



PENNSSTATE



# Parallel breadth-first search on distributed memory systems

Aydın Buluç

Lawrence Berkeley National Laboratory

Kamesh Madduri

The Pennsylvania State University

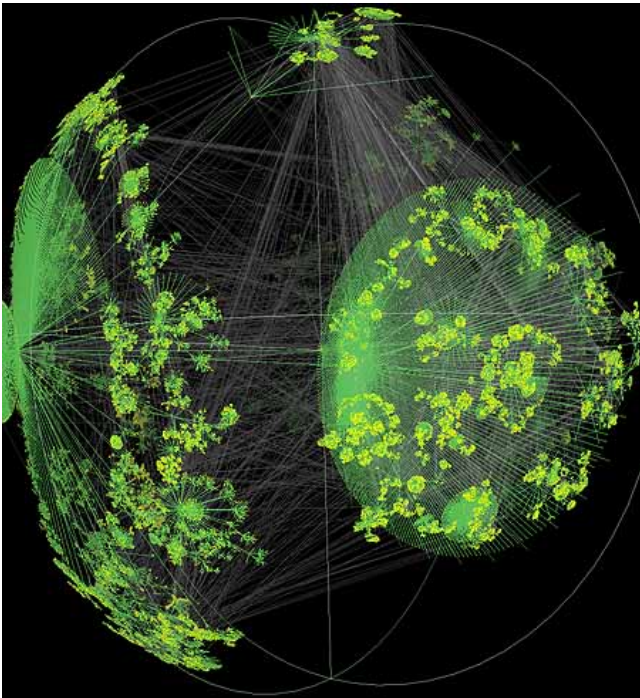
Supercomputing, Seattle

November 17, 2011

# Large graphs are everywhere

Internet structure  
Social interactions

Scientific datasets: biological,  
chemical, cosmological, ecological, ...



