Parallel Breadth First Search on GPU Clusters

http://mapgraph.io

Zhisong Fu
SYSTAP, LLC
fuzhisong@systap.com

Harish Kumar Dasari
University of Utah
hdasari@sci.utah.edu

Bradley Bebee
SYSTAP, LLC
beebs@systap.com

Martin Berzins
University of Utah
mb@sci.utah.edu

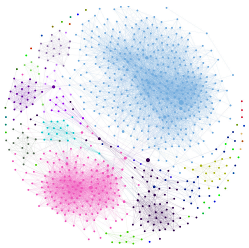
Bryan Thompson
SYSTAP, LLC
bryan@systap.com
(Presenting)

This work was (partially) funded by the DARPA XDATA program under AFRL Contract #FA8750-13-C-0002. This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. D14PC00029. The authors wo˙uld like to thank Dr. White, NVIDIA, and the MVAPICH group at Ohio State University for their support of this work.

Z. Fu, H.K. Dasari, B. Bebee, M. Berzins, B. Thompson. Parallel Breadth First Search on GPU Clusters. IEEE Big Data. Bethesda, MD. 2014.

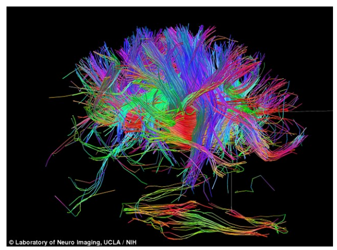
Many-Core is Your Future

Graphs are everywhere and need for graph analytics is growing rapidly.



Communication and Social Networks

Human Brain and Biological Networks



E-Commerce and Online Service Analytics
Real-time Fraud Detection

- *Facebook has ~ 1 trillion edges in their graph.*
- *Over 30 minutes per iteration using 200 machines.*
- *All computations require multiple iterations (6-50).*
- *We could do it in seconds on a cluster of GPUs.*

SYSTAP, LLC

Small Business, Founded 2006

100% Employee Owned

Graph Database

- High performance, Scalable
  - 50B edges/node
  - High level query language
  - Efficient Graph Traversal
  - High 9s solution
- Open Source
  - Subscriptions

GPU Analytics

- Extreme Performance
  - 100s of times faster than CPUs
  - 10,000x faster than graph databases
  - 30,000,000,000 edges/sec on 64 GPUs
- DARPA funding
- Disruptive technology
  - Early adopters
  - Huge ROIs

Customers Powering Their Graphs with SYSTAP

Information Management / Retrieval

AUTODESK

EMC²

YAHOO!7

WIKIMEDIA FOUNDATION

WIKIDATA

Genomics / Precision Medicine

syapse

FOUNDATION MEDICINE

NESCent
National Evolutionary Synthesis Center

Harvard Medical School

Defense, Intel, Cyber

DARPA

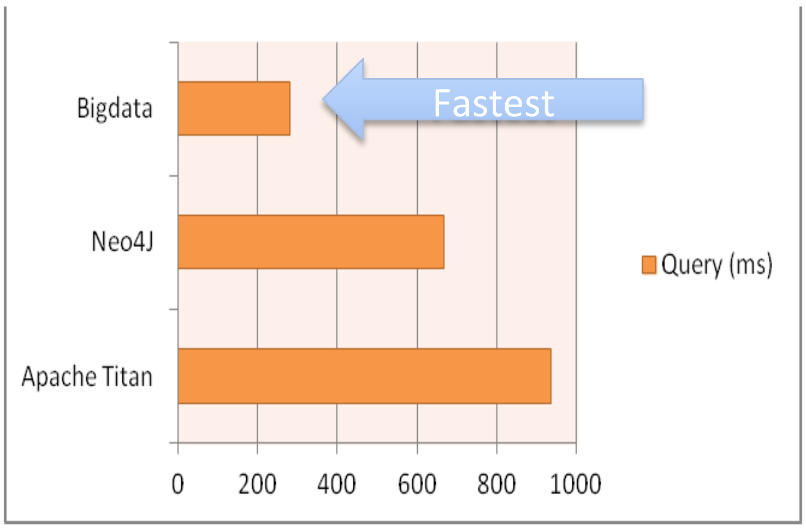
Chesapeake Technology International

IMAGE MATTERS
Locate. Innovate.

SYSTAP is focused on building software that enable graphs at speed and scale.

- Blazegraph™ for Property and RDF Graphs
  - High Availability (HA) Architecture with Horizontal Scaling

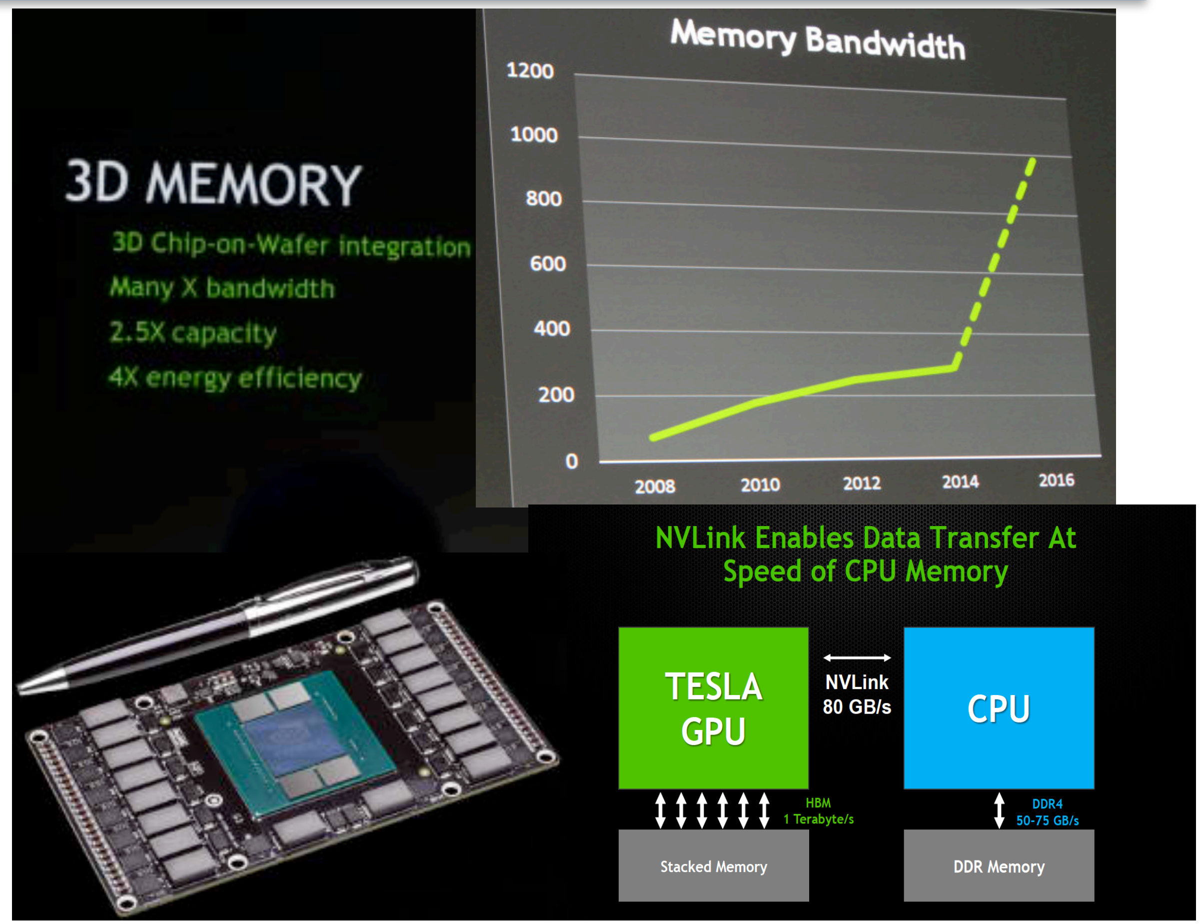
Blazegraph™ Comparison



- Mapgraph™ GPU-accelerated data parallel graph analytics
  - Vertex-Centric API.
  - Single or multi-GPU.
  - 10,000X faster than Accumulo, Neo4J, Titan, etc.
  - 3x faster then Cray XMT-2 at 1/3rd the price.

GPU Hardware Trends

- K40 GPU (today)
  - 12G RAM/GPU
  - 288 GB/s bandwidth
  - PCIe Gen 3
- Pascal GPU (Q1 2016)
  - 32G RAM/GPU
  - 1 TB/s bandwidth
  - Unified memory across CPU, GPUs
- NVLINK
  - High bandwidth access to host memory



MapGraph: Extreme performance

- GTEPS is Billions (10^9) of Traversed Edges per Second.
  - This is the basic measure of performance for graph traversal.

