GraphIt - A High-Performance Graph DSL

Yunming Zhang, Sherry Yang, Riyadh Baghdadi, Shoaib Kamil, Julian Shun, Saman Amarasinghe

sherryy@google.com

Sept 17, 2018

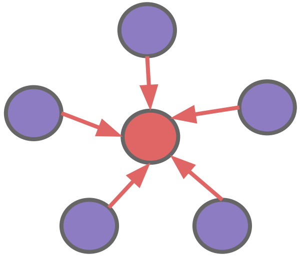
Motivation

1. Graph algorithms exhibit different performance characteristics.

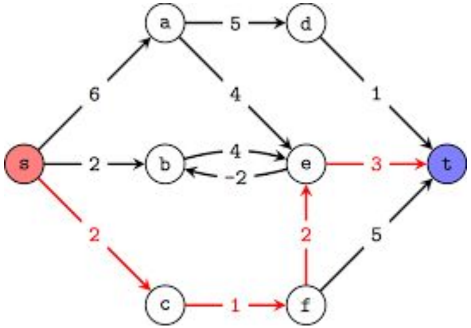
Google

PageRank

$$PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)}$$

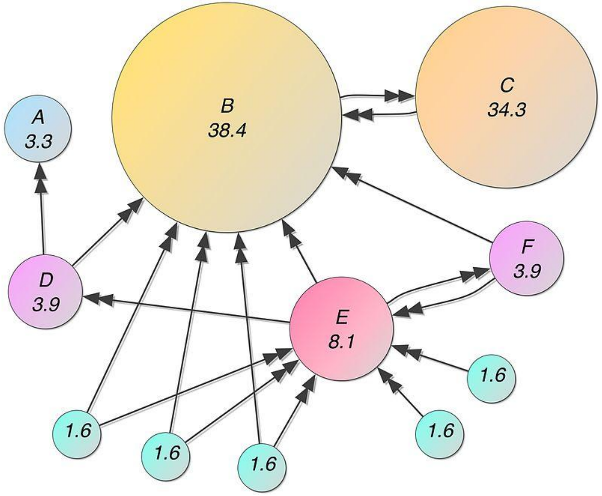


Single-Source Shortest Path



Motivation

2. Diverse graph structures.



Power-law distribution:
web graphs
social networks

Regular:
road graphs

Motivation

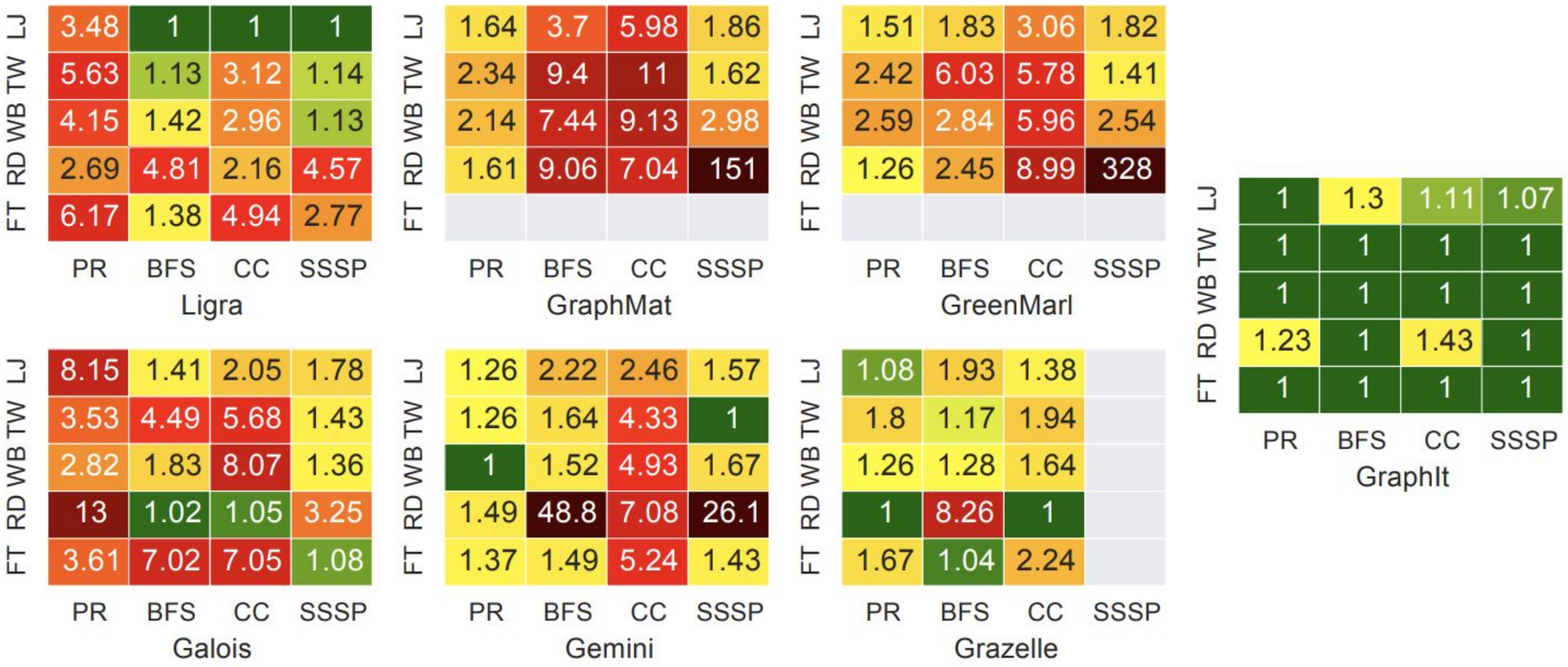
3. Different hardware platforms.

Motivation

No graph processing framework or library can take into account all graphs, algorithms, and hardware configurations

GraphIt

- First graph compiler to separate algorithms from scheduling
- Consistently achieves high-performance



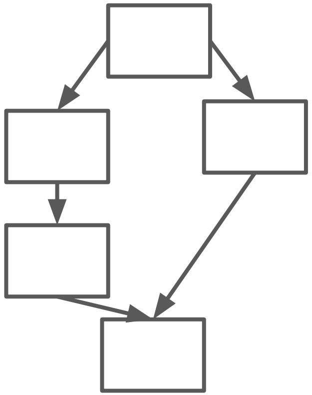
Highlights

- Locality, work-efficiency, and parallelism trade-off analysis
- Separation of graph algorithms and performance scheduling
- Graph iteration space model to encode optimizations

1. Trade-off Analysis

1. Trade-off Analysis

- **Locality**: spatial and temporal reuse
- **Work-efficiency**: the inverse of the total number of instructions
- **Parallelism**: relative amount of work that can be executed independently by different processing units

