HiPa: Hierarchical Partitioning for Fast PageRank on NUMA Multicore Systems

YuAng Chen, Yeh-ching Chung
The Chinese University of Hong Kong, Shenzhen
Content

1. Introduction
2. Design of HiPa
3. Evaluation
4. Conclusion
1 Introduction

PageRank

Issues
PageRank: Larry Page - or webpage
Introduction

(a) PageRank

\[ PR_{new}(v) = 1 - d |V| + d \times \sum_{u \in \text{in}(v)} PR_{old}(u) \times |E_{out}(u)| \]

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(d)</td>
<td>damping factor</td>
</tr>
<tr>
<td>(</td>
<td>V</td>
</tr>
<tr>
<td>(u)</td>
<td>in-neighbors of (v)</td>
</tr>
<tr>
<td>(</td>
<td>E_{out}(u)</td>
</tr>
</tbody>
</table>

the workload of PageRank mainly depends on the edges of the graph
Introduction

(b) Issue of graph processing on multicore systems

Pointer-based Data Structure

Random graph

```
0 1 2 3 4 5
0 1 1 1 1
1 1 1 1 1
2 1 1 1 1
3 1 1 1 1
4 1 1 1
5 1 1
```
Introduction

(b) Issue of graph processing on multicore systems

*Skewed* power-law degree distribution

![Graph showing skewed power-law distribution](image)
Hierarchical Partitioning

- Memory
- Cache
- Thread
Design of HiPa

(x) Abstraction
Goal: **co-locate** the computation and data within the same NUMA node.

Step 1 - Intuition:
Each NUMA node $i$ is allocated with the same number of edges $|E|/N$.

\[
|E_i| = \frac{|E|}{N}
\]

\[
V_i = \{v \in V | \sum_{v \in V} D(v) = \frac{|E|}{N} \}
\]

Step 2 - Roundup:
- The number of vertices allocated to a NUMA node must be a multiple of L2-partitions;
- The size of a L2-partition $P$ is fixed to $|P| = \{\text{L2 cache size}\} / \{\text{single vertex size}\}$.

\[
|\hat{V}_i| = \text{ceil}\left(\frac{|V_i|}{|P|}\right) \cdot |P| = \left( \frac{|V_i| - 1}{|P|} + 1 \right) \cdot |P| = n_i \cdot |P|
\]

\[
|\hat{E}_i| = \sum_{v \in \hat{V}_i} D(v)
\]
Goal:
promote high cache **locality**

Step 3 – Distribution of partitions
- These L-2 partitions are organized in groups $G$ and then distributed to cores $C$.
- Each group $G$ of Core $j$ ($1 < j < C$) contains (roughly) the same no. of edges

\[
\begin{align*}
n_i &= \sum_{j=1}^{C} m_j \\
|G_j| &= m_j \cdot |P| \\
\frac{|\tilde{E}_i|}{C} &= \sum_{v \in G_j} D(v)
\end{align*}
\]

Why partitioning by L2 cache?
Cache-able disjoint partitions of a graph, limits the vertex access within
L2 cache for high cache locality
The optimal partition size is to be discussed later.
1. The boxes represent cache-able partitions of the graph data.
2. P0-2 hold 10 edges, P3-4 hold 15 edges, and P5-6 hold 30 edges.
3. The processor cores are allocated with *unequal* numbers of partitions but *equal* number of edges.
Algorithm 2: Numa-aware scatter-gather model

**Input:** numa_Partitions → numa-ly allocated partitions
**Input:** numa_Threads → numa-ly bound threads

```
Function Th_Func(numa_part)
  for i ← 0 to iter do
    scatter(numa_part);
    synchronize with other threads;
    gather(numa_part);
  for th ∈ numa_Threads do in parallel  \(\triangleright\) parallel region
    match th with \(p ∈\) numa_Partitions;
    th calls Th_Func(p);
  Graph ← concatenate (numa_Partitions)
```

YuAng Chen
3 Evaluation

- Execution Time
- Memory Access
- Sensitivity
## Evaluation

### Experiment Setup

<table>
<thead>
<tr>
<th>Machine</th>
<th>Configurations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Xeon Silver 4210 processors</td>
<td>2</td>
</tr>
<tr>
<td>Physical, Virtual Cores</td>
<td>20, 40</td>
</tr>
<tr>
<td>L1, L2, LLC Caches</td>
<td>64KB, <strong>1MB</strong>, 13.75MB.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Contemporary works</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>P-PR</td>
<td>Hand-optimized code, partition-centric</td>
</tr>
<tr>
<td>V-PR</td>
<td>Hand-optimized code, vertex-centric</td>
</tr>
<tr>
<td>GPOP</td>
<td>Framework, partition-centric</td>
</tr>
<tr>
<td>Polymer</td>
<td>Framework, vertex-centric, NUMA-aware</td>
</tr>
</tbody>
</table>
3 Evaluation

(a) Execution Time

Execution time (in seconds) of 20-iteration PageRank with various implementations.

<table>
<thead>
<tr>
<th></th>
<th>HiPa</th>
<th>p-PR</th>
<th>v-PR</th>
<th>GPOP</th>
<th>Polymer</th>
</tr>
</thead>
<tbody>
<tr>
<td>journal</td>
<td>0.31</td>
<td>0.41</td>
<td>0.54</td>
<td>1.14</td>
<td>1.72</td>
</tr>
<tr>
<td>pld</td>
<td>2.43</td>
<td>3.37</td>
<td>8.44</td>
<td>4.18</td>
<td>22.27</td>
</tr>
<tr>
<td>wiki</td>
<td>1.74</td>
<td>1.80</td>
<td>1.96</td>
<td>3.90</td>
<td>4.63</td>
</tr>
<tr>
<td>kron</td>
<td>7.20</td>
<td>10.06</td>
<td>32.82</td>
<td>11.29</td>
<td>76.62</td>
</tr>
<tr>
<td>twitter</td>
<td>8.43</td>
<td>9.83</td>
<td>12.09</td>
<td>14.91</td>
<td>41.06</td>
</tr>
<tr>
<td>mpi</td>
<td>13.93</td>
<td>17.54</td>
<td>24.41</td>
<td>33.90</td>
<td>64.00</td>
</tr>
</tbody>
</table>

HiPa > others
Hand-coded (HiPa, p-PR, v-PR) > framework-based (GPOP, Polymer)
Partition-centric (p-PR, GPOP) > vertex-centric (v-PR, Polymer)
The total bar is the total memory accesses: remote + local memory accesses.

The lower, shadowed bar segment: the remote memory accesses.

HiPa achieves the least remote memory access, which is the key reason for the performance gain.
Evaluation

(c) Scalability

The lowest point means the best performance
• p-PR and GPOP @ 20 threads, and then decay as the #thread grows
• HiPa, v-PR and Polymer @ 40 threads exhibits higher scalability
  • Thread-data pinning of HiPa: thread contention ↓, scalability ↑
The optimal partition size = \( \frac{1}{4} \times \text{L2 cache size on Skylake} = 256\text{KB} \)

= \( \frac{1}{2} \times \text{L2 cache size on Haswell} = 128\text{KB} \)
4 Conclusion

Key Features
4 Conclusion

(a) Key Features
4 Conclusion

(c) Main Achievement

- Execution Speedup
- Reduced remote memory access
- High Scalability
- Performance Gain
Thank you