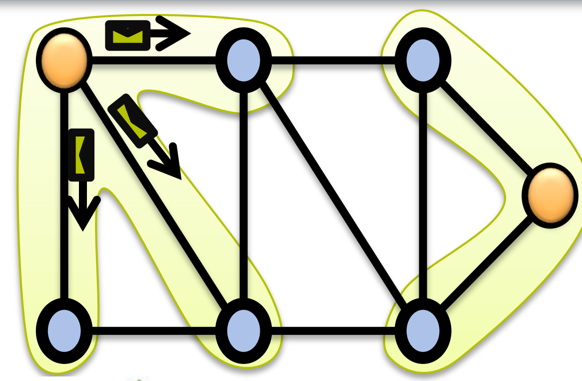


Configuration	Cost	GTEPS	$/GTEPS
4-Core CPU	$4,000	0.2	$5,333
4-Core CPU + K20 GPU	**$7,000**	**3.0**	**$2,333**
XMT-2 (rumored price)	$1,800,000	10.0	$188,000
64 GPUs (32 nodes with 2x K20 GPUs per node and InfiniBand DDRx4 – today)	**$500,000**	**30.0**	**$16,666**
16 GPUs (2 nodes with 8x Pascal GPUs per node and InfiniBand DDRx4 – Q1, 2016)	**$125,000**	**>30.0**	**<$4,166**

SYSTAP's MapGraph APIs Roadmap

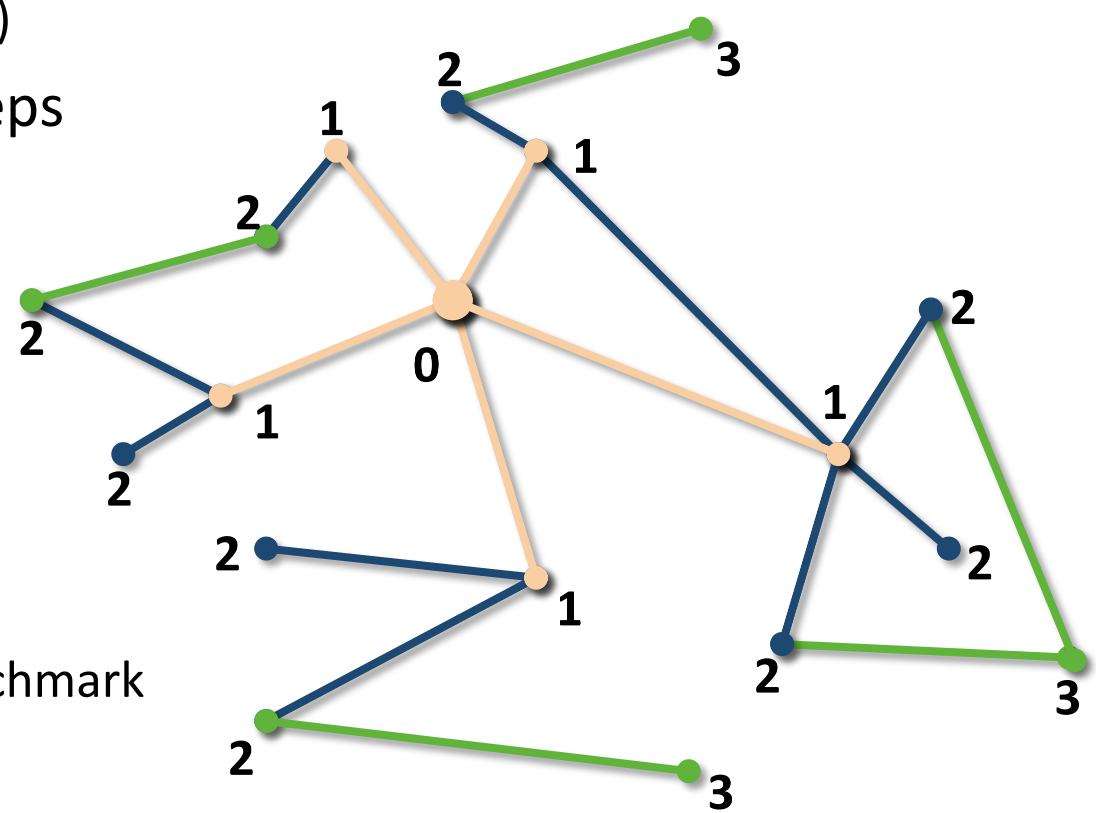
- Vertex Centric API
  - Same performance as CUDA.
- Schema-flexible data-model
  - Property Graph / RDF
  - Reduce import hassles
  - Operations over merged graphs
- Graph pattern matching language
- DSL/Scala => CUDA code generation

Ease of Use



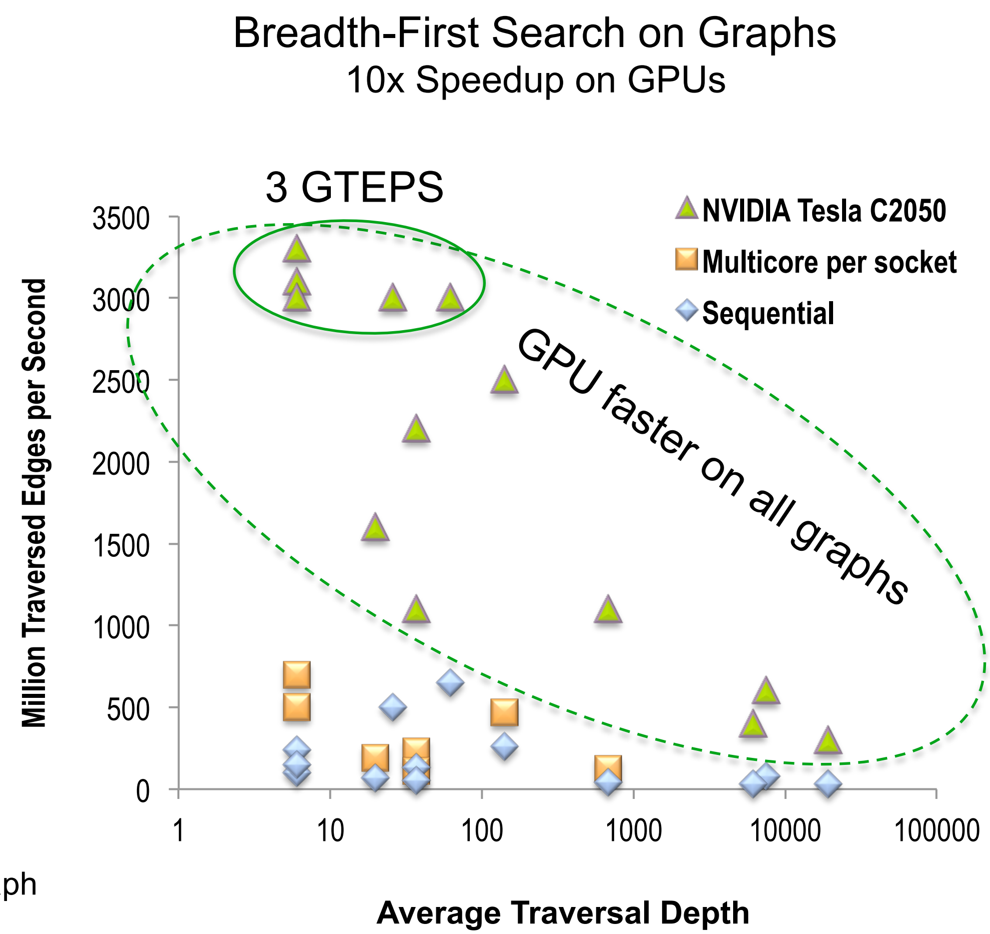
Breadth First Search

- Fundamental building block for graph algorithms
  - Including SPARQL (JOINs)
- In level synchronous steps
  - Label visited vertices
- For this graph
  - Iteration 0
  - Iteration 1
  - Iteration 2
- Hard problem!
  - Basis for Graph 500 benchmark



GPUs – A Game Changer for Graph Analytics

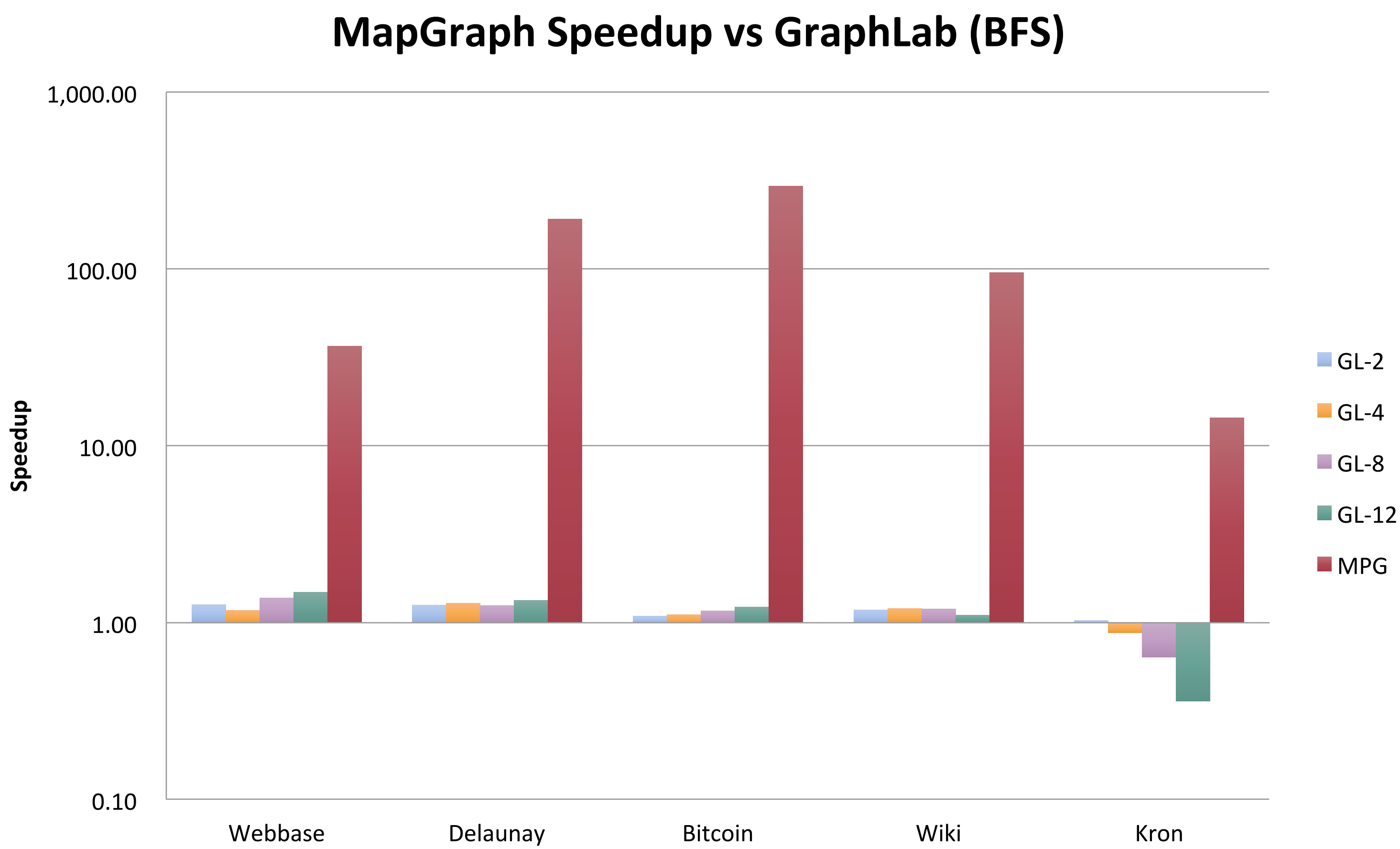
- Graphs are a hard problem
  - Non-locality
  - Data dependent parallelism
  - Memory bus and communication bottlenecks
- GPUs deliver effective parallelism
  - 10x+ memory bandwidth
  - Dynamic parallelism



