SFU

DZiG: Sparsity-Aware Incremental Processing of Streaming Graphs

Mugilan Mariappan

Joanna Che

Keval Vora

Simon Fraser University

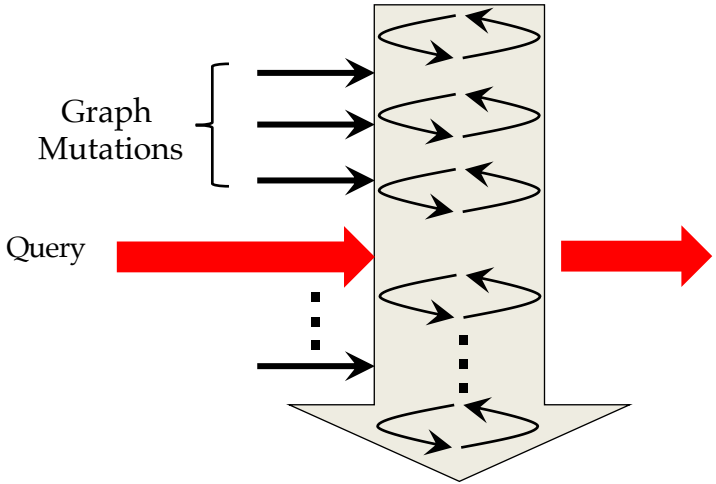
EuroSys'21

April 28, 2021

Streaming Graph Processing

- Graph structure keeps on changing via a stream of edge/vertex updates

- **Incremental Processing**
  - Adjust results incrementally
- Reuse work that has already been done



graphbolt

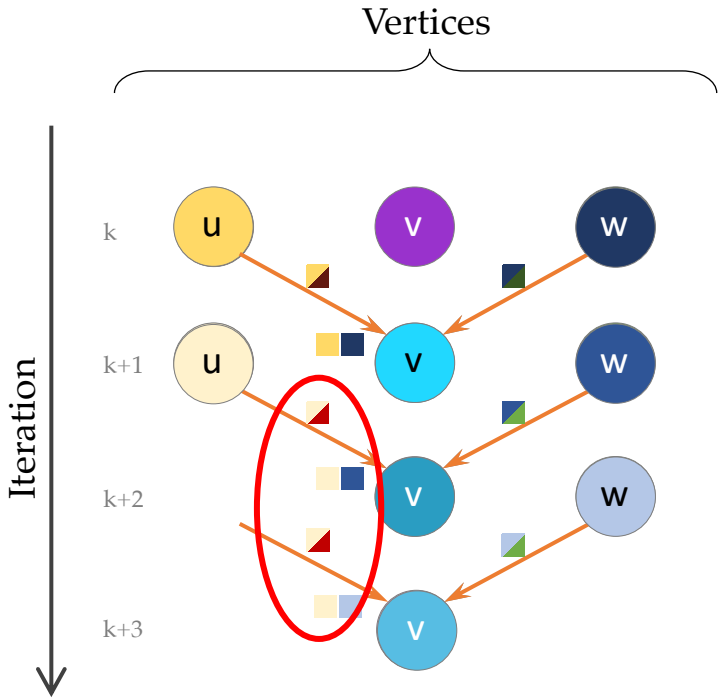
- **Dependency-Driven Incremental Refinement**
- Generates same results at every iteration as a BSP execution from scratch

GraphBolt: Dependency-Driven Synchronous Processing of Streaming Graphs. Mugilan Mariappan and Keval Vora, EuroSys 2019.

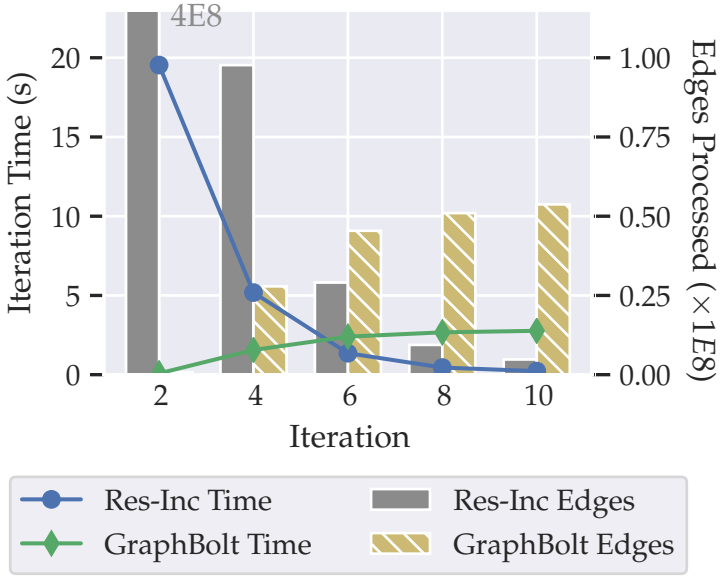
Dependency-Driven Incremental Refinement

- Upon graph mutation:
  - Identify changes in values
  - Propagate changes step-by-step

- Changes not retained across iterations
- Not friendly to **computation sparsity**



Computation Sparsity



- Hybrid execution
- **Turn off dependency-driven incremental refinement** during sparse iterations

How to retain high performance with dependency-driven incremental processing during sparse computations?