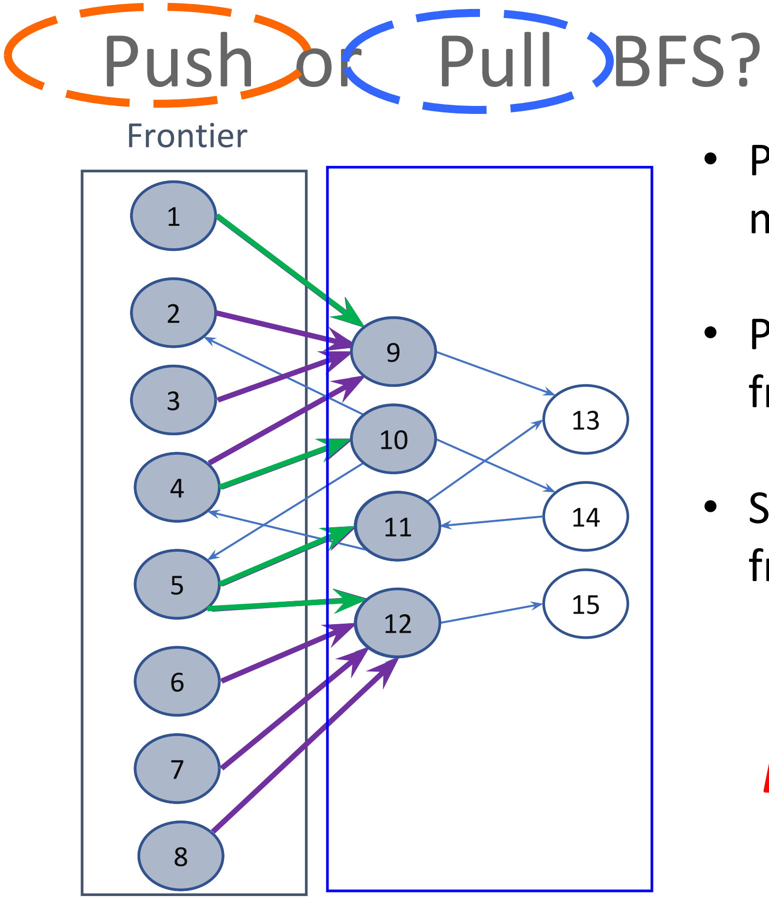


Work = 5

Span = 4

Parallelism = 5/4

1. Trade-off Analysis

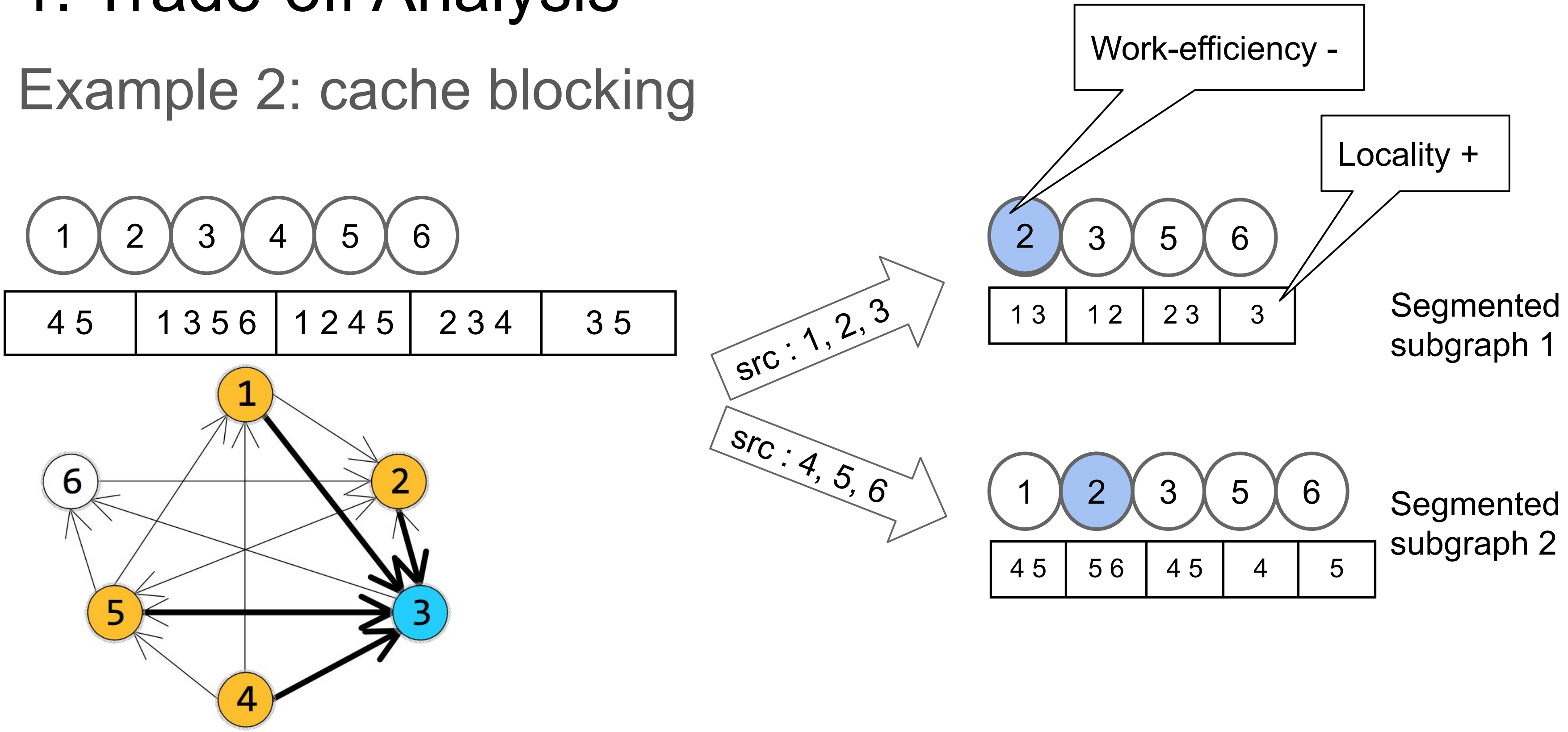


- Pull method better when frontier is large and many vertices have been visited
- Push (traditional) method better for small frontiers
- Switch between the two methods based on frontier size [Beamer et al. SC '12]

Limited to BFS?

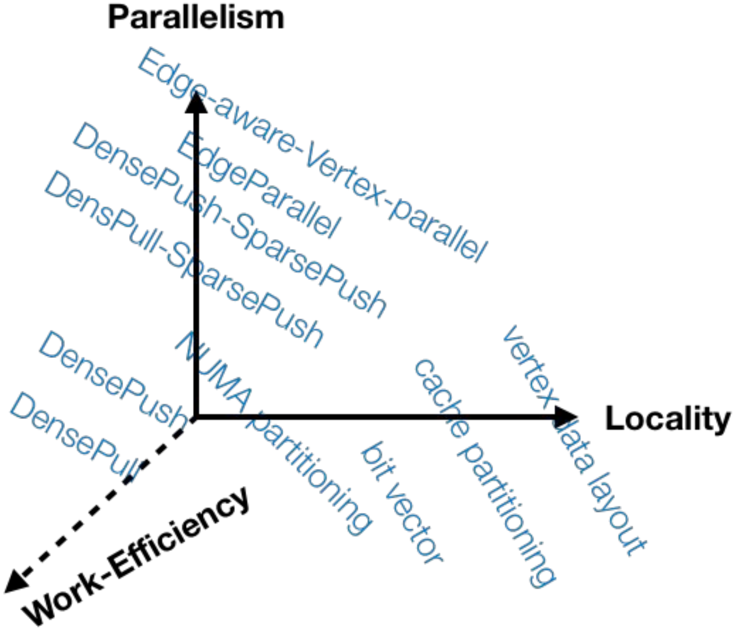
1. Trade-off Analysis

Example 2: cache blocking



1. Trade-off Analysis

Frameworks	Traversal Directions	Dense Frontier Data Layout	Parallelization	Vertex Data Layout	Cache Opt.	NUMA Opt.	Optimization Combinations Count
GraphIt	SPS, DPS, SP, DP, SPS-DP, DPS-SPS	BA, BV	WSVP, WSEVP, SPVP	AoS, SoA	Partitioned, No Partition	Partitioned, Interleaved	**100+**



- The need for a scheduling language
- The need for auto-tuning

2. The Algorithm and Scheduling Languages

2. The Algorithm and Scheduling Languages

Algorithm language

```
func updateEdge(src, dst)
    parent[dst] = src;

func BFS()
    while (frontier)
        #s1# frontier = edges.apply(updateEdge)
```