Graphic: Merrill, Garland, and Grimshaw, “GPU Sparse Graph Traversal”, GPU Technology Conference, 2012

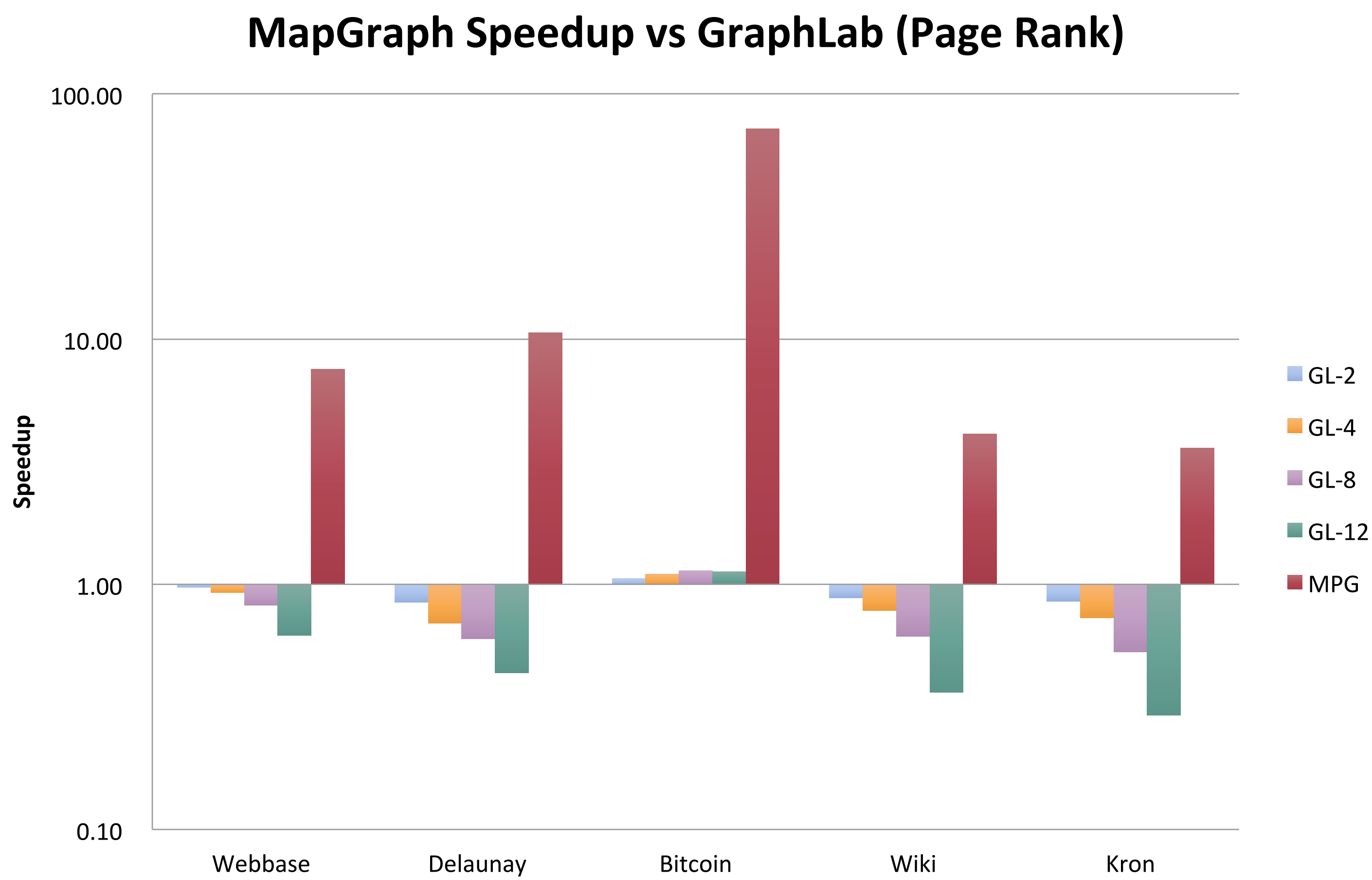
BFS Results : MapGraph vs GraphLab

- CPU vs GPU
- GPU 15x-300x faster
- More CPU cores does not help



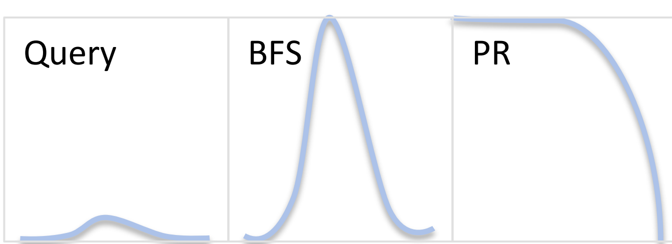
PageRank : MapGraph vs GraphLab

- CPU vs GPU
- GPU 5x-90x faster
- CPU slows down with more cores!

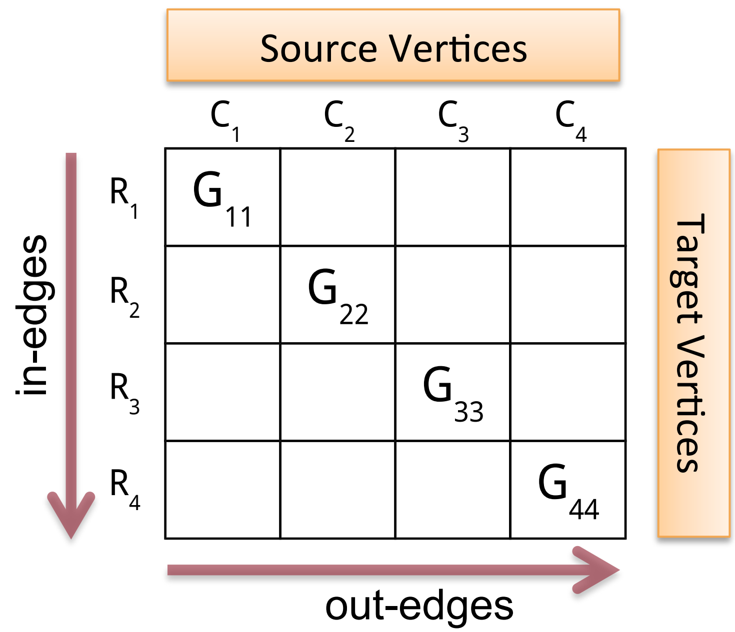


2D Partitioning (aka Vertex Cuts)

- p x p compute grid
  - Edges in rows/cols
  - Minimize messages
    - log(p) (versus p^2)
  - One partition per GPU
- Batch parallel operation
  - Grid row: out-edges
  - Grid column: in-edges
- Representative frontiers

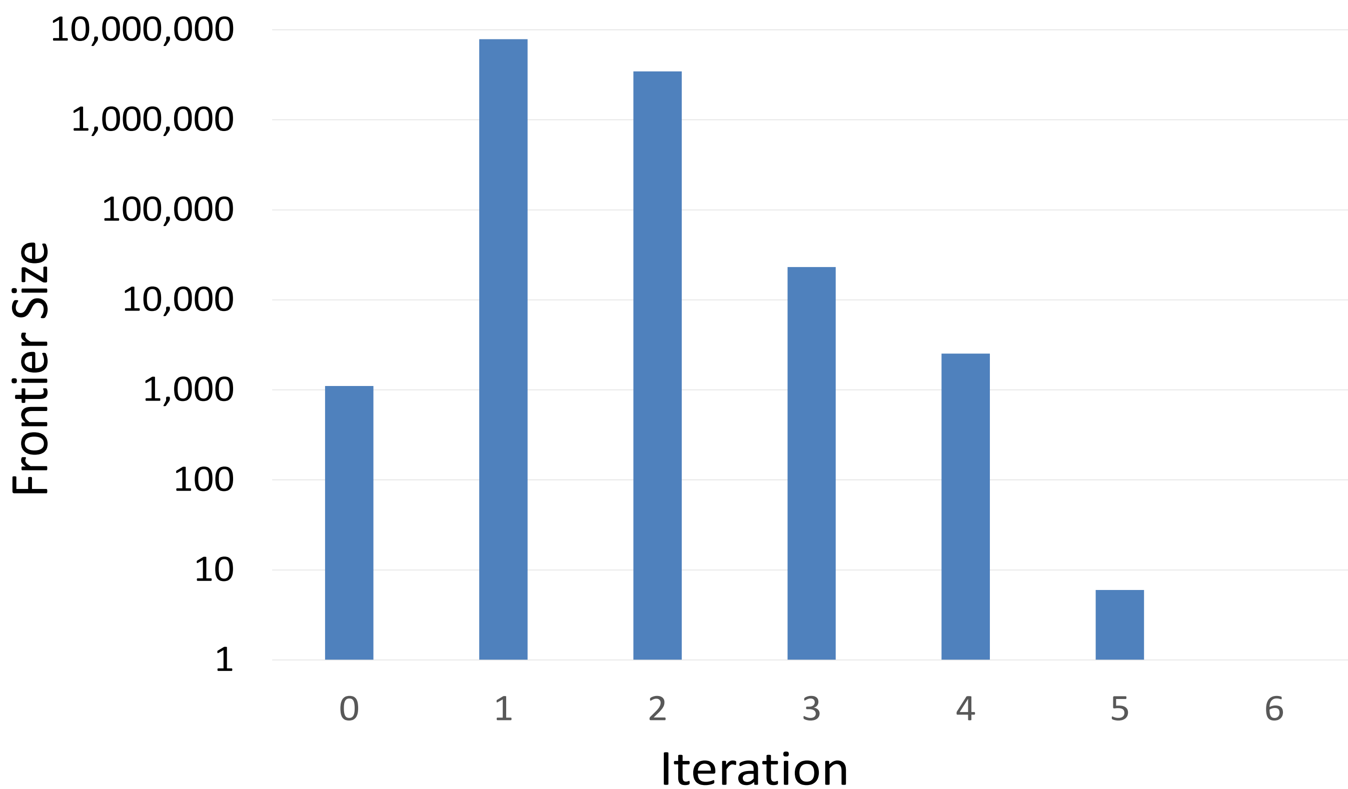


- Parallelism – work must be distributed and balanced.
- Memory bandwidth – memory, not disk, is the bottleneck



Scale 25 Traversal

- Work spans multiple orders of magnitude.