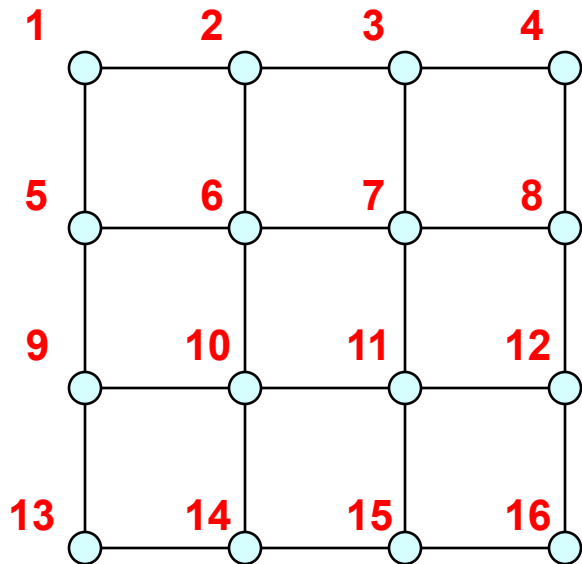
WWW snapshot, courtesy Y. Hyun



Yeast protein interaction network, courtesy H. Jeong

# Breadth-first search (BFS)

- Level-by-level graph traversal
- Serial complexity:  $\Theta(m+n)$



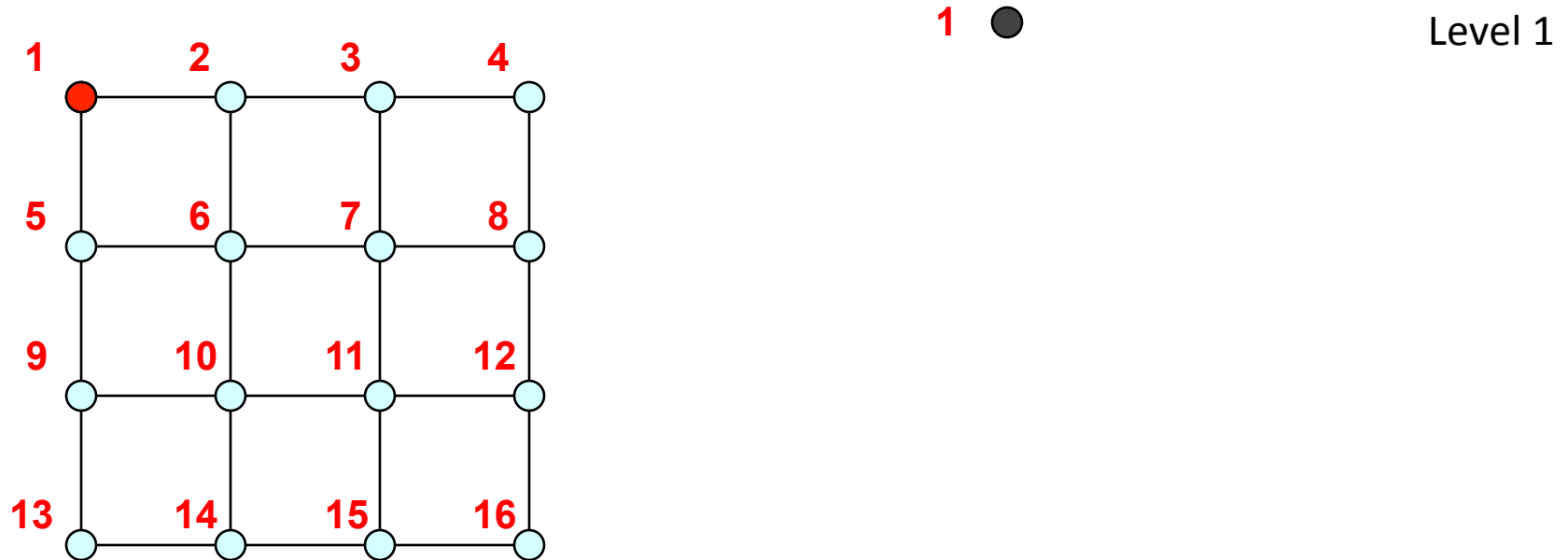
Graph:  **$G(E, V)$**

**$E$** : Set of edges (size  $m$ )

**$V$** : Set of vertices (size  $n$ )

# Breadth-first search (BFS)

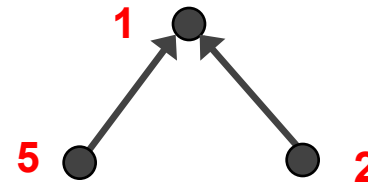
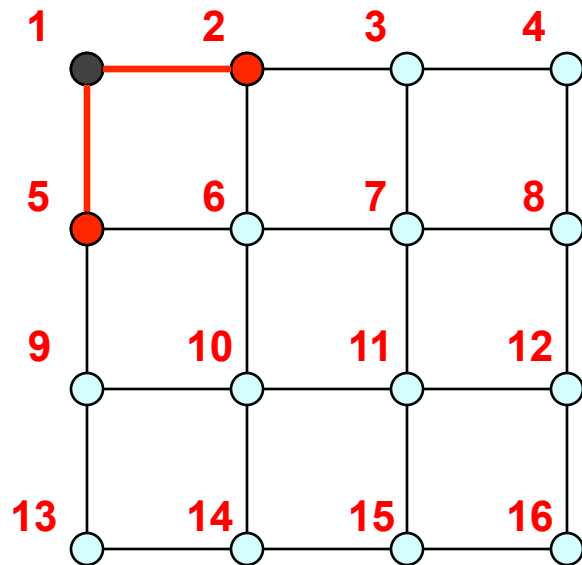
- Level-by-level graph traversal
- Serial complexity:  $\Theta(m+n)$



Current 'frontier' shown in red

# Breadth-first search (BFS)

- Level-by-level graph traversal
- Serial complexity:  $\Theta(m+n)$



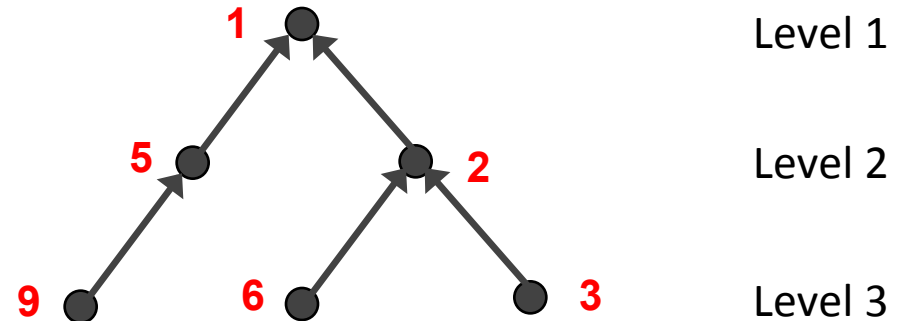
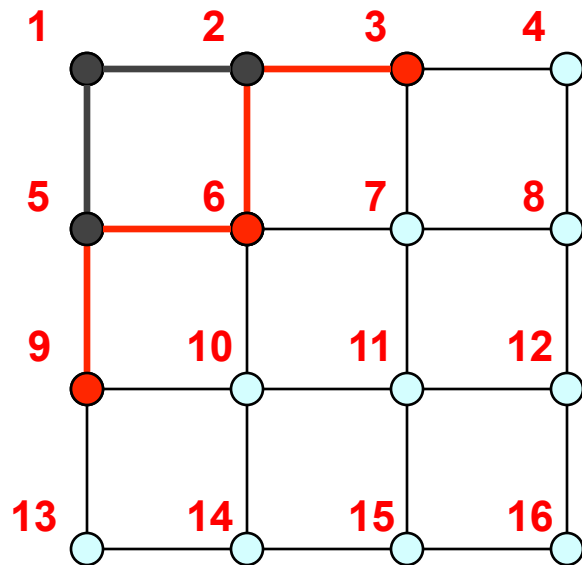
Level 1

