EXPLOITING LOCALITY IN GRAPH ANALYTICS THROUGH HARDWARE ACCELERATED TRAVERSAL SCHEDULING

Anurag Mukkara, Nathan Beckmann, Maleen Abeydeera, Xiaosong Ma, Daniel Sanchez

MICRO 2018
The locality problem of graph processing

- Irregular structure of graphs causes seemingly random memory references
- On-chip caches are too small to fit most real-world graphs
- Software frameworks improve locality through offline preprocessing
- Preprocessing is expensive and often impractical

1 PageRank iteration on UK web graph
Improving locality in an online fashion

- Traversal schedule decides order in which graph edges are processed
- Many real-world graphs have strong community structure
- Traversals that follow community structure have good locality
- Performing this in software without preprocessing is not practical due to scheduling overheads
Contributions

- **BDFS**: Bounded Depth-First Scheduling
  - Performs a series of bounded depth-first explorations
  - Improves locality for graphs with good community structure

- **HATS**: Hardware Accelerated Traversal Scheduling
  - A simple unit specialized for traversal scheduling
  - Cheap and implementable in reconfigurable logic
Agenda

- Background
- BDFS
- HATS
- Evaluation
Graph data structures

Graph representation

Compressed Sparse Row (CSR) Format

Offset array

Neighbor array

Algorithm-specific

Vertex data
Vertex-ordered (VO) schedule follows layout order

- Simplifies scheduling and parallelism
- Poor locality for vertex data accesses

![Diagram](image)

**PageRank on UK web graph**

- Full Spatial Locality
- Low Spatial Locality
- No Temporal Locality
- Low Temporal Locality

<table>
<thead>
<tr>
<th>Time</th>
<th>Load/Store</th>
<th>Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

**Graph Metrics**

- Neighbors
- Offsets
- Vertex Data

**Main Memory Accesses**

- VO
  - 0.0
  - 0.2
  - 0.4
  - 0.6
  - 0.8
  - 1.0
Agenda

- Background
- BDFS
- HATS
- Evaluation
BDFS: Bounded Depth-First Scheduling

- Vertex data accesses have high potential temporal locality
- Following community structure helps harness this locality
- BDFS performs a series of bounded depth-first explorations
- Traversal starts at vertex with id 0
- Processes all edges of first community before moving to second
- Divide-and-conquer nature of BDFS
  - Small depth bounds capture most locality
  - Good locality at all cache levels
BDFS reduces total main memory accesses

- High Spatial Locality
- Low Temporal Locality
- Lower Spatial Locality
- No Temporal Locality

PageRank on UK web graph

- High Spatial Locality
- No Temporal Locality
- Low Spatial Locality
- Low Temporal Locality
- Low Spatial Locality
- High Temporal Locality
BDFS in software does not improve performance

- Scheduling overheads negate the benefits of better locality
- Higher instruction count
- Limited ILP and MLP
  - Interleaved execution of traversal scheduling and edge processing
  - Unpredictable data-dependent branches
Agenda

- Background
- BDFS
- HATS
- Evaluation
HATS: Hardware Accelerated Traversal Scheduling

- Decouples traversal scheduling from edge processing logic
- Small hardware unit near each core to perform traversal scheduling
- General-purpose core runs algorithm-specific edge processing logic
- HATS is decoupled from the core and runs ahead of it
HATS operation and design

L2

HATS

L1

Core

Prefetches

HATS

accs.

Config.

FIFO Edge Buffer

Core

accs.

Scan

Fetch Offsets

Fetch Neighbors

Prefetch

VO-HATS

Scan

Fetch Offsets

Fetch Neighbors

Prefetch

BDFS-HATS

Stack

Exploration FSM
HATS costs

- Adds only one new instruction
  - Fetches edge from FIFO buffer to core registers

- Very cheap and energy-efficient over a general-purpose core
  - RTL synthesis with a 65nm process and 1GHz target frequency

<table>
<thead>
<tr>
<th></th>
<th>ASIC</th>
<th>FPGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area</td>
<td>0.4% of core</td>
<td>0.2% of core</td>
</tr>
<tr>
<td>TDP</td>
<td>0.2% of core</td>
<td>3200 LUTs</td>
</tr>
</tbody>
</table>
HATS benefits

- Reduces work for general-purpose core for VO
- Enables sophisticated scheduling like BDFS
- Performs accurate indirect prefetching of vertex data
- Accelerates a wide range of algorithms
- Requires changes to graph framework only, not algorithm code
Agenda

- Background
- BDFS
- HATS
- Evaluation
Evaluation methodology

- Event-driven simulation using zsim
- 16-core processor
  - Haswell-like OOO cores
  - 32 MB L3 cache
  - 4 memory controllers

- IMP [Yu, MICRO’15]
  - Indirect Memory Prefetcher
  - Configured with graph data structure information for accurate prefetching

- 5 applications from Ligra framework
  - PageRank (PR)
  - PageRank Delta (PRD)
  - Connected Components (CC)
  - Radii Estimation (RE)
  - Maximal Independent Set (MIS)

- 5 large real-world graph inputs
  - Millions of vertices
  - Billions of edges
HATS improves performance significantly

- IMP improves performance by hiding latency
- VO-HATS outperforms IMP by offloading traversal scheduling from general-purpose core
- BDFS-HATS gives further gains by reducing memory accesses
HATS reduces both on-chip and off-chip energy

- IMP reduces static energy due to faster execution time
- VO-HATS reduces core energy due to lower instruction count
- BDFS-HATS reduces memory energy due to better locality

On-chip (Core + Cache)  Off-chip (Memory)
HATS on an on-chip reconfigurable fabric
- Parallelism enhancements to maintain throughput at slower clock cycle

Sensitivity to on-chip location of HATS (L1, L2, LLC)

Adaptive-HATS
- Avoids performance loss for graphs with no community structure

HATS versus other locality optimizations
Graph processing is bottlenecked by main memory accesses.

BDFS exploits community structure to improve cache locality.

HATS accelerates traversal scheduling to make BDFS practical.

Thanks For Your Attention!
Questions Are Welcome!