Locality Analysis of Graph Reordering Algorithms

Mohsen Koohi Esfahani, Peter Kilpatrick, Hans Vandierendonck
Queen’s University Belfast

Presenter: Zhi Wei Gan
Problem

• Power-Law Distribution of Graphs
  • Leads to Random Memory Accesses
  • Time spent on Memory Accesses = Bottleneck

• Current Graph Reordering Algorithms
  • Improve locality of graph traversals by assigning new IDs to vertices in a way that vertices that are accessed together are read from main memory together
  • Hard to properly measure the performance of these reordering algorithms (excluding pure runtime)
  • Need lightweight metrics and techniques to analyze locality
Definitions

• Low-degree Vertex
  • Less than $|E|/|V|$ edges

• High-degree Vertex
  • More than $|E|/|V|$ edges

• In-hub
  • Vertices with in-degree larger than $\sqrt{V}$

• Out-hub
  • Vertices with out-degree larger than $\sqrt{V}$
Sparse Matrix-Vector (SpMV) multiplication

Algorithm 1: SpMV graph traversal

Input: $G(V, E), D^i$

Output: $D^{i+1}$

1 for $v \in V$ do
2     sum = 0;
3     for $u \in v.neighbours$ do
4         sum += $D^i[u]$;
5     end
6     $D^{i+1}[v] = sum$;
7 end

Pull-direction SpMV
Differs from Bucketing/Frontier-based as memory access pattern unpredictable, but can be used as a representative for these types
## Datasets

### TABLE I: Datasets

| Dataset | Name          | Source | |V| (M) | |E| (B) | Type  |
|---------|---------------|--------|---|------|------|------|-------|
| WebB    | WebBase-2001  | LWA    | 115| 1.0  |      | WG   |
| TwtrMpi | Twitter MPI   | NR     | 41 | 1.5  |      | SN   |
| Frndstr | Friendster    | NR     | 65 | 1.8  |      | SN   |
| SK      | SK-Domain     | LWA    | 50 | 2.0  |      | WG   |
| WbCc    | Web-CC12      | NR     | 89 | 2.0  |      | WG   |
| UKDls   | UK-Delis      | LWA    | 110| 4.0  |      | WG   |
| UU      | UK-Union      | LWA    | 133| 5.5  |      | WG   |
| UKDmn   | UK-Domain     | KN     | 105| 6.6  |      | WG   |
| CIWb9   | ClueWeb09     | NR     | 1,700| 7.9 |      | WG   |

SN = Social Network  
WG = Web Graph
Locality Types

• Type I: Spatial Reuse, proximity IDs of consecutive neighbors' results in neighbors being placed on the same cache line
• Type II: Temporal Reuse, cache reuses data of some vertex $u$ after using it to process another vertex $v$.
• Type III: Type II but to a second degree (neighbors of $u$ are also reused)
• Type IV: Reusing a cache line that was used by another thread into a shared cache (Type II but with multithreading)
• Type V: (Type III but with multithreading)
Experimental Setup

- 768 GB Main Memory
- 32KB L1 Cache
- 1MB L2 Cache
- 22MB L3 Shared Cache
Metrics to Measure Locality

• N2N AID (Spatial Locality)
• Cache Miss Rate Degree Distribution (Temporal and Spatio-Temporal Locality)
• Real Execution Performance Metrics:
  • L3 Cache Misses
  • DTLB misses
  • Idle time
  • Effective Cache Size (ECS)
Neighbor to Neighbor Average ID Distance (N2N AID)

• How RAs succeed to bring neighbors close to each other

\[ AID_v = \frac{\sum_{i=2}^{i=|N_v|} |N_{v,i} - N_{v,i-1}|}{|N_v|} \]

• Lower AID values = better spatial locality
Cache Miss Rate Degree Distribution

• They collect cache miss rates, but running it on a real machine is time consuming.
• They simulated it, but simulating cache miss rates are time consuming for large graphs.
• They optimize their simulations by doing the following:
  • Ignoring execution of non-time-consuming instructions
  • Implemented their own cache replacement policies optimized for SpMV
• Has a 15% error
Graph Reordering Algorithms

• SlashBurn
• Rabbit-Order
• GOrder
SlashBurn (SB)

- **Main idea:**
  - Finds communities of vertices by removing hubs and finding connected components
  - Assigns consecutive node IDs to hubs of the main graph

- **Locality Analysis:**
  - Improves locality types IV and V
  - SB is designed for power law graphs, but it only holds true if power-law graphs are deconstructed recursively
    - This doesn’t hold true over different iterations! Reduces locality types I and III

- **Real Execution:**
  - Destroys spatial locality.
Does SB work?