Level 2

Current 'frontier' shown in red

# Breadth-first search (BFS)

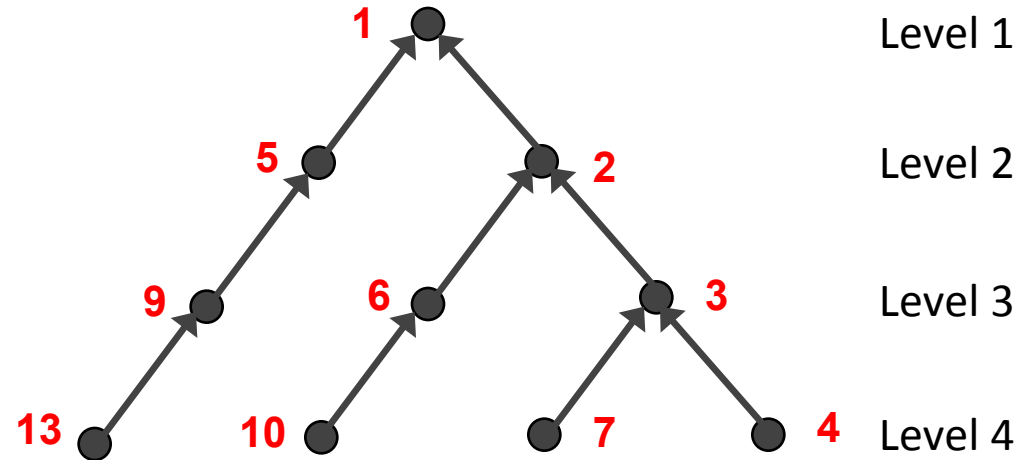
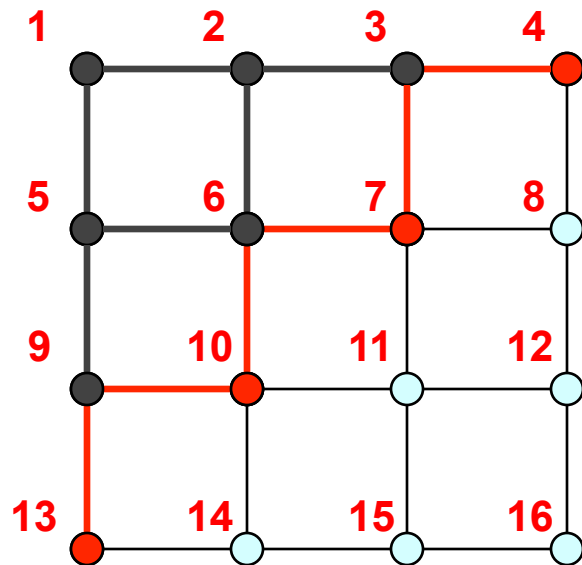
- Level-by-level graph traversal
- Serial complexity:  $\Theta(m+n)$



Current 'frontier' shown in red

# Breadth-first search (BFS)

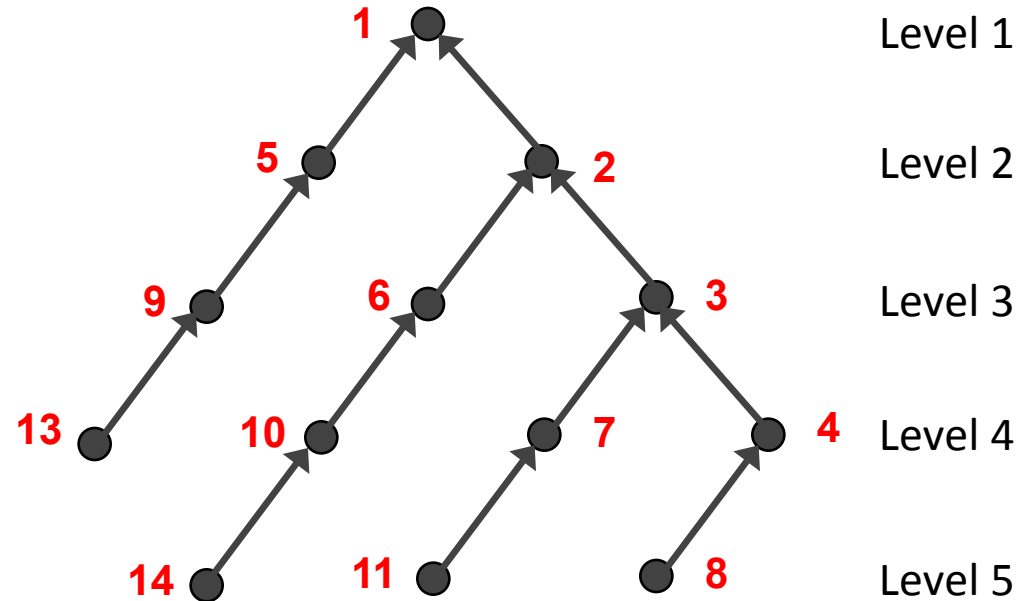
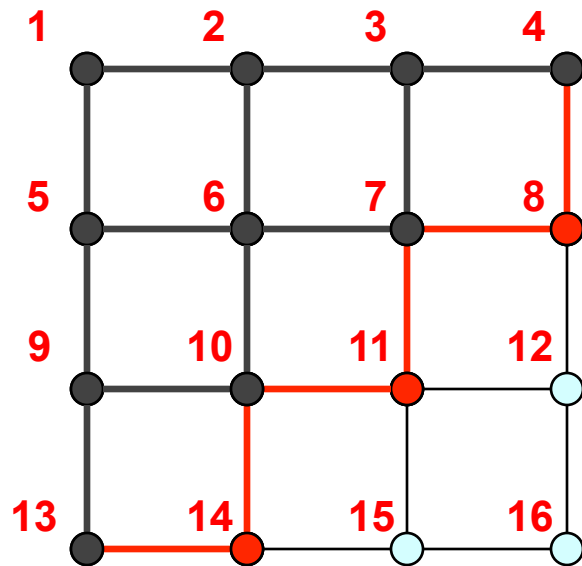
- Level-by-level graph traversal
- Serial complexity:  $\Theta(m+n)$