Scheduling language

```
program
->configApplyNUMA("s1", "static-parallel");
->configApplyDirection("s1", "DensePull")
->configApplyParallelization("s1",
                             "dynamic-vertex-parallel")
```

Generated C++

```
for (int segmentId = 0; segmentId < g.getNumSegments("s1"); segmentId++) {
  auto sg = g.getSegmentedGraph(std::string("s1"), segmentId);
  parallel_for (NodeID localId=0; localId < sg->numVertices; localId++) {
    NodeID dst = sg->graphId[localId];
    if (to_func(dst)){
      for (int64_t ngh = sg->vertexArray[localId]; ngh < sg->vertexArray[localId+1]; ngh++) {
        NodeID src = sg->edgeArray[ngh];
        if (frontier.get_bit(src)) {
          if (apply_func(src, dst)) {
            next[dst] = 1;
            if (!to_func(dst)) break;
          }
}}}}}
```

3. The Graph Iteration Space

3. The Graph Iteration Space

1) Enables the compiler to easily compose optimizations

2) Easy to reason about validity through dependence analysis

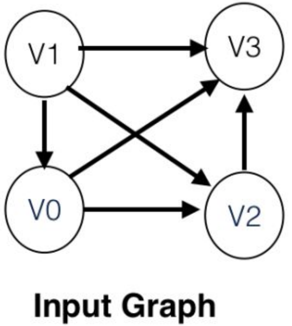
3) Guides the generate nested loop traversal code

