclusion

• In-hub vertices connect to a large fraction of the edges, and incur a higher-than-average miss rate.

• iHTL states that
  – **Incoming edges to in-hubs** have a large number of source vertices but a very small number of destinations, so
  – **Push** is the suitable direction for this sub-graph as the small number of destinations can be accommodated in cache.

• iHTL creates
  – A number of **flipped blocks** (containing incoming edges to in-hubs) that are processed in **push** direction and
  – A **sparse block** that is processed in **pull** direction.
Thank You

Have you any question?

Further discussions on our website:
https://blogs.qub.ac.uk/GraphProcessing/Exploiting-in-Hub-Temporal-Locality-in-SpMV

Thrifty Label Propagation: Structure Aware Connected Components
IS COMING . . .
Next Month, IEEE CLUSTER’21