Current 'frontier' shown in red

# Breadth-first search (BFS)

- Level-by-level graph traversal
- Serial complexity:  $\Theta(m+n)$



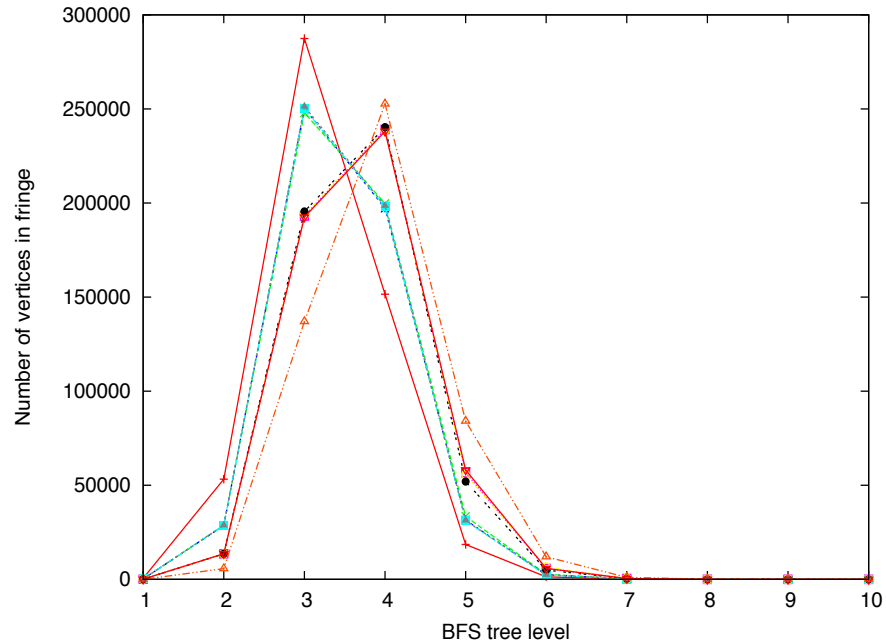
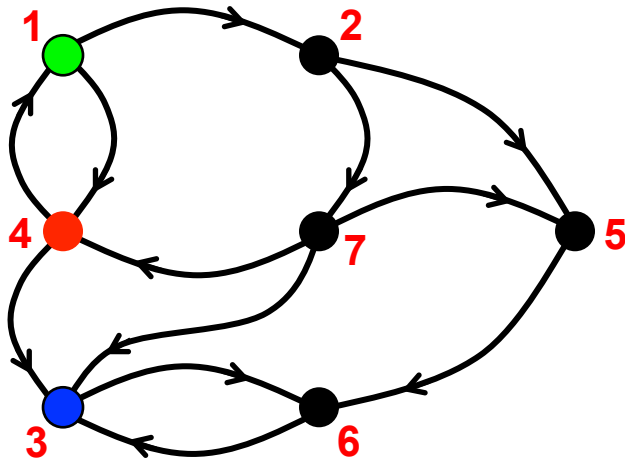
Current 'frontier' shown in red



# BFS as a graph building block

- BFS is representative of communication intensive graph computations in distributed memory
- BFS is a subroutine for many algorithms
  - Betweenness centrality
  - Maximum flows
  - Connected components
  - Spanning forests
  - Testing for bipartiteness
  - Reverse Cuthill-McKee ordering