DZiG: Sparsity-Aware Incremental Refinement

- Dependency-driven incremental processing
- Delivers **high performance in presence of sparse computations**
- Generates same results at every iteration as a BSP execution from scratch

How to **identify sparse computations** during incremental refinement?

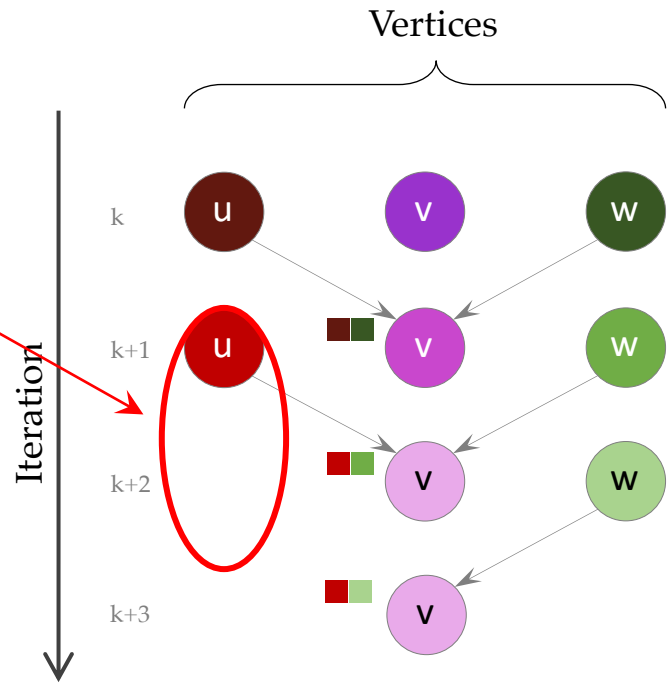
How to **retain computation sparsity** during incremental refinement?

DelZero: Sparse Updates

$$\delta_i(v) = \varnothing \quad \text{iff} \quad |v_i - v_{i-1}| \leq \epsilon$$

```
while(curr not empty) {
  parallel_for u in curr {
    float change = (pr[u] - prev_pr[u]) / numOutNbrs(u);
    parallel_for v in outNeighbors(u) {
      atomicAdd(&sum[v], change);
      next.add(v);
    }
  }
  swap(curr, next);
  next.clear();
  parallel_for v in curr {
    float rank = 0.15 + 0.85 * sum[v];
    if(fabs(rank - pr[v]) > ϵ) {
      prev_pr[v] = pr[v];
      pr[v] = rank;
      next.add(v);
    }
  }
```

PageRank



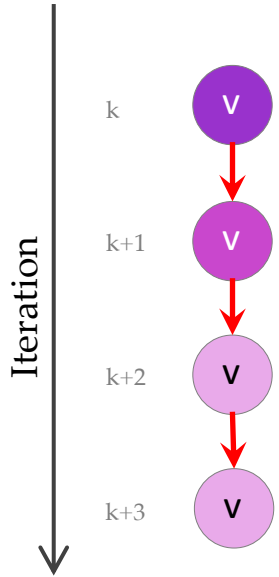
DelZero-Aware Incremental Refinement

- Express **incremental refinement recursively**

$$g_i^T(v) = g_i(v) \oplus \Delta_i^T(v)$$

$$\Delta_i^T(v) = \Delta_{i-1}^T(v) \biguplus_{\forall e=(u,v)\in E_a \; s.t.\; \delta_{i-1}(u)\neq\varnothing} \delta_{i-1}(u) \biguplus_{\forall e=(u,v)\in E_d \; s.t.\; \delta_{i-1}(u)\neq\varnothing} \delta_{i-1}(u)$$

$$\biguplus_{\forall e=(u,v)\in E^c \; s.t.\; \delta_{i-1}(u)\neq\varnothing} \delta_{i-1}(u) \biguplus_{\forall e=(u,v)\in E^c \; s.t.\; \delta_{i-1}^T(u)\neq\varnothing} \delta_{i-1}^T(u)$$