Distributed BFS Algorithm

procedure BFS(*Root*, *Predecessor*)
  $In^0_{ij} \leftarrow \text{LocalVertex}(Root)$ ← Starting vertex
  for $t \leftarrow 0$ **do**
    $\text{Expand}(In^t_i, Out^t_{ij})$ ← Data parallel 1-hop expand on all GPUs
    $LocalFrontier_t \leftarrow \text{Count}(Out^t_{ij})$ ← Local frontier size
    $GlobalFrontier_t \leftarrow \text{Reduce}(LocalFrontier_t)$ ← Global frontier size
    if $GlobalFrontier_t > 0$ **then** ← Termination check
      $\text{Contract}(Out^t_{ij}, Out^t_j, In^{t+1}_i, Assign_{ij})$ ← Global Frontier contraction ("wave")
      $\text{UpdateLevels}(Out^t_j, t, level)$ ← Record level of vertices in the In / Out frontier
    else
      $\text{UpdatePreds}(Assign_{ij}, Preds_{ij}, level)$ ← Compute predecessors from local In / Out levels (no communication)
      break ← Done
    end if
    t++ ← Next iteration
  end for
end procedure

Distributed BFS Algorithm

procedure BFS($Root$, $Predecessor$)
    $In^0_{ij} \leftarrow$ LocalVertex($Root$) ← Starting vertex
    for $t \leftarrow 0$ **do**
        Expand(In^t_i, Out^t_{ij}) ← Data parallel 1-hop expand on all GPUs
        $LocalFrontier_t \leftarrow$ Count(Out^t_{ij}) ← Local frontier size
        $GlobalFrontier_t \leftarrow$ Reduce($LocalFrontier_t$) ← Global frontier size
        if $GlobalFrontier_t > 0$ **then** ← Termination check
            Contract(Out^t_{ij}, Out^t_j, In^{t+1}_i, $Assign_{ij}$) ← Global Frontier contraction (“wave”)
            UpdateLevels(Out^t_j, t, $level$) ← Record level of vertices in the In / Out frontier
        else
            UpdatePreds($Assign_{ij}$,$Preds_{ij}$, $level$) ← Compute predecessors from local In / Out levels (no communication)
            break ← Done
        end if
        t++ ← Next iteration
    end for
end procedure

- Key differences
  - log(p) parallel scan (vs sequential wave)
  - GPU-local computation of predecessors
  - 1 partition per GPU
  - GPUDirect (vs RDMA)

Expand

procedure EXPAND(In_i^t, Out_{ij}^t)
  $L_{in} \leftarrow \text{convert}(In_i^t)$
  $L_{out} \leftarrow \emptyset$
  for all $v \in L_{in}$ in parallel **do**
    for $i \leftarrow \text{RowOff}[v], \text{RowOff}[v+1]$ **do**
      $c \leftarrow \text{ColIdx}[i]$
      $L_{out} \leftarrow c$
    end for
  end for
  $Out_{ij}^t \leftarrow \text{convert}(L_{out})$
end procedure

- The GPU implementation uses multiple strategies to handle data-dependent parallelism. See our SIGMOD 2014 paper for details.

Global Frontier Contraction and Communication

procedure $\textsc{Contract}(Out^t_{ij}, Out^t_j, In^{t+1}_i, Assign_{ij})$

$\text{Prefix}^t_{ij} \leftarrow \text{ExclusiveScan}_j(Out^t_{ij})$ ← Distributed prefix sum. Prefix is vertices discovered by your left neighbors (log p)

$Assigned_{ij} \leftarrow \text{Assigned}_{ij} \cup (Out^t_{ij} - \text{Prefix}^t_{ij})$ ← Vertices first discovered by this GPU.

if $i = p$ **then**

$Out^t_j \leftarrow Out^t_{ij} \cup \text{Prefix}^t_{ij}$ ← Right most column has global frontier for the row

end if

$\text{Broadcast}(Out^t_j, p, ROW)$ ← Broadcast frontier over row.

if $i = j$ **then**

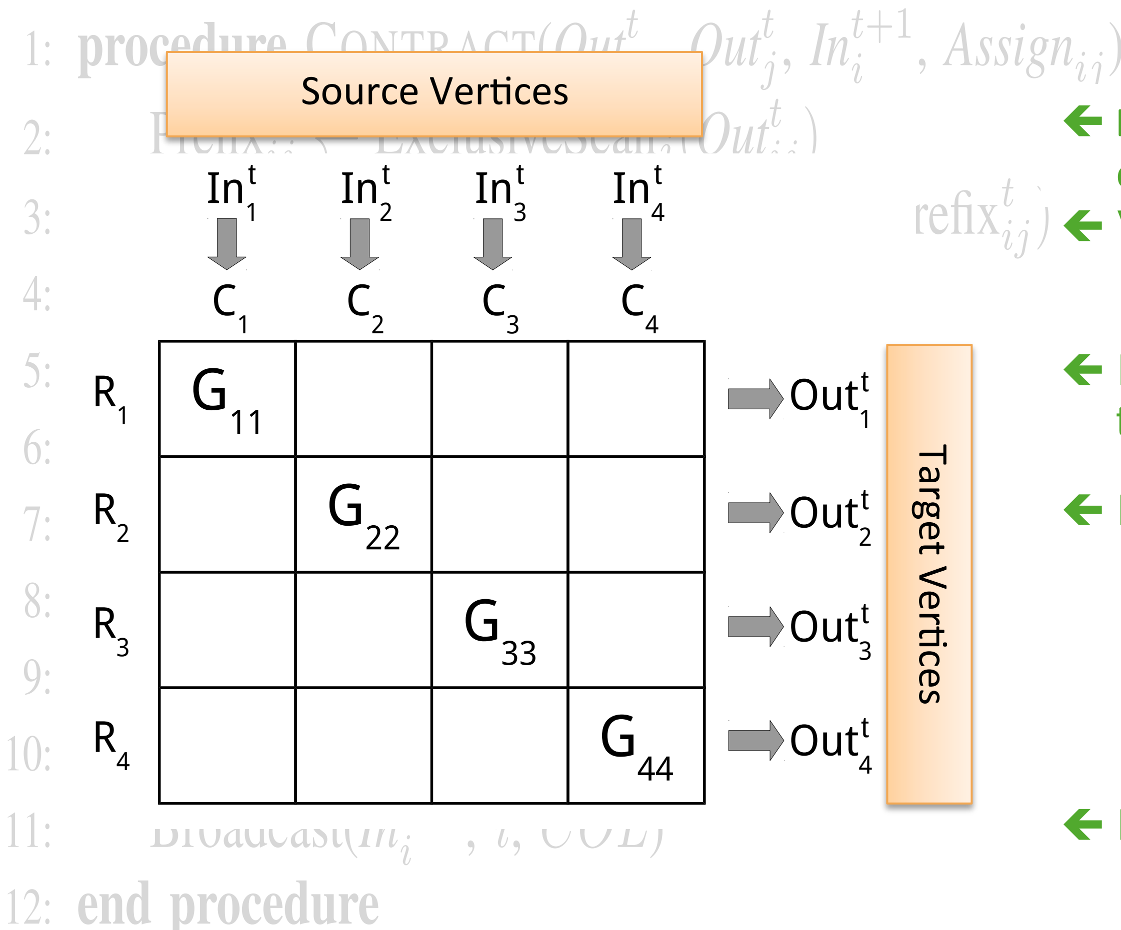
$In^{t+1}_i \leftarrow Out^t_j$

end if

$\text{Broadcast}(In^{t+1}_i, i, COL)$ ← Broadcast frontier over column.