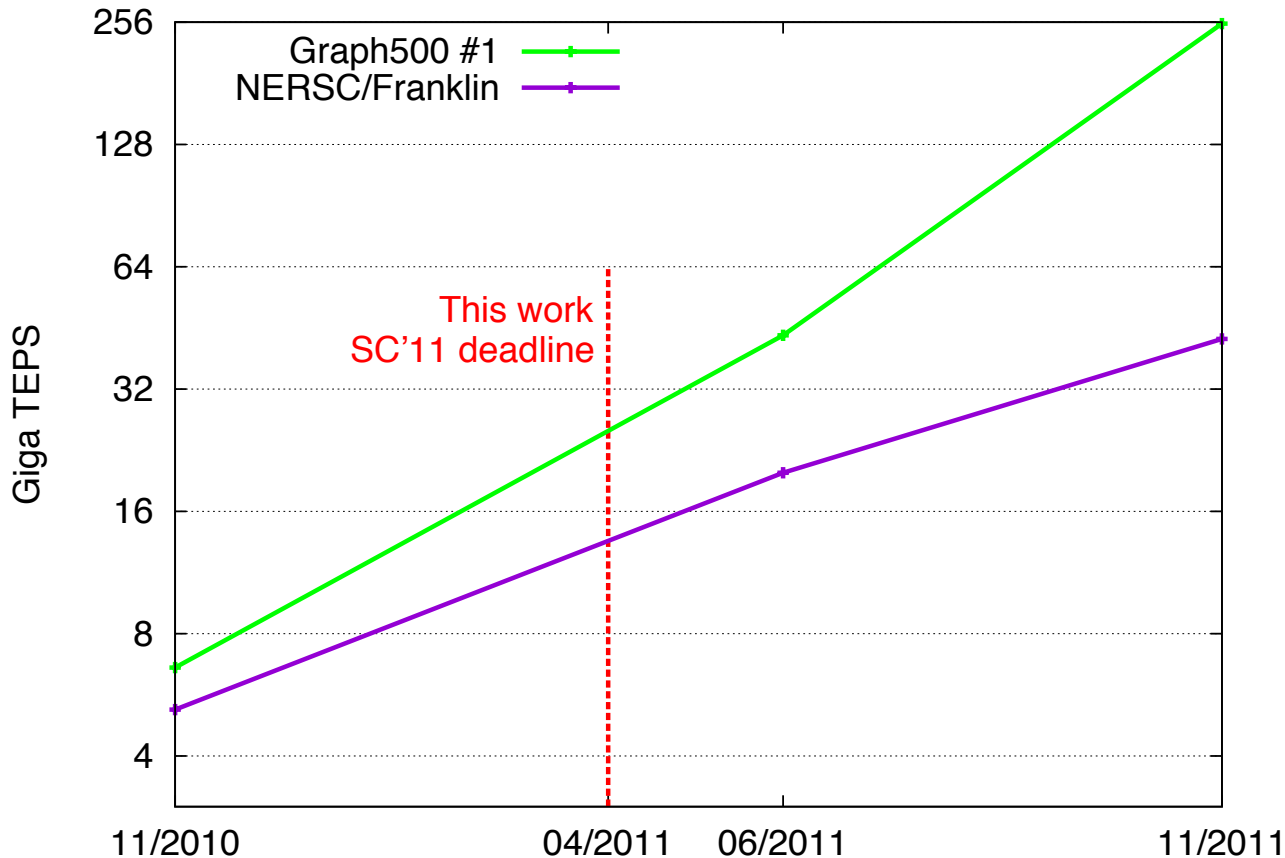
Breadth-first search  
on a low-diameter graph  
with skewed degree  
distribution



- Short span (critical path)
- High parallelism (work/span)

Performance metric: TEPS  
“Traversed Edges Per Second”

# Graph 500: implementation challenge



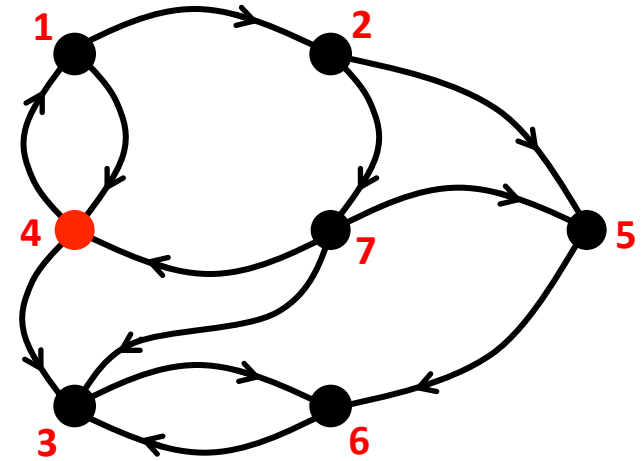
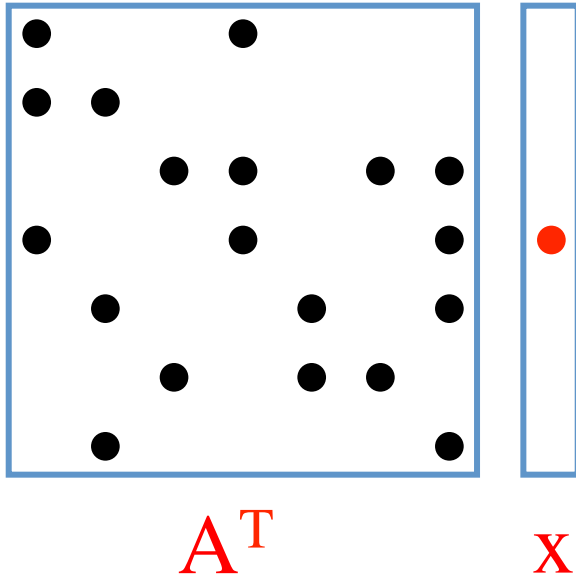
8.2X improvement  
on same hardware,  
in one year.