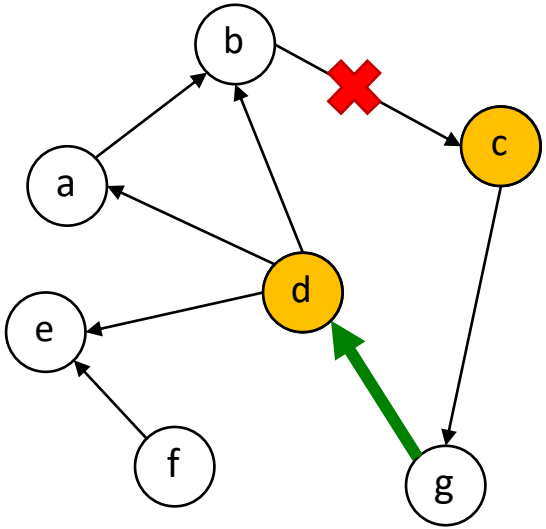
DelZero-Aware Incremental Refinement

- Express **incremental refinement recursively**

$$g_i^T(v) = g_i(v) \oplus \Delta_i^T(v)$$

$$\Delta_i^T(v) = \Delta_{i-1}^T(v) \biguplus_{\forall e=(u,v)\in E_a \; s.t.\; \delta_{i-1}(u)\neq\varnothing} \delta_{i-1}(u) \biguplus_{\forall e=(u,v)\in E_d \; s.t.\; \delta_{i-1}(u)\neq\varnothing} \delta_{i-1}(u)$$

$$\biguplus_{\forall e=(u,v)\in E^c \; s.t.\; \delta_{i-1}(u)\neq\varnothing} \delta_{i-1}(u) \biguplus_{\forall e=(u,v)\in E^c \; s.t.\; \delta_{i-1}^T(u)\neq\varnothing} \delta_{i-1}^T(u)$$



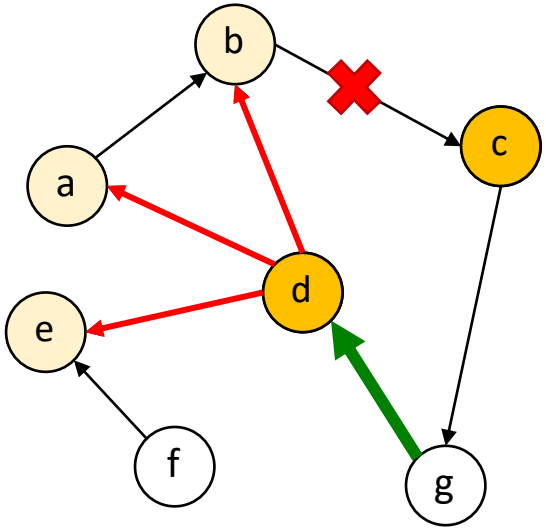
DelZero-Aware Incremental Refinement

- Express **incremental refinement recursively**

$$g_i^T(v) = g_i(v) \oplus \Delta_i^T(v)$$

$$\Delta_i^T(v) = \Delta_{i-1}^T(v) \biguplus_{\substack{\forall e=(u,v)\in E_a \\ s.t.\ \delta_{i-1}(u)\neq\varnothing}} \delta_{i-1}(u) \biguplus_{\substack{\forall e=(u,v)\in E_d \\ s.t.\ \delta_{i-1}(u)\neq\varnothing}} \delta_{i-1}(u)$$

$$\biguplus_{\substack{\forall e=(u,v)\in E^c \\ s.t.\ \delta_{i-1}(u)\neq\varnothing}} \delta_{i-1}(u) \biguplus_{\substack{\forall e=(u,v)\in E^c \\ s.t.\ \delta_{i-1}^T(u)\neq\varnothing}} \delta_{i-1}^T(u)$$



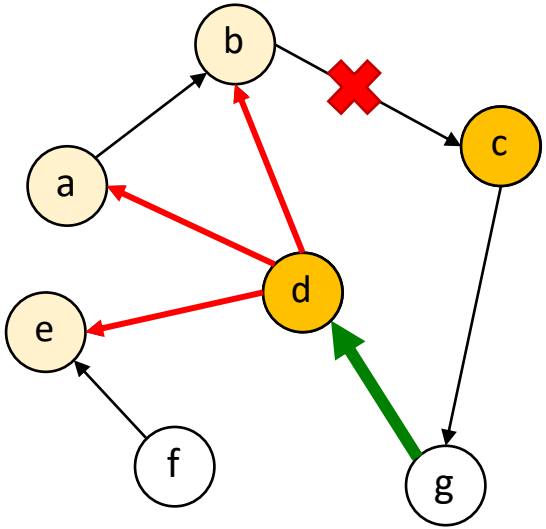
DelZero-Aware Incremental Refinement

- Express **incremental refinement recursively**

$$g_i^T(v) = g_i(v) \oplus \Delta_i^T(v)$$

$$\Delta_i^T(v) = \Delta_{i-1}^T(v) \biguplus_{\substack{\forall e=(u,v)\in E_a \\ s.t.\ \delta_{i-1}(u)\neq\varnothing}} \delta_{i-1}(u) \biguplus_{\substack{\forall e=(u,v)\in E_d \\ s.t.\ \delta_{i-1}(u)\neq\varnothing}} \delta_{i-1}(u)$$

$$\biguplus_{\substack{\forall e=(u,v)\in E^c \\ s.t.\ \delta_{i-1}(u)\neq\varnothing}} \delta_{i-1}(u) \biguplus_{\substack{\forall e=(u,v)\in E^c \\ s.t.\ \delta_{i-1}^T(u)\neq\varnothing}} \delta_{i-1}^T(u)$$