Fig. 2: [Real execution] Degree distribution of initial graph and GCC after SB iterations
Rabbit-Order

• Main idea:
  • Finds communities by using neighbors of vertices.
    • Starts at vertex with lowest degree
    • Searches for neighbor with maximum “gain” that can be reached through merging
    • Merges until there are still “gains” to be made
    • Runs a DFS on the final merged vertices to assign IDs
  • Gain function: \[ \Delta Q_{u,v} = 2\left( \frac{w_{u,v}}{2|V|} - \frac{\text{deg}_u \cdot \text{deg}_v}{(2|V|)^2} \right). \]

• Locality Analysis:
  • Reduces AID of low-degree vertices and improves spatial locality, but the DFS cannot assign consecutive IDs so AID and cache-miss rates are increased for high-degree vertices

• Real Execution:
  • Reduces L3 misses, but execution time is not better.
  • Improving locality does not translate to improved performance since RAs don’t change the locality of consecutive vertices, improving locality may increase idle time.
G-Order

• Main idea:
  • Scores between two vertices: \( S(u,v) = S_s(u,v) + S_n(u,v) \) where:
    • \( S_s \) is the sibling score (the number of common in-neighbors between \( u \) and \( v \))
    • \( S_n \) is the neighborhood score (the number of edges \( u \) and \( v \))
  • Concentrates on temporal reuse instead of identifying communities

• Locality Analysis:
  • Reduces the cache miss rate on high-degree vertices but doesn’t perform well for low-degree vertices
  • Increases the number of reloads of high-degree vertices to provide space in cache for low-degree vertices

• Real execution:
  • Reduces L3 misses
## Results

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Time (ms)</th>
<th>Idle (%)</th>
<th>L3 Misses (M)</th>
<th>DTLB Misses (K)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BI</td>
<td>SB</td>
<td>GO</td>
<td>RO</td>
</tr>
<tr>
<td>WebB</td>
<td>90</td>
<td>145</td>
<td>89</td>
<td>79</td>
</tr>
<tr>
<td>TwtrMpi</td>
<td>354</td>
<td>339</td>
<td>299</td>
<td>366</td>
</tr>
<tr>
<td>Frndstr</td>
<td>771</td>
<td>761</td>
<td>578</td>
<td>667</td>
</tr>
<tr>
<td>SK</td>
<td>117</td>
<td>168</td>
<td>109</td>
<td>109</td>
</tr>
<tr>
<td>WbCc</td>
<td>438</td>
<td>414</td>
<td>311</td>
<td>297</td>
</tr>
<tr>
<td>UKDls</td>
<td>194</td>
<td>317</td>
<td>180</td>
<td>180</td>
</tr>
<tr>
<td>UU</td>
<td>282</td>
<td>486</td>
<td>285</td>
<td>180</td>
</tr>
<tr>
<td>UKDmn</td>
<td>297</td>
<td>459</td>
<td>281</td>
<td>281</td>
</tr>
<tr>
<td>ClWb9</td>
<td>2,221</td>
<td>2,811</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Locality Analysis of Datasets

• High-degree vertices have close connection to each other in social networks

• Low-degree vertices constitute most web graphs

• Asymmetry: the fraction of in-neighbors that are not out-neighbors for each vertex

Fig. 4: [Calculation] Asymmetricity degree distribution
CSC vs CSR

CSR

Index Pointers
0 2 3 3 6 6 7

Indices
0 2 2 2 3 4 3

Data
8 2 5 7 1 2 9

CSC

Index Pointers
0 1 1 4 6 7

Indices
0 0 1 4 4 6 4

Data
8 2 5 7 1 9 2
Pull vs. Push Traversal for SpMV

• We use CSR for push
• CSC for pull

TABLE VI: [Real execution] CSC vs. CSR read traversals

<table>
<thead>
<tr>
<th>Dataset</th>
<th>L3 Misses (M)</th>
<th>Traversal Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CSC</td>
<td>CSR</td>
</tr>
<tr>
<td>WebB</td>
<td>4.3</td>
<td>3.8</td>
</tr>
<tr>
<td>TwtrMpi</td>
<td>15.7</td>
<td>21.7</td>
</tr>
<tr>
<td>SK</td>
<td>5.7</td>
<td>4.6</td>
</tr>
<tr>
<td>UKDs</td>
<td>10.1</td>
<td>9.3</td>
</tr>
<tr>
<td>CIWb9</td>
<td>100.9</td>
<td>96.5</td>
</tr>
</tbody>
</table>

Why? For pull-traversal, out-hubs have a constructive effect on locality since data is constantly accessed and reused, but for push traversal in-hubs improve locality.

Web graphs are better with CSR traversal
Social networks are better with CSC traversal

Fig. 6: [Calculation] Comparison of percentage of edges covered by in-hubs in CSR traversal vs. out-hubs in CSC traversal
Optimizing RAs

• SB: continue iterating while GCC-max-degree >= sqrt(V)

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Preprocessing (s)</th>
<th>Traversal (ms)</th>
<th>L3 Misses (M)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SB</td>
<td>SB++</td>
<td>SB</td>
</tr>
<tr>
<td>TwtrMpi</td>
<td>46</td>
<td>21</td>
<td>339</td>
</tr>
<tr>
<td>Frndstr</td>
<td>75</td>
<td>43</td>
<td>761</td>
</tr>
<tr>
<td>WbCc</td>
<td>81</td>
<td>39</td>
<td>414</td>
</tr>
</tbody>
</table>

• RO: skip relabeling vertices that are not in an efficacy degree range to reduce preprocessing time and memory