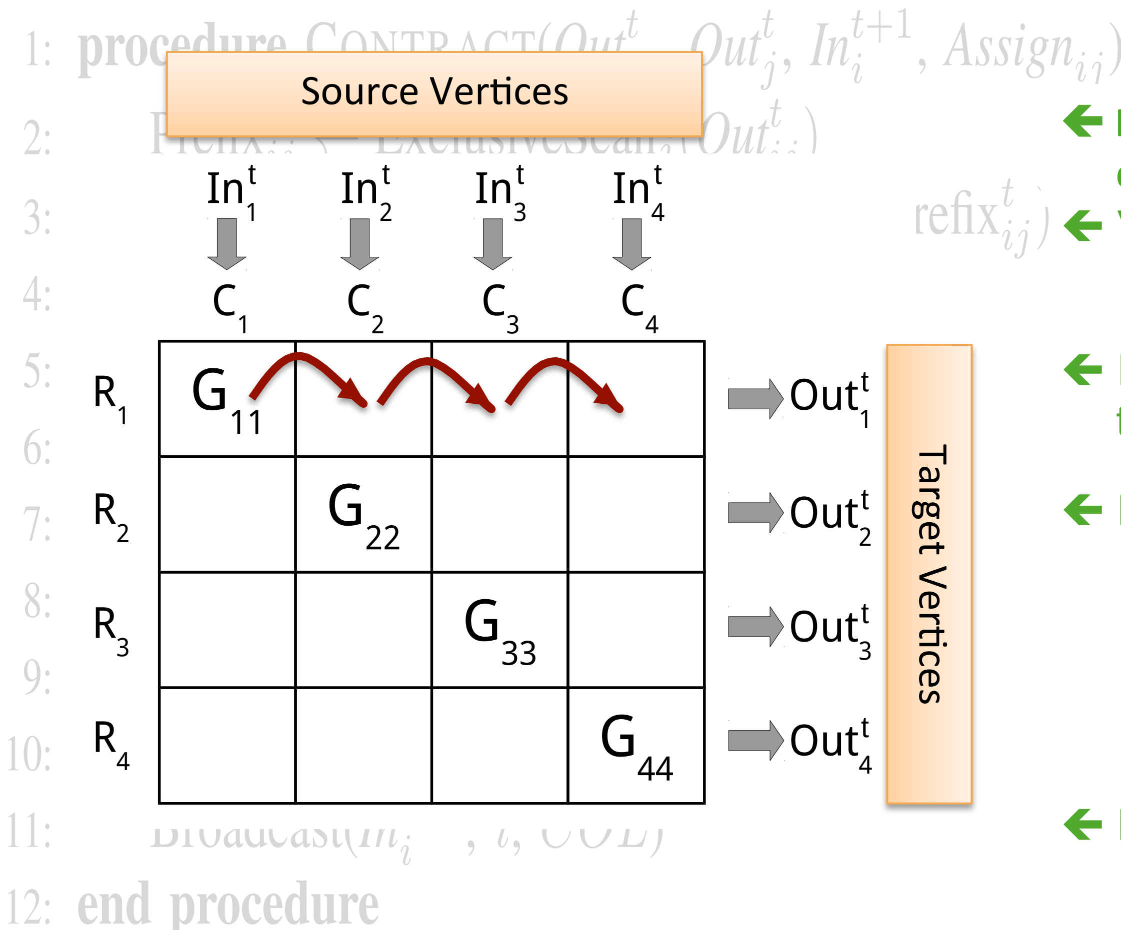
end procedure

Global Frontier Contraction and Communication



- ← Distributed prefix sum. Prefix is vertices discovered by your left neighbors (log p)
- ← Vertices first discovered by this GPU.
- ← Right most column has global frontier for the row
- ← Broadcast frontier over row.
- ← Broadcast frontier over column.

Global Frontier Contraction and Communication



← **Distributed prefix sum. Prefix is vertices discovered by your left neighbors (log p)**

← Vertices first discovered by this GPU.

← Right most column has global frontier for the row

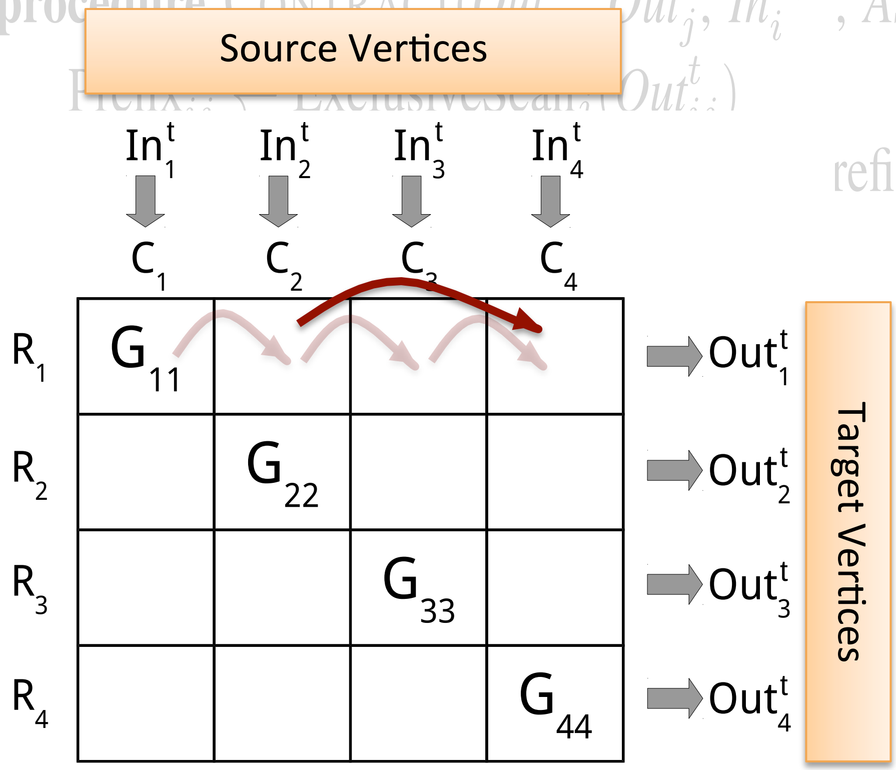
← Broadcast frontier over row.

← Broadcast frontier over column.

- We use a parallel scan that minimizes communication steps (vs work).
- This improves the overall scaling efficiency by 30%.

Global Frontier Contraction and Communication

procedure CONTRACT($Out^t_{..}$, Out^t_j, In^{t+1}_i, $Assign_{ij}$)



← **Distributed prefix sum. Prefix is vertices discovered by your left neighbors (log p)**

← Vertices first discovered by this GPU.

← Right most column has global frontier for the row

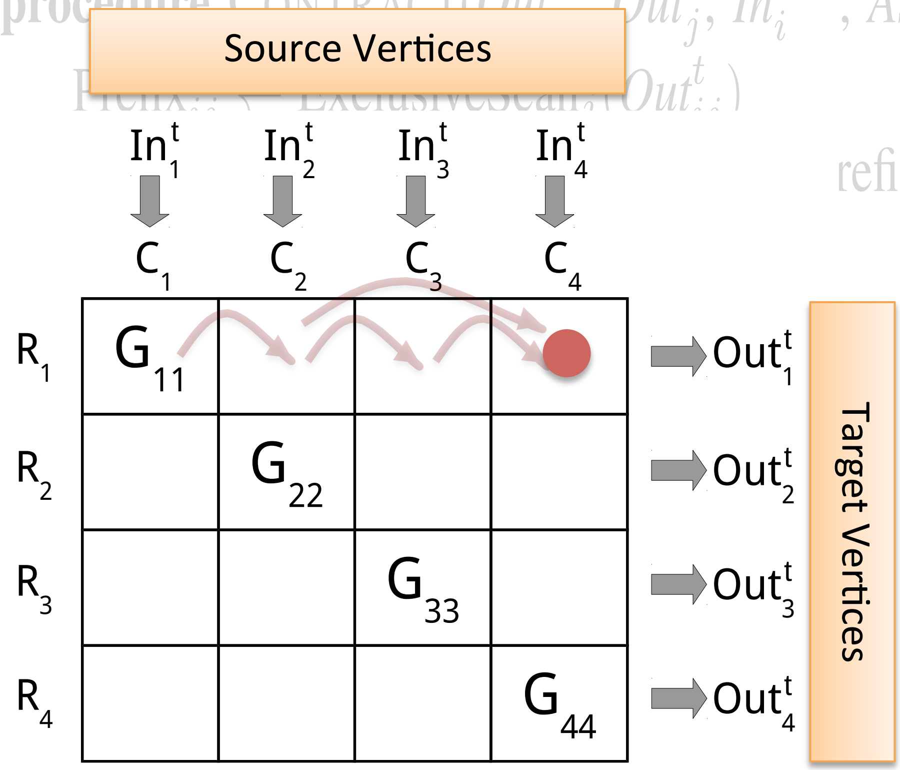
← Broadcast frontier over row.

← Broadcast frontier over column.

end procedure

- We use a parallel scan that minimizes communication steps (vs work).
- This improves the overall scaling efficiency by 30%.

Global Frontier Contraction and Communication



← Distributed prefix sum. Prefix is vertices discovered by your left neighbors (log p)

← Vertices first discovered by this GPU.