DelZero-Aware Incremental Refinement

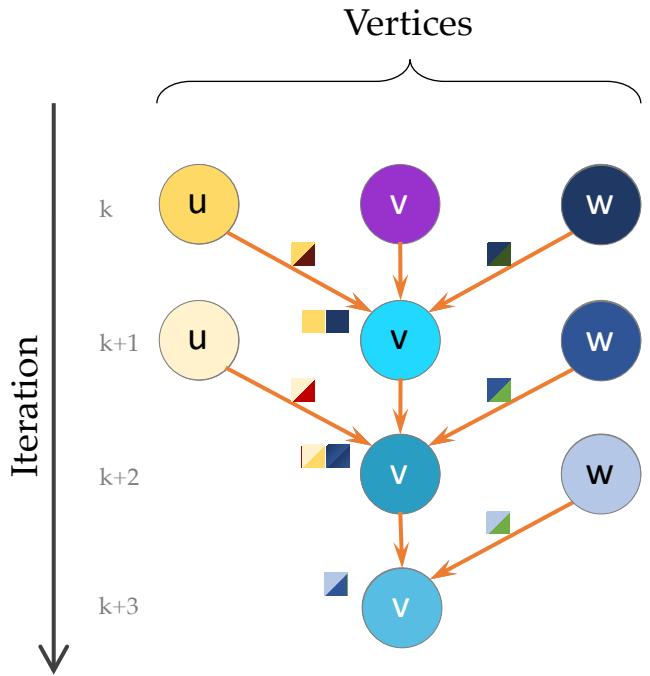
- Express **incremental refinement recursively**

$$g_i^T(v) = g_i(v) \oplus \Delta_i^T(v)$$

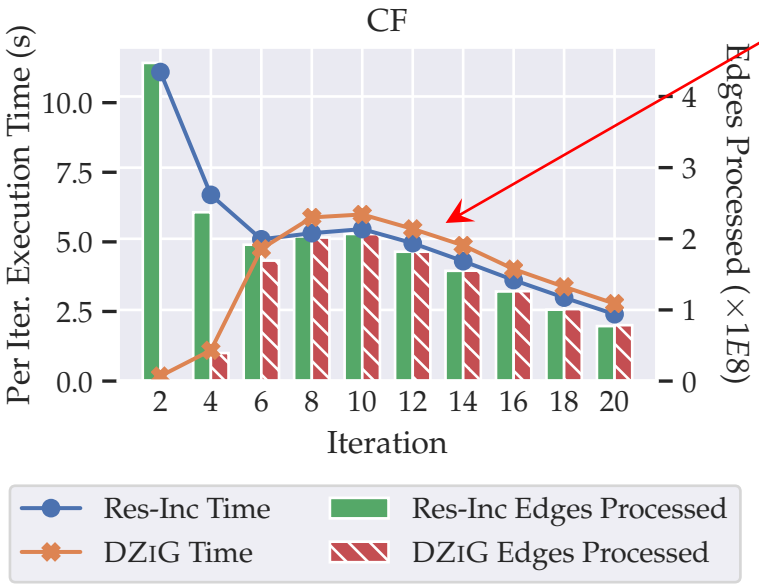
$$\Delta_i^T(v) = \Delta_{i-1}^T(v) \biguplus_{\substack{\forall e=(u,v)\in E_a \\ s.t.\ \delta_{i-1}(u)\neq\varnothing}} \delta_{i-1}(u) \biguplus_{\substack{\forall e=(u,v)\in E_d \\ s.t.\ \delta_{i-1}(u)\neq\varnothing}} \delta_{i-1}(u)$$

$$\biguplus_{\substack{\forall e=(u,v)\in E^c \\ s.t.\ \delta_{i-1}(u)\neq\varnothing}} \delta_{i-1}(u) \biguplus_{\substack{\forall e=(u,v)\in E^c \\ s.t.\ \delta_{i-1}^T(u)\neq\varnothing}} \delta_{i-1}^T(u)$$

Correctness Proof in Paper!

