High-Performance Graph Processing: Locality, Vectorization and Reduced Precision

Hans Vandierendonck
Queen’s University Belfast
1 December 2020
GRAPH ALGORITHMS

Iteratively calculate a property of each vertex in a graph
E.g.: PageRank values, label of connected components, embeddings in graph convolutional neural networks, etc.

Vertex model

Driven by frontier: set of active vertices
Many updates in parallel, with conflicts
Programming Interface

Ligra [Shun PPoPP’13]
Assume graph G=(V,E)

**EdgeMap** applies an operation F to each edge \((u,v)\) \(\in E\) where \(u \in U\) and \(C(v) = true\). It returns a frontier that contains all \(v\) where any call to \(F(u,v)\) returned true

**VertexMap** applies an operation F to each vertex \(v\) \(\in U\) and returns a frontier that contains \(v\) iff \(v \in U\) and \(F(v) = true\)

In both cases, F may have side effects, e.g., updating properties for the vertices

\[
\begin{align*}
\text{size}(U : \text{frontier}) & : N \\
\text{returns} & : |U| \\
\text{EdgeMap}(G : \text{graph}, \ U : \text{frontier}, \ F : (\text{vertex} \times \text{vertex}) \rightarrow \text{bool}, \ C : \text{vertex} \rightarrow \text{bool}) & : \text{frontier} \\
\text{VertexMap}(U : \text{frontier}, \ F : \text{vertex} \rightarrow \text{bool}) & : \text{frontier}
\end{align*}
\]
EXAMPLE: CONNECTED COMPONENTS

Graph G = ...;
VID label[G.getNumVertices()];
G.vertexmap( [&](VID v) { label[v] = v; } );
Frontier F = Frontier::all_true();
while( ! F.empty() ) {
    Frontier newF = G.edgemap( F,
        [&](VID src, VID dst) { return label[dst] min= label[src]; } );
    F = newF;
}

Question: How to implement vertexmap and edgemap efficiently?
Part I: Non-Uniform Memory Architectures
GOAL

How to map graph analytics over immutable graphs onto a NUMA architecture while minimising execution time?

Remote access has higher latency, lower bandwidth than local access
Stores are more affected than loads
EDGEMAP, VERTEXMAP AND NUMA-AWARENESS

**Goal**: map code and data to NUMA nodes

One type of arrays
- Properties (per vertex)

Two types of loops
- Loops over edges
- Loops over vertices

Two types of iteration
- Sparse frontier
- Dense frontier
RECAP: DATA RACES

A pair of load and store instructions, at least one of which is a store, that access the same memory location

In a concurrent program with data races, the outcome of the program may differ depending on the relative execution speed of threads

Typical solutions:
• mutual exclusion
• atomic memory operations
• owner-computes


**OWNER-COMPUTES**

Decomposition based on partitioning input/output data is referred to as the owner computes rule. Each partition performs all the computations involving data that it owns:

- **Input data decomposition**: A task performs all the computations that can be done using these input data.
- **Output data decomposition**: A task computes all the results in the partition assigned to it.

---

**Input partitioning**
Red and blue processors

**Output partitioning**

NUMA-AWARE LAYOUT FOR EDGEMAP

Goal
Determine cuts of \{ code, data \} such that performance is maximised

How?
Partition graph such that each partition (NUMA node) has an equal:

1. \#edges, \#cuts [PowerGraph OSDI’12]
   ... breaks locality
2. \#sources [X-stream SOSP’13]
   ... remote updates
3. \#edges [Polymer PPoPP ‘15]
4. \((\alpha \#destinations + \#edges)\) [Gemini OSDI’16]
NUMA-AWARE LAYOUT FOR EDGEMAP

It depends! [GraphGrind ICS’17]

“Vertex-oriented” algorithms
- Best performance with equal #destinations
- Frontier density mostly below 50%
- BFS, Betweenness Centrality, Bellman-Ford

“Edge-oriented” algorithms
- Best performance with equal #edges
- Frontier density mostly close to 100%
- PageRank, SpMV, Belief Prop., PageRankDelta
NUMA-AWARE LAYOUT FOR VERTEXMAP

“Vertex-oriented” algorithms
• Trivial

“Edge-oriented” algorithms
• Need to choose between balancing compute and minimising traffic across NUMA nodes
• Better to balance compute and incur additional inter-node traffic [GraphGrind ICS’17]
• Consequently, data is partitioned the same way as compute in edgemap, but differently in vertexmap
NUMA-AWARENESS CHOICES

- Baseline is CPU balance and memory imbalance
  - Implies remote accesses during vertex map

- CPU imbalance
  - No remote accesses during vertex map

- Memory balance
  - No remote accesses during vertex map
  - Many remote accesses during edge map

4-socket 2.6GHz Intel Xeon E7-4860 v2, 48 threads, 256 GiB

Performance improvement

- Baseline
- CPU imbalance
- Memory balance

J. Sun, H. Vandierendonck and D. S. Nikolopoulos, ”GraphGrind: addressing load imbalance of graph partitioning”, ICS’17
CAN WE MEET BOTH REQUIREMENTS?

Have our cake and eat it too!
LOAD BALANCE

Execution time/partition highly dependent on the degree of vertices

Reorder vertices
- in order of decreasing in-degree
- using list scheduling

VEBO: Vertex and Edge Balanced Partitioning

Revisiting edge balance:
Two partitions with 3 edges
Which partition is processed faster?
VEBO BENEFITS

All metrics per thousand instructions
Partitions are processed faster as a side-effect of reordering
Remote cache misses are traded for local misses

PageRank
PERFORMANCE

Comparing Ligra, Polymer (NUMA-aware), and 3 versions of GraphGrind

Twitter graph

4-socket 2.6GHz Intel Xeon E7-4860 v2, 48 threads, 256 GiB

Similar results hold for other graphs

VEBO relabels vertex IDs to achieve load balance