← **Right most column has global frontier for the row**

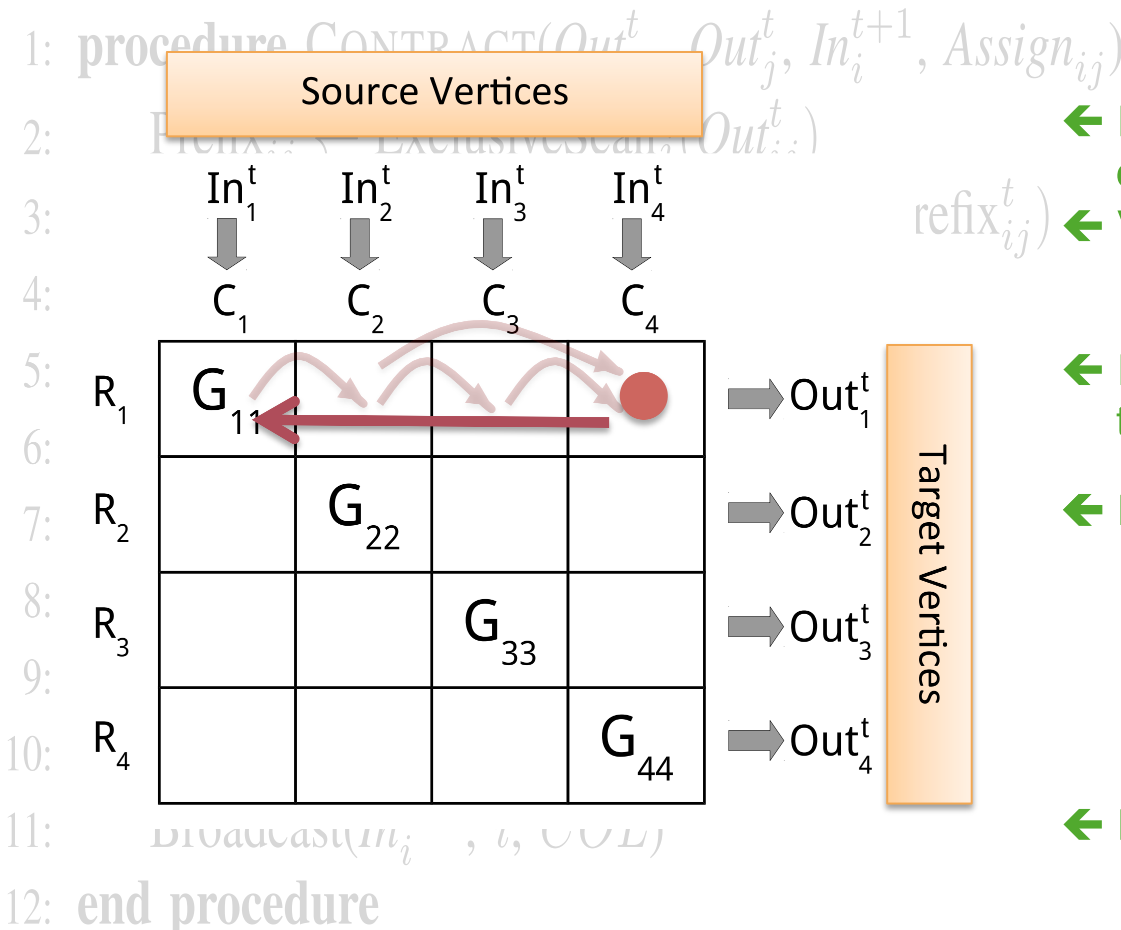
← Broadcast frontier over row.

← Broadcast frontier over column.

end procedure

- We use a parallel scan that minimizes communication steps (vs work).
- This improves the overall scaling efficiency by 30%.

Global Frontier Contraction and Communication



- ← Distributed prefix sum. Prefix is vertices discovered by your left neighbors (log p)
- ← Vertices first discovered by this GPU.
- ← Right most column has global frontier for the row
- ← **Broadcast frontier over row.**
- ← Broadcast frontier over column.

Global Frontier Contraction and Communication

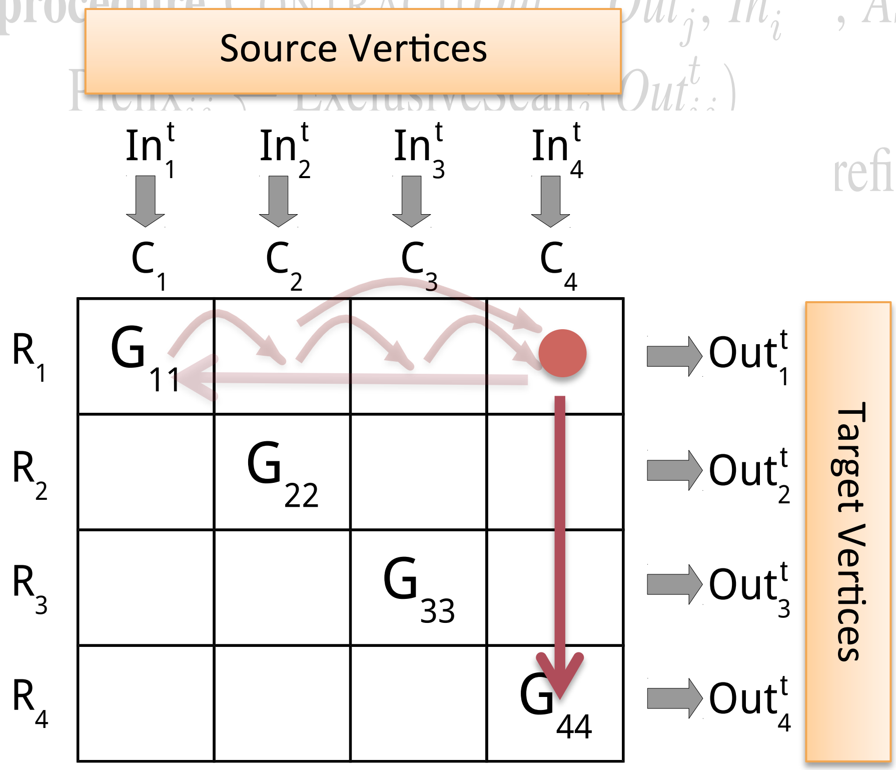
procedure CONTRACT($Out^t_{..}$, Out^t_j, In^{t+1}_i, $Assign_{ij}$)

Prefix ← ExclusiveScan(Out^t_{ij})

...refix$^t_{ij}$)

Broadcast(In_i ..., t, COL)

end procedure



← Distributed prefix sum. Prefix is vertices discovered by your left neighbors (log p)

← Vertices first discovered by this GPU.

← Right most column has global frontier for the row

← Broadcast frontier over row.

← **Broadcast frontier over column.**

- All GPUs now have the new frontier.

Update Levels

procedure UPDATELEVELS(Out_j^t, t, $level$)
  for all $v \in Out_j^t$ in parallel **do**
    $level[v] \leftarrow t$
  end for
end procedure

- We store both the In and Out levels.
- This allows us to compute the predecessors in a GPU local manner.

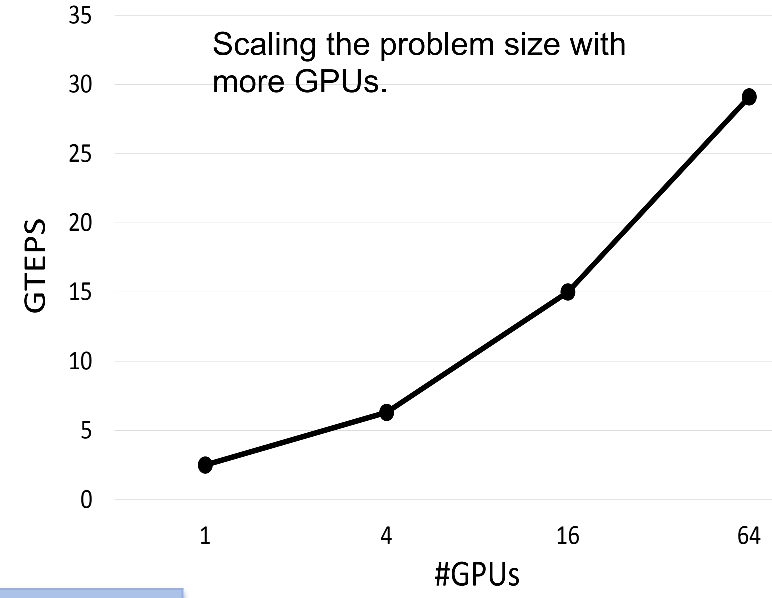
Predecessor Computation

procedure UPDATEPREDS($Assigned_{ij}$, $Preds_{ij}$, $level$)
  for all $v \in Assigned_{ij}$ in parallel **do**
    $Pred[v] \leftarrow -1$
    for $i \leftarrow \text{ColOff}[v], \text{ColOff}[v+1]$ **do**
      if $level[v] == level[\text{RowIdx}[i]] + 1$ **then**
        $Pred[v] \leftarrow \text{RowIdx}[i]$
      end if
    end for
  end for
end procedure

- Predecessors are computed after the traversal is complete using node-local In/Out levels.
- No inter-node communication is required.

Weak Scaling

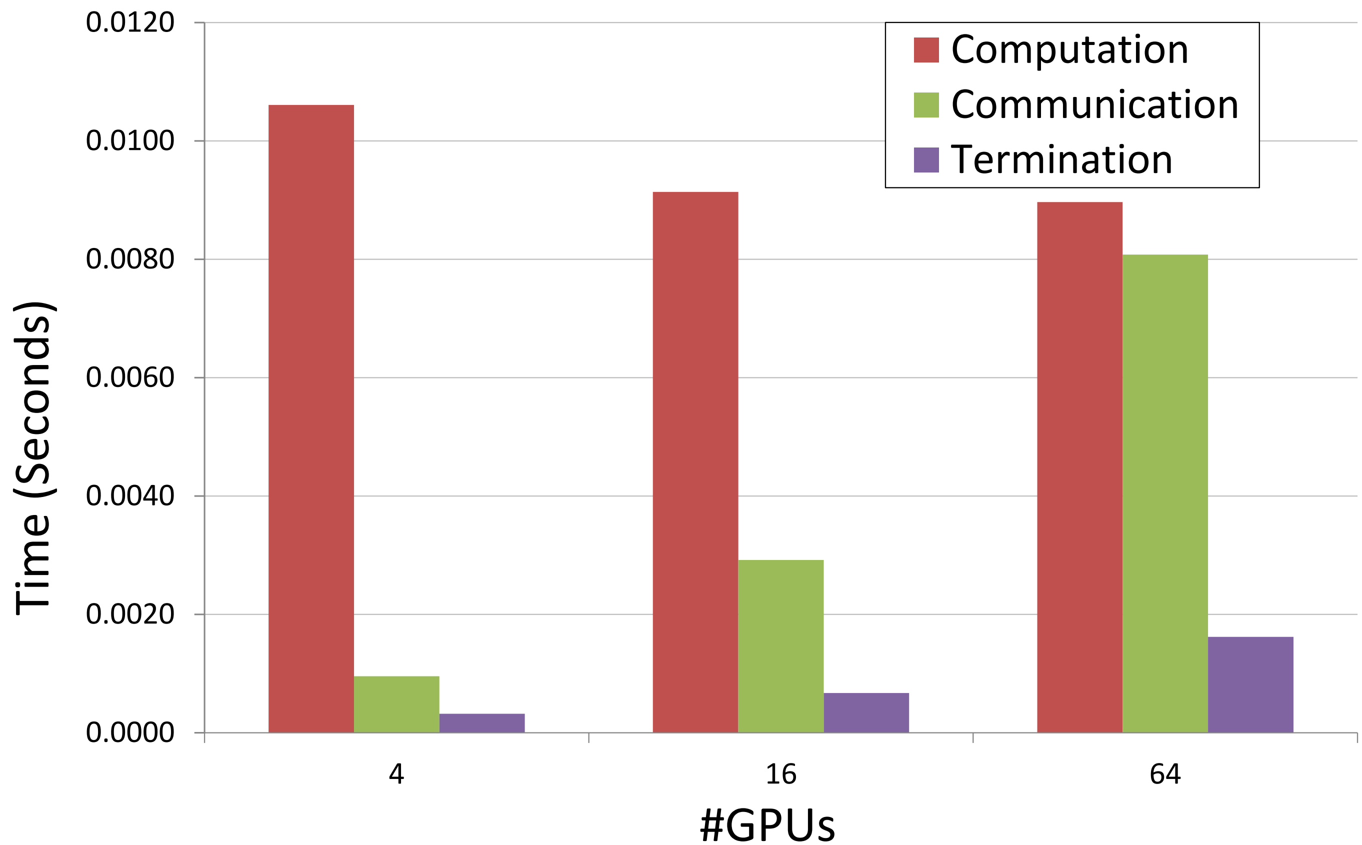
- Scaling the problem size with more GPUs
  - Fixed problem size per GPU.
- Maximum scale 27 (4.3B edges)
  - 64 K20 GPUs => .147s => 29 GTEPS
  - 64 K40 GPUs => .135s => 32 GTEPS
    - K40 has faster memory bus.