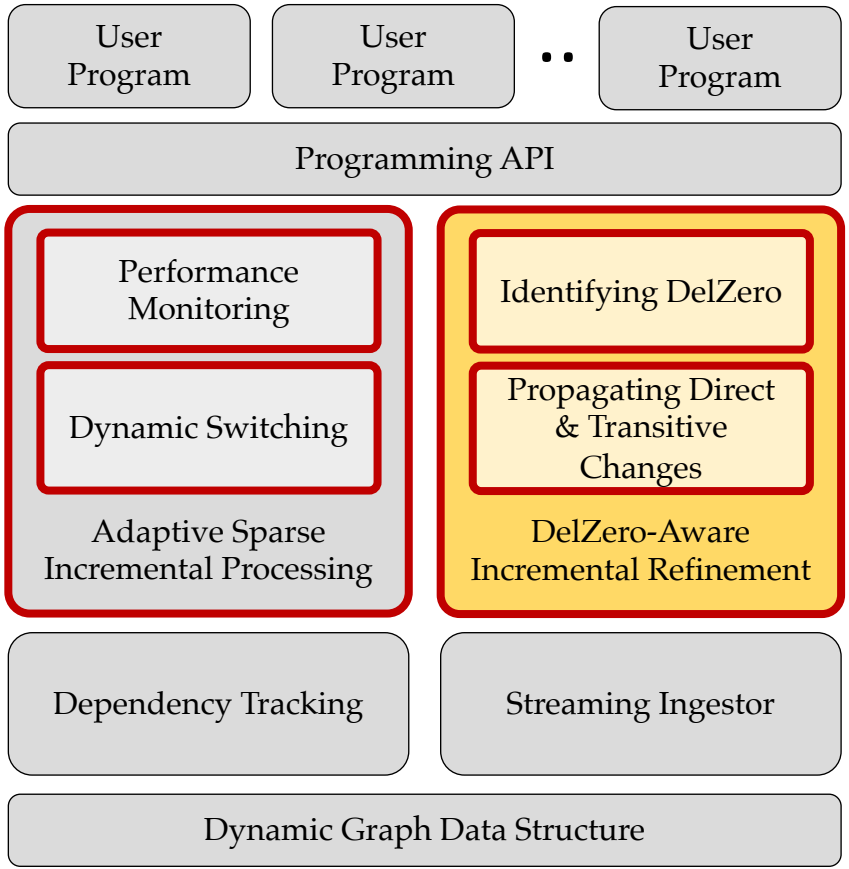


Adaptive Sparse Incremental Processing

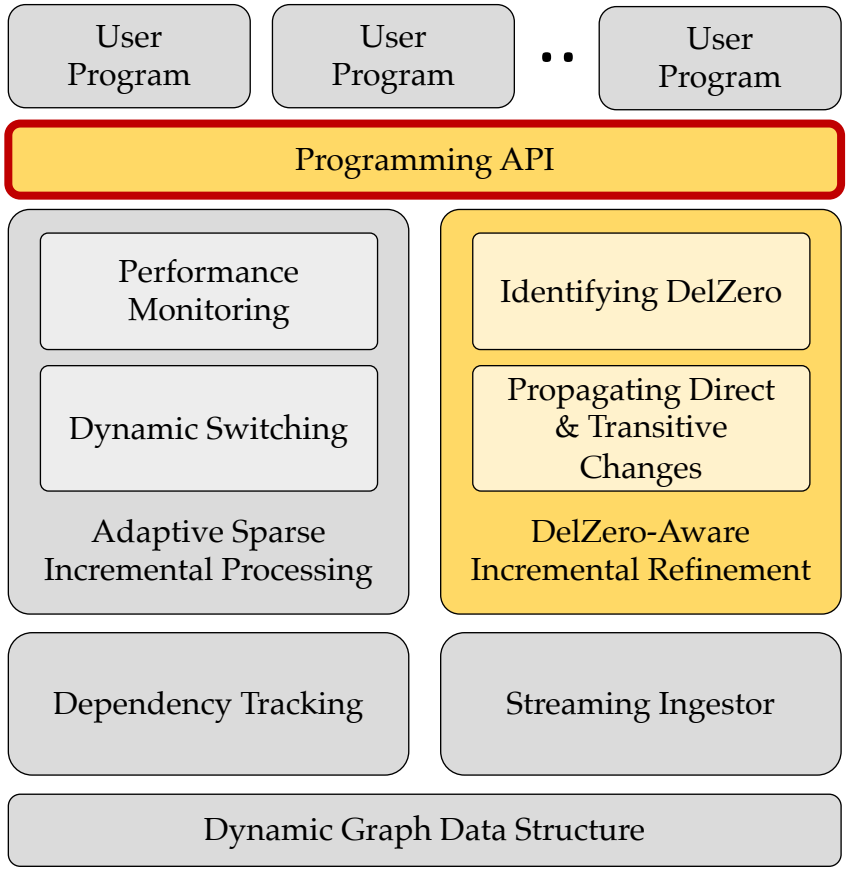


- Minor overhead due to **two ops per edge** (retract old value & propagate new value)
- **Automatically switch** to traditional incremental processing
- **Linear regression** to estimate iteration time

DZiG System


User Program
User Program
..
User Program
Programming API
Performance Monitoring
Dynamic Switching
Adaptive Sparse Incremental Processing
Identifying DelZero
Propagating Direct & Transitive Changes
DelZero-Aware Incremental Refinement
Dependency Tracking
Streaming Ingestor
Dynamic Graph Data Structure