Performance numbers are just a snapshot in time  
But the insights and conclusions are still valid today

# Outline

- BFS overview and applications
- **BFS as sparse matrix-sparse vector multiply**
- Parallel BFS: 1D and 2D approaches
- Experimental results and insight
- Conclusions / Contributions
- Future Directions

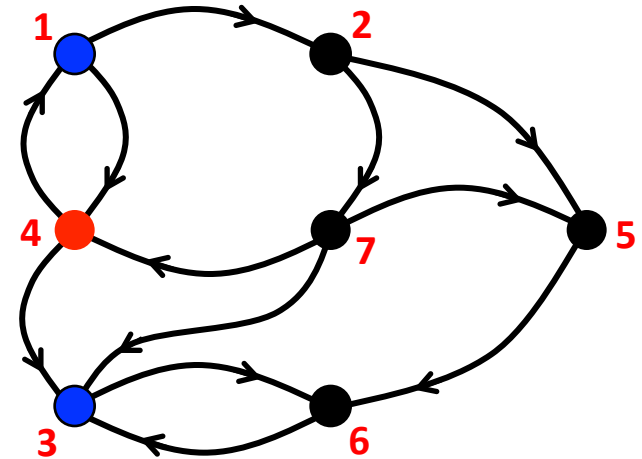
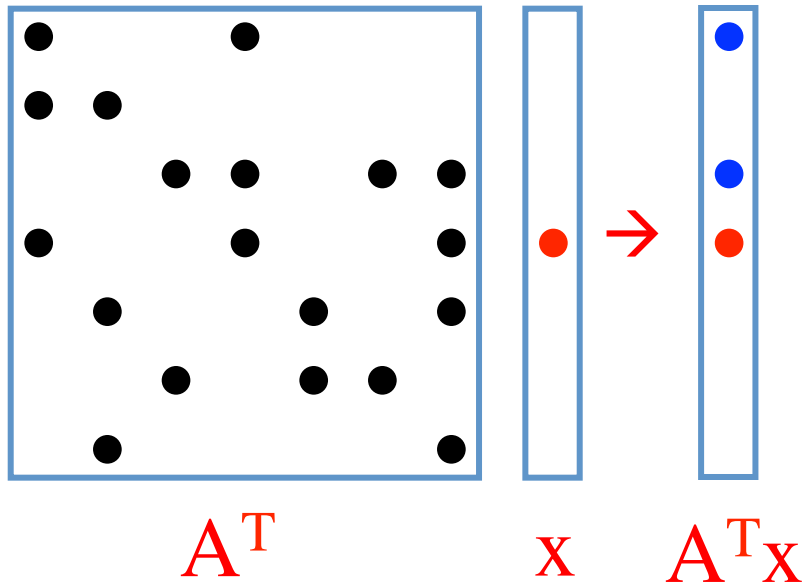
# Breadth-first search: a matrix view



[loops are not drawn]

- Adjacency matrix: sparse array w/ nonzeros for graph edges
- Multiply by adjacency matrix  $\rightarrow$  step to neighbor vertices