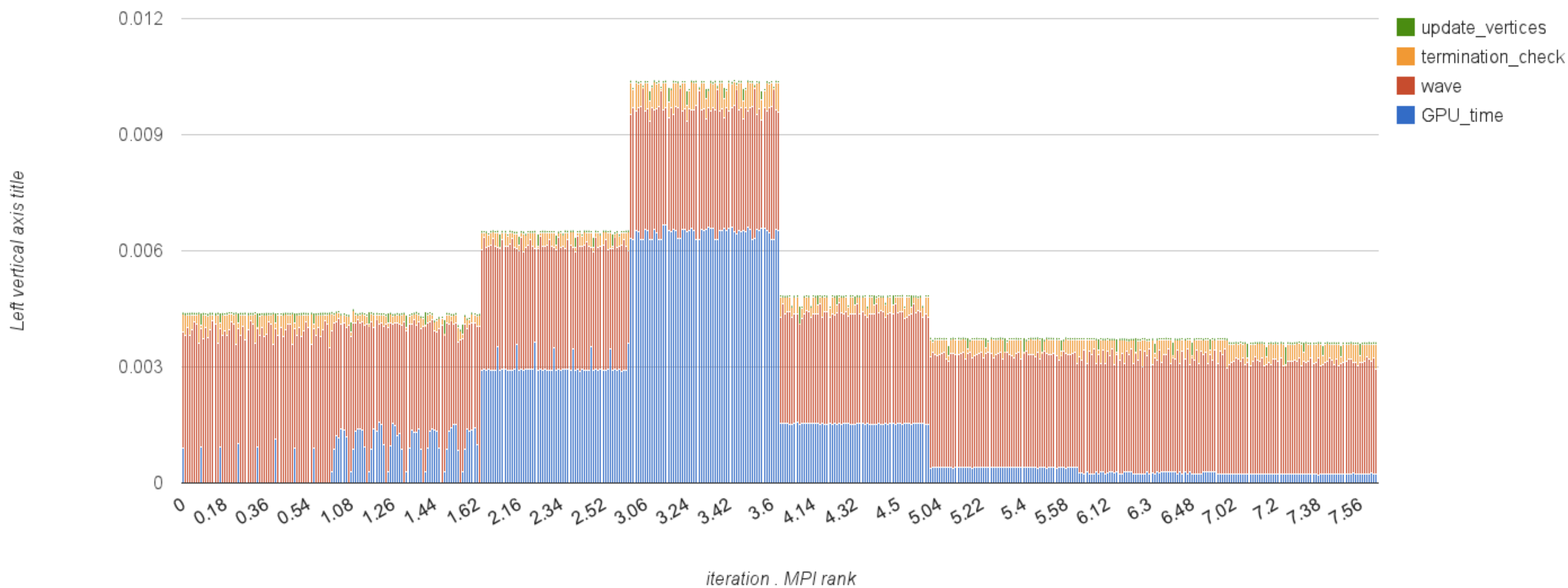
GPUs	Scale	Vertices	Edges	Time (s)	GTEPS
1	21	2,097,152	67,108,864	**0.0254**	2.5
4	23	8,388,608	268,435,456	**0.0429**	6.3
16	25	33,554,432	1,073,741,824	**0.0715**	15.0
64	27	134,217,728	4,294,967,296	**0.1478**	29.1

Central Iteration Costs (Weak Scaling)

- Communication costs are not constant.
- 2D design implies cost grows as
  2 log(2p)/log(p)
- How to scale?
  - Overlapping
  - Compression
    - Graph
    - Message
  - Hybrid partitioning
  - Heterogeneous computing



Costs During BFS Traversal

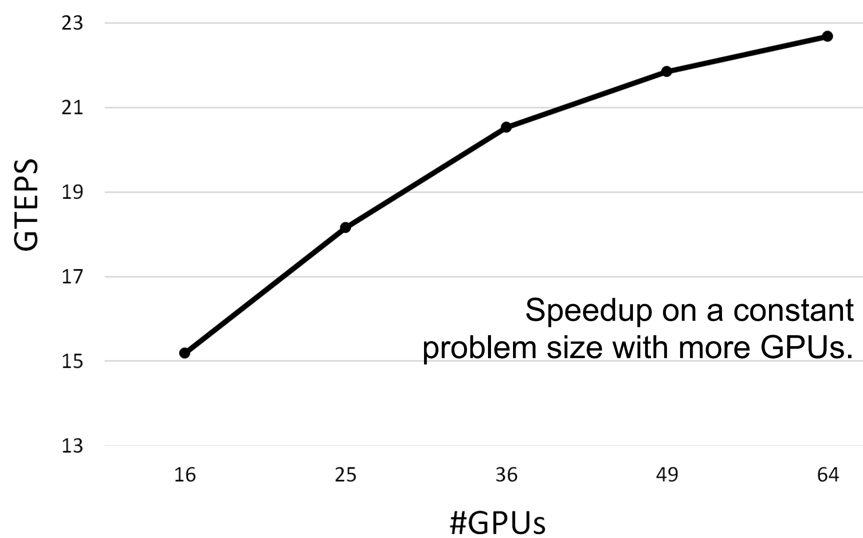


- Chart shows the different costs for each GPU in each iteration (64 GPUs).
- Wave time is essentially constant, as expected.
- Compute time peaks during the central iterations.
- Costs are reasonably well balanced across all GPUs after the 2nd iteration.

Strong Scaling

- Speedup on a constant problem size with more GPUs
- Problem scale 25
  - 2^25 vertices (33,554,432)
  - 2^26 directed edges (1,073,741,824)
- Strong scaling efficiency of 48%
  - Versus 44% for BG/Q

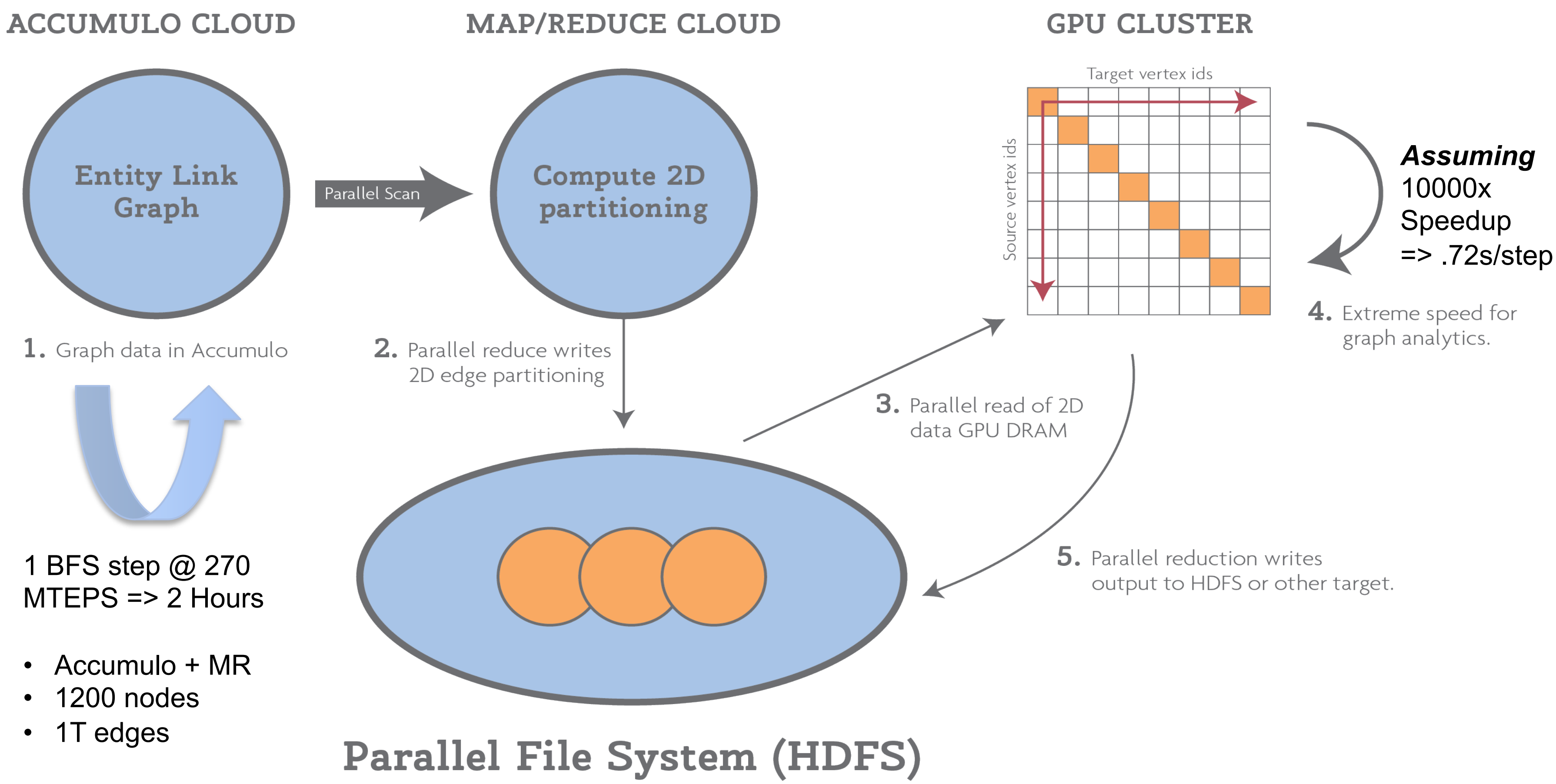
GPUs	GTEPS	Time (s)
16	15.2	**0.071**
25	18.2	**0.059**
36	20.5	**0.053**
49	21.8	**0.049**
64	22.7	**0.047**