DZiG System


User Program
User Program
..
User Program
Programming API
Performance Monitoring
Dynamic Switching
Adaptive Sparse Incremental Processing
Identifying DelZero
Propagating Direct & Transitive Changes
DelZero-Aware Incremental Refinement
Dependency Tracking
Streaming Ingestor
Dynamic Graph Data Structure

Extracting DelZeros

PageRank:

$$PR(u) = 0.15 + 0.85 \times \sum_{(u,v)\in G} \frac{PR(v)\times W(u,v)}{N(v)}$$

```
bool notDelZero(VValueType oldVal, VValueType newVal) {
  return fabs(newVal - oldVal) > epsilon;
}
DeltaType vertexChange(VId v, VValueType oldVal,
                       VValueType newVal, Graph G) {
  return (newVal - oldVal) / G.out_degree[v];
}
DeltaType edgeChange(Edge e, DeltaType change,
     VValueType oldVal, VValueType newVal, Graph G) {
  return change * G.edge_weight[e];
}
void addChange(AggregationType* aggr, DeltaType delta) {
  atomicAdd(aggr, delta);
}
void removeChange(AggregationType* aggr, DeltaType delta){
  atomicSubtract(aggr, delta);
}
VValueType computeVertex(VId v, AggregationType aggr,
                         VValueType oldVal, Graph G) {
  return 0.15 + (0.85 * aggr);
}
```

Extracting DelZeros

PageRank:

$$PR(u) = 0.15 + 0.85 \times \sum_{(u,v)\in G} \frac{PR(v)\times W(u,v)}{N(v)}$$

```
bool notDelZero(VValueType oldVal, VValueType newVal) {
  return fabs(newVal - oldVal) > epsilon;
}
DeltaType vertexChange(VId v, VValueType oldVal,
                       VValueType newVal, Graph G) {
  return (newVal - oldVal) / G.out_degree[v];
}
DeltaType edgeChange(Edge e, DeltaType change,
     VValueType oldVal, VValueType newVal, Graph G) {
  return change * G.edge_weight[e];
}
void addChange(AggregationType* aggr, DeltaType delta) {
  atomicAdd(aggr, delta);
}
void removeChange(AggregationType* aggr, DeltaType delta){
  atomicSubtract(aggr, delta);
}
VValueType computeVertex(VId v, AggregationType aggr,
                         VValueType oldVal, Graph G) {
  return 0.15 + (0.85 * aggr);
}
```

Extracting DelZeros

PageRank:

$$PR(u) = 0.15 + 0.85 \times \sum_{(u,v)\in G} \frac{PR(v)\times W(u,v)}{N(v)}$$

```
bool notDelZero(VValueType oldVal, VValueType newVal) {
  return fabs(newVal - oldVal) > epsilon;
}
DeltaType vertexChange(VId v, VValueType oldVal,
                       VValueType newVal, Graph G) {
  return (newVal - oldVal) / G.out_degree[v];
}
DeltaType edgeChange(Edge e, DeltaType change,
     VValueType oldVal, VValueType newVal, Graph G) {
  return change * G.edge_weight[e];
}
void addChange(AggregationType* aggr, DeltaType delta) {
  atomicAdd(aggr, delta);
}
void removeChange(AggregationType* aggr, DeltaType delta){
  atomicSubtract(aggr, delta);
}
VValueType computeVertex(VId v, AggregationType aggr,
                         VValueType oldVal, Graph G) {
  return 0.15 + (0.85 * aggr);
}
```

Extracting DelZeros

PageRank:

$$PR(u) = 0.15 + 0.85 \times \sum_{(u,v)\in G} \frac{PR(v)\times W(u,v)}{N(v)}$$

```
bool notDelZero(VValueType oldVal, VValueType newVal) {
  return fabs(newVal - oldVal) > epsilon;
}
DeltaType vertexChange(VId v, VValueType oldVal,
                       VValueType newVal, Graph G) {
  return (newVal - oldVal) / G.out_degree[v];
}
DeltaType edgeChange(Edge e, DeltaType change,
     VValueType oldVal, VValueType newVal, Graph G) {
  return change * G.edge_weight[e];
}
void addRemoveChange(AggregationType aggr,
     DeltaType oldDelta, DeltaType newDelta)
  atomicAdd(&aggr, newDelta - oldDelta);
}
VValueType computeVertex(VId v, AggregationType aggr,
                         VValueType oldVal, Graph G) {
  return 0.15 + (0.85 * aggr);
}
```

Experimental Setup

- **Graph Benchmarks:**
  PageRank (PR), Belief Propagation (BP), Collaborative Filtering (CF),
  Co-Training Expectation Maximization (CoEM), Label Propagation (LP)