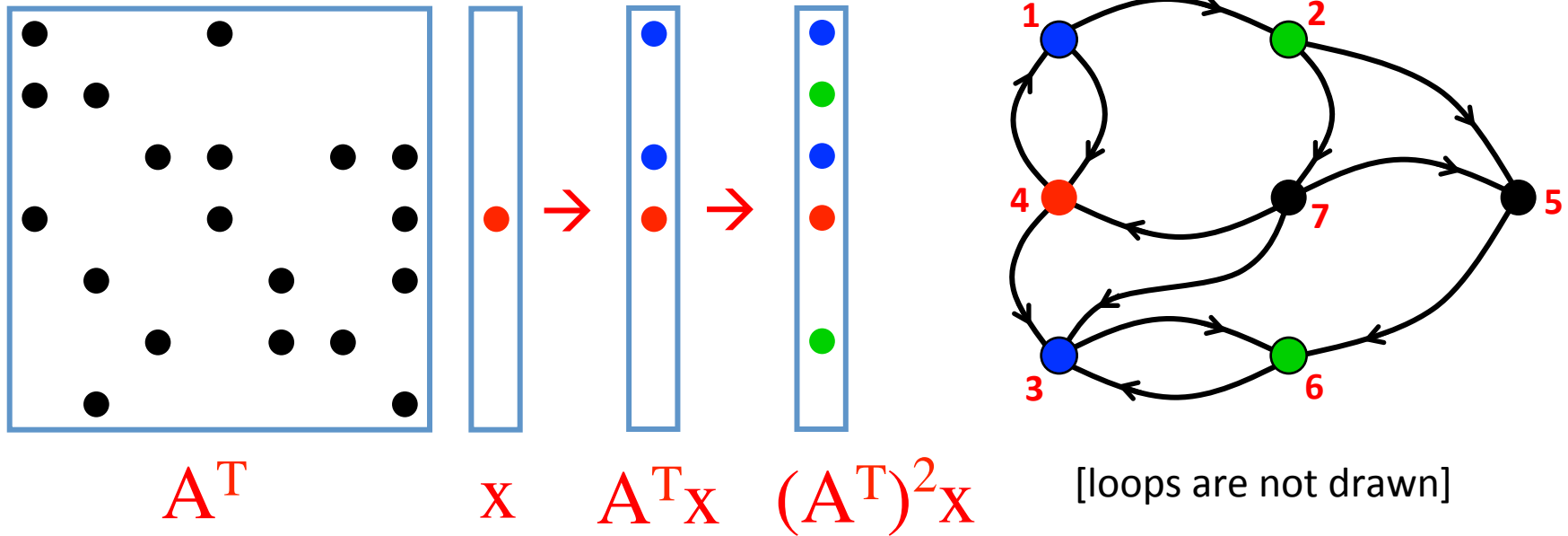
# Breadth-first search: a matrix view



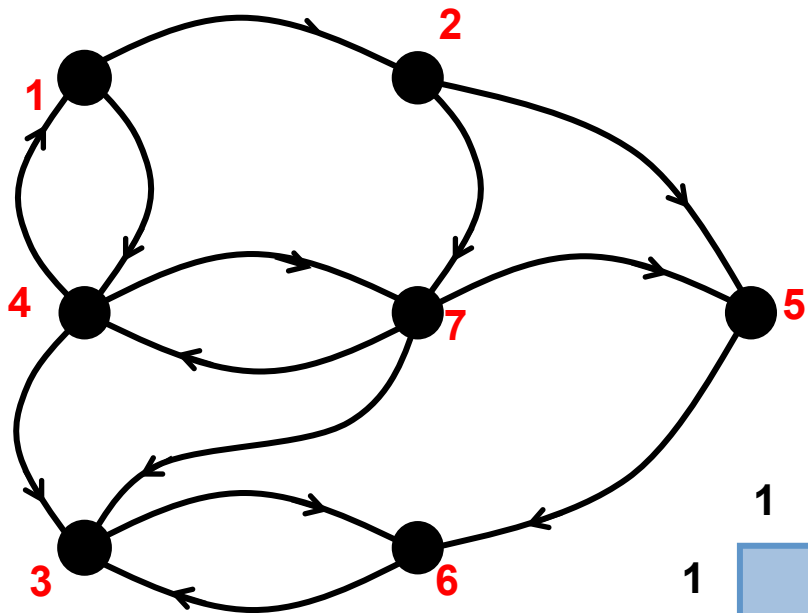
[loops are not drawn]

- Adjacency matrix: sparse array w/ nonzeros for graph edges
- Multiply by adjacency matrix  $\rightarrow$  step to neighbor vertices