Directions to Improve Scaling

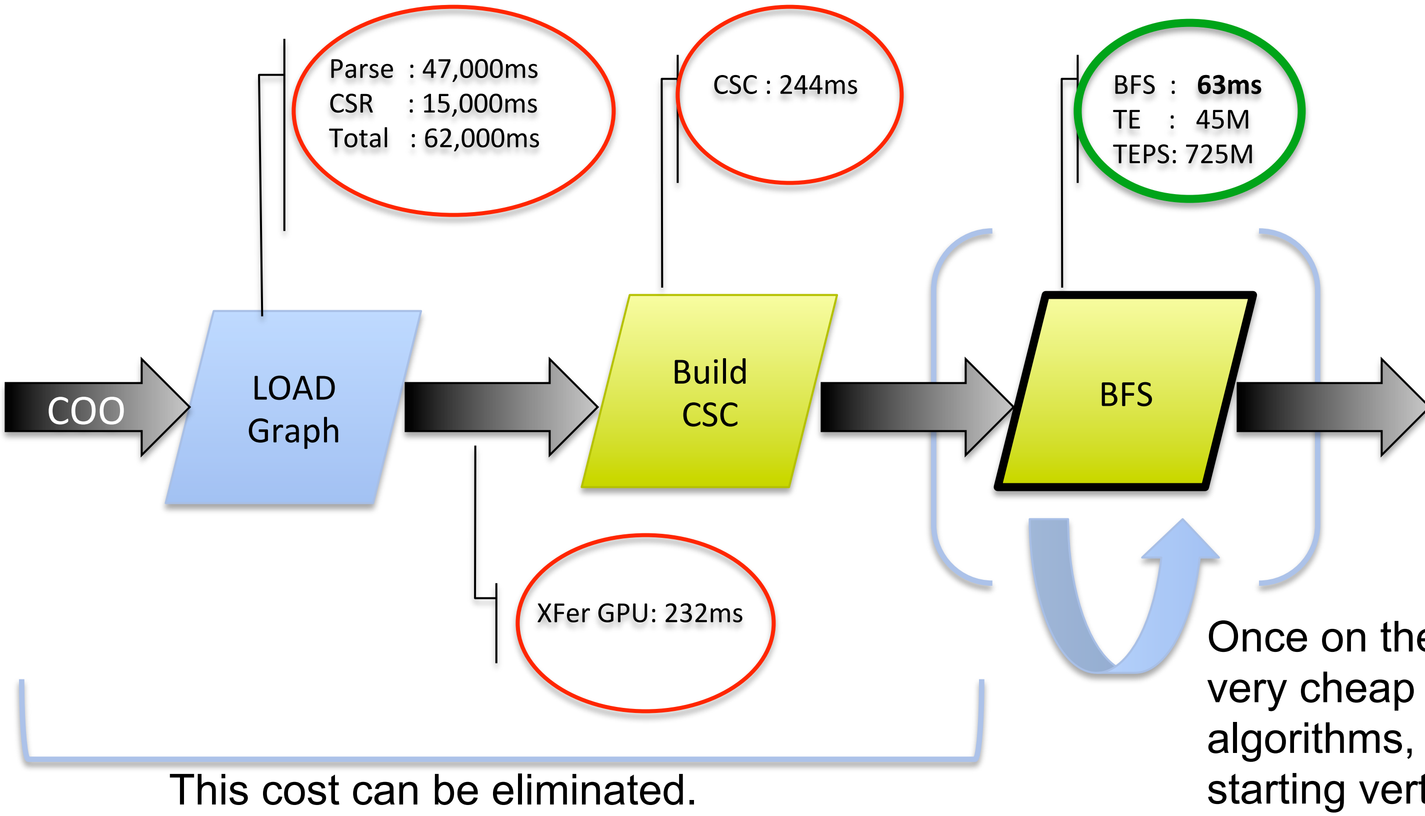
- Overlap computation with communication
  - Multiple partitions per GPU
  - Frontier compression
- Hybrid partitioning
  - Degree aware data layout + bottom up search optimization
    - This also requires asynchronous communications and per-target node buffers.
  - Graph aware partitioning plus 2D data layout
- Uintah style data warehouse
  - Hand off tasks to workers (Uintah)
  - Hybrid CPU/GPU computation strategies (TOTEM)

Concept: Accelerate Key Value Stores



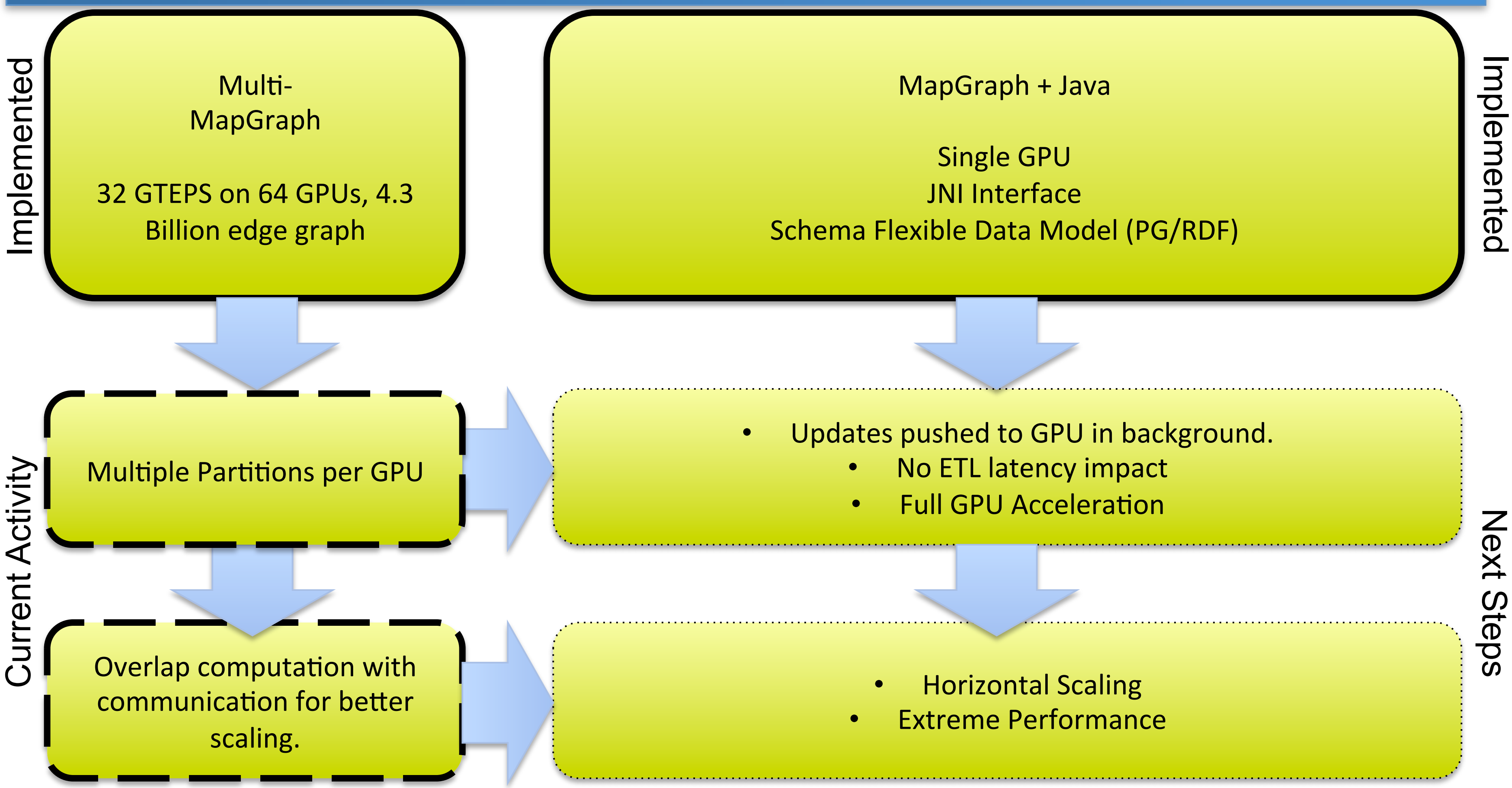
MapGraph Timings (single GPU)

- Orkut social network 2.3M vertices, 92M edges. Most time is *load* on CPU.
- Next step eliminates overhead: 62500ms => 63ms (1000x faster)



Once on the GPU, it is very cheap to run other algorithms, different starting vertices, etc.

Current and Future Code Streams



MapGraph Beta Customer?

Contact
Bradley Bebee
beebs@systap.com