Graph	Edges	Vertices
UKDomain (UK) [7]	1.0B	39.5M
Twitter (TW) [28]	1.5B	41.7M
TwitterMPI (TT) [10]	2.0B	52.6M
Friendster (FT) [16]	2.5B	68.3M
Yahoo (YH) [62]	6.6B	1.4B

- **Server:** 32-cores @ 2 GHz, 231 GB RAM

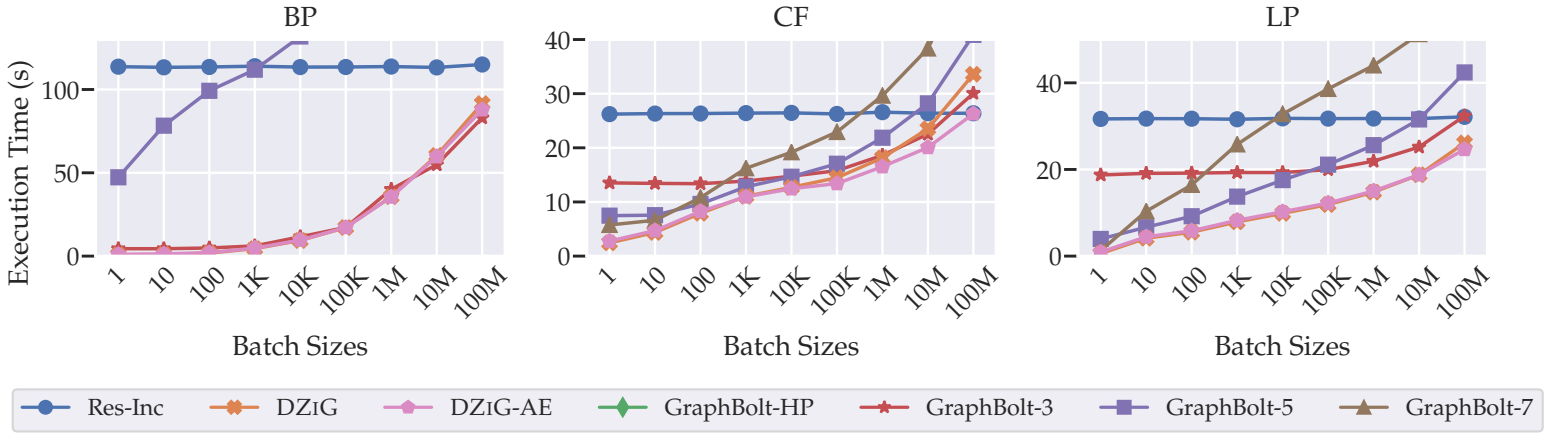
DZiG Performance

	PR-1K			BP-1K			CF-1K			CoEM-1K			LP-1K		
	UK	TT	FT	UK	TT	FT	UK	TT	FT	UK	TT	FT	UK	TT	FT
DZiG	0.578	11.2	21	1.4	4.77	1.23	4.71	11.1	4.25	0.73	7.64	5.43	0.609	8.05	6.03
Res-Inc	4.38	15.8	29.6	35.8	114	164	10.2	26.5	43.1	5.5	22.9	31.9	9.11	31.6	49.8
GraphBolt	1.5	11.7	21.4	2.7	6.37	5	7.83	12.9	8.39	0.983	7.52	5.63	3.18	12.3	9.25

Execution time (in seconds)

- Up to **133x faster** than restarting from scratch
- Comparable performance with the best hand-picked GraphBolt switching
- Up to **5.22x faster** than GraphBolt

DZiG Performance



- **Pushes scale** of dependency-driven incremental processing
- Adaptive Sparse Incremental Processing helps for large batch sizes
- **Better & consistent performance**

Comparison with Other Frameworks

	Graph Update			PageRank			SSSP		
	1	100	10K	1	100	10K	1	100	10K
DZiG	0.06	0.09	0.33	8.62	10.56	11.20	0.05	0.05	0.09
GraphBolt	1.15	1.17	1.42	9.04	11.14	12.25	0.05	0.05	0.09
Aspen	1E-4	2E-3	7E-3	29.48	29.29	29.68	3.31	3.31	3.27
GraphOne	4E-3	0.09	1.64	19.91	19.73	20.02	13.24	13.09	18.41
LLAMA	x	x	x	20.55	20.55	20.55	2.77	2.77	2.77
Stinger	0.02	2.88	7.06	431.18	437.11	488.78	145.15	145.12	147.17

Execution time (in seconds)

- **Significantly lower end-to-end processing times** (update + analytics time)
- Up to **3.39x/30.1x faster than Aspen** on PageRank/SSSP
- Up to **2.2x/120x faster than GraphOne** on PageRank/SSSP