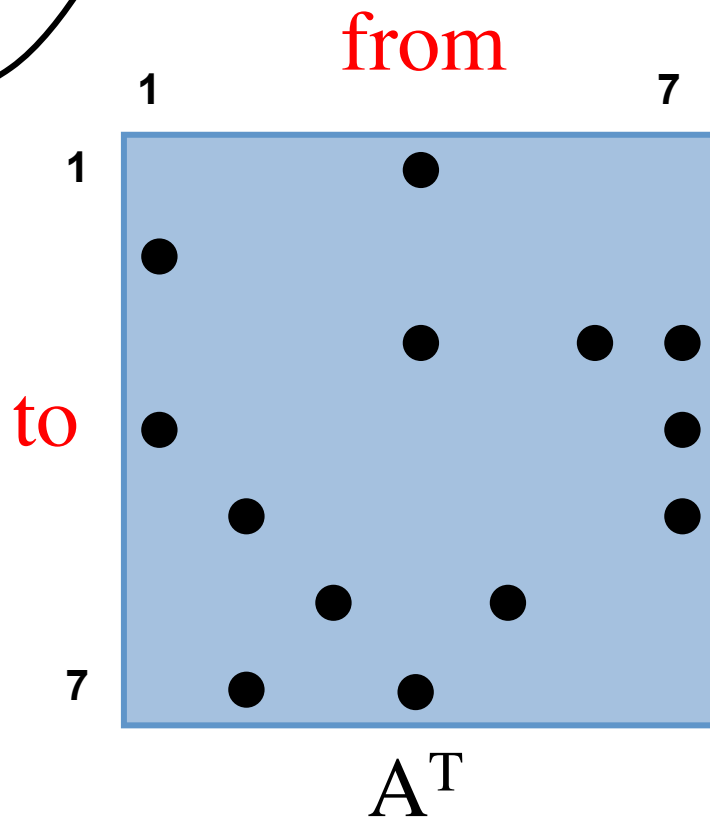
# Breadth-first search: a matrix view

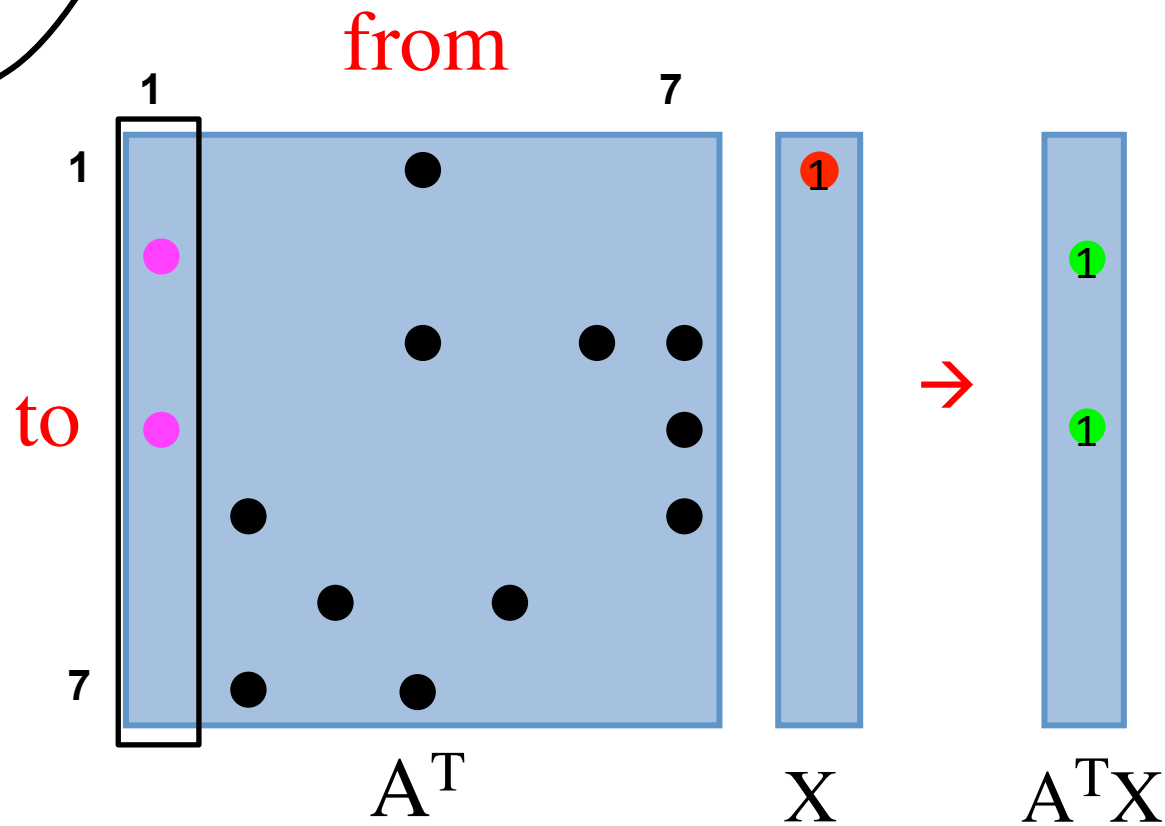
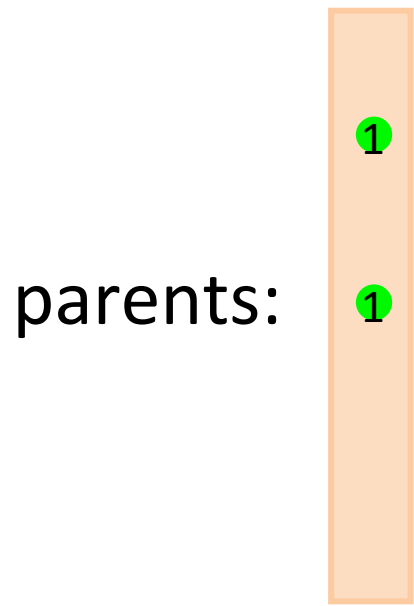