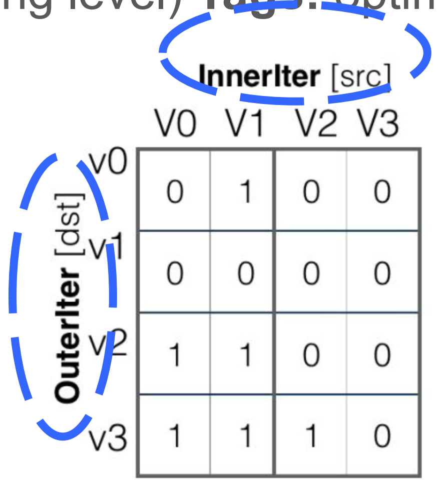
4) Enables auto-tuning

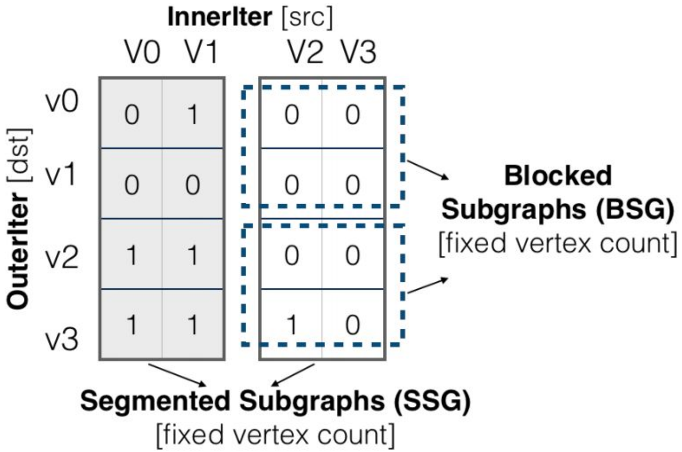
3. The Graph Iteration Space

represents combinations of optimizations as 4-D vectors

Index: dimension (nesting level) **Tags:** optimizations







<SSG, BSG, OuterIter, InnerIter>

3. The Graph Iteration Space

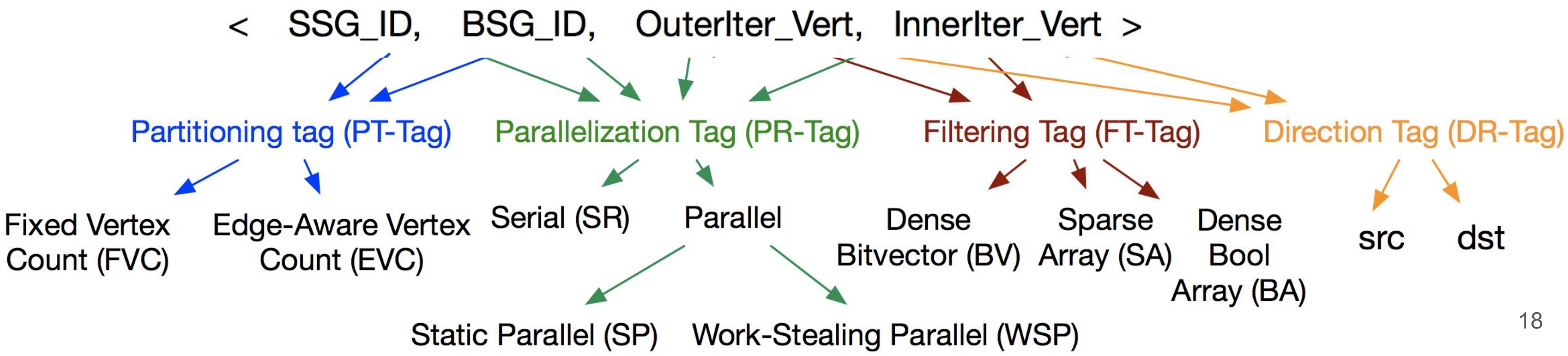
Tags: optimizations

Partitioning tag: vertex count, edge count

Parallelization tag: serial, parallel, parallel with work-stealing

Direction tag: src, dst

Filtering tag: bitvector, sparse array, dense array



Performance Summary

- Up to 4.8X faster
- Never more than 43% slower

Ligra

	PR	BFS	CC	SSSP
LJ	3.48	1	1	1
TW	5.63	1.13	3.12	1.14
WB	4.15	1.42	2.96	1.13
RD	2.69	4.81	2.16	4.57
FT	6.17	1.38	4.94	2.77

GraphMat

	PR	BFS	CC	SSSP
LJ	1.64	3.7	5.98	1.86
TW	2.34	9.4	11	1.62
WB	2.14	7.44	9.13	2.98
RD	1.61	9.06	7.04	151
FT				

GreenMarl

	PR	BFS	CC	SSSP
LJ	1.51	1.83	3.06	1.82
TW	2.42	6.03	5.78	1.41
WB	2.59	2.84	5.96	2.54
RD	1.26	2.45	8.99	328
FT				

GraphIt

	PR	BFS	CC	SSSP
LJ	1	1.3	1.11	1.07
TW	1	1	1	1
WB	1	1	1	1
RD	1.23	1	1.43	1
FT	1	1	1	1

Galois

	PR	BFS	CC	SSSP
LJ	8.15	1.41	2.05	1.78
TW	3.53	4.49	5.68	1.43
WB	2.82	1.83	8.07	1.36
RD	13	1.02	1.05	3.25
FT	3.61	7.02	7.05	1.08

Gemini

	PR	BFS	CC	SSSP
LJ	1.26	2.22	2.46	1.57
TW	1.26	1.64	4.33	1
WB	1	1.52	4.93	1.67
RD	1.49	48.8	7.08	26.1
FT	1.37	1.49	5.24	1.43

Grazelle

	PR	BFS	CC	SSSP
LJ	1.08	1.93	1.38	
TW	1.8	1.17	1.94	
WB	1.26	1.28	1.64	
RD	1	8.26	1	
FT	1.67	1.04	2.24	

Summary

- Identifies locality, work-efficiency, and parallelism trade-offs
- Provides an algorithm language and a scheduling language
- Graph iteration space model to encode optimizations
- Supports for auto-tuning

Open source: http://graphit-lang.org/