Comparison with Other Frameworks

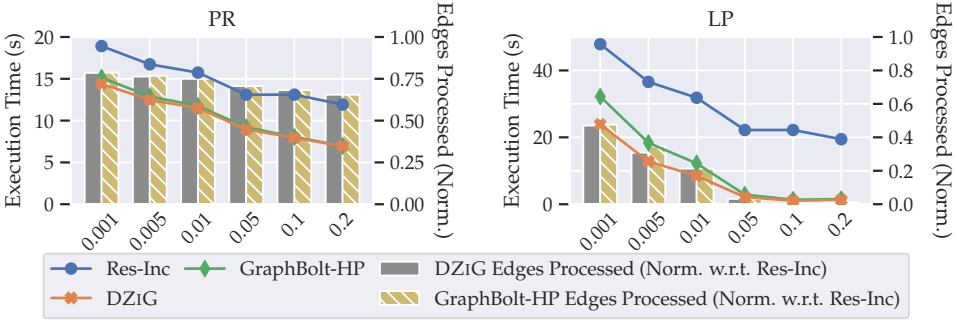
	Graph Update			PageRank			SSSP		
	1	100	10K	1	100	10K	1	100	10K
DZIG	0.06	0.09	0.33	8.62	10.56	11.20	0.05	0.05	0.09
GraphBolt	1.15	1.17	1.42	9.04	11.14	12.25	0.05	0.05	0.09
Aspen	1E-4	2E-3	7E-3	29.48	29.29	29.68	3.31	3.31	3.27
GraphOne	4E-3	0.09	1.64	19.91	19.73	20.02	13.24	13.09	18.41
LLAMA	x	x	x	20.55	20.55	20.55	2.77	2.77	2.77
Stinger	0.02	2.88	7.06	431.18	437.11	488.78	145.15	145.12	147.17

Execution time (in seconds)

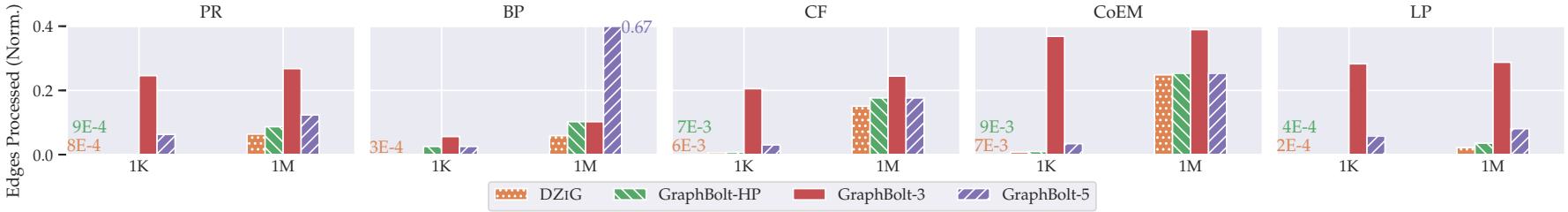
- **Significantly lower end-to-end processing times** (update + analytics time)
- Up to **3.39x/30.1x faster than Aspen** on PageRank/SSSP
- Up to **2.2x/120x faster than GraphOne** on PageRank/SSSP

Other Experiments

	1K					1M				
	PR	BP	CF	CoEM	LP	PR	BP	CF	CoEM	LP
DZIG	1.21	4.51	6.89	3.54	2.46	1.71	10.2	9.45	4.2	2.82
GraphBolt-HP	3.73	14.1	12.3	3.05	3.68	4.49	84.2	15.3	5.47	4.34
GraphBolt-3	5.35	14.4	16.7	6.29	9.31	5.45	84.2	16.4	6.56	9.14
GraphBolt-5	4.54	14.7	13.5	3.05	5.41	4.66	87.6	15.6	5.47	5.96
Res-Inc	4.97	70.6	17.7	13.2	13.2	5.27	68.1	17.4	13	12.7



	UK		TW		TT		FT	
	DZIG	GB	DZIG	GB	DZIG	GB	DZIG	GB
PR	4.8%	5.4%	3.8%	6.1%	3.9%	4.3%	4.1%	4.7%
BP	10.3%	10.3%	9.0%	8.9%	10.0%	9.1%	9.6%	8.8%
CF	12.9%	19.0%	11.4%	15.7%	11.4%	17.1%	12.1%	18.3%
CoEM	5.4%	7.7%	4.5%	4.4%	4.5%	4.4%	4.9%	4.9%
LP	7.9%	7.6%	6.7%	6.5%	5.6%	6.5%	7.0%	6.0%



Conclusion

`github.com/pdclab/graphbolt`

- Dependency-Driven incremental processing important to process dynamic graphs
- **DelZero-Aware incremental refinement** crucial to retain high performance
  - Pushes the scale of dependency-driven incremental processing to **high mutation rates**
- **Adaptive sparse incremental** processing
  - **Automatic switching** based on **performance monitoring**
- Efficient programming model **naturally exposes DelZeros**
  - **Merged edge updates** for performance
- Efficient **dynamic graph data structure**
- DZiG outperforms state-of-the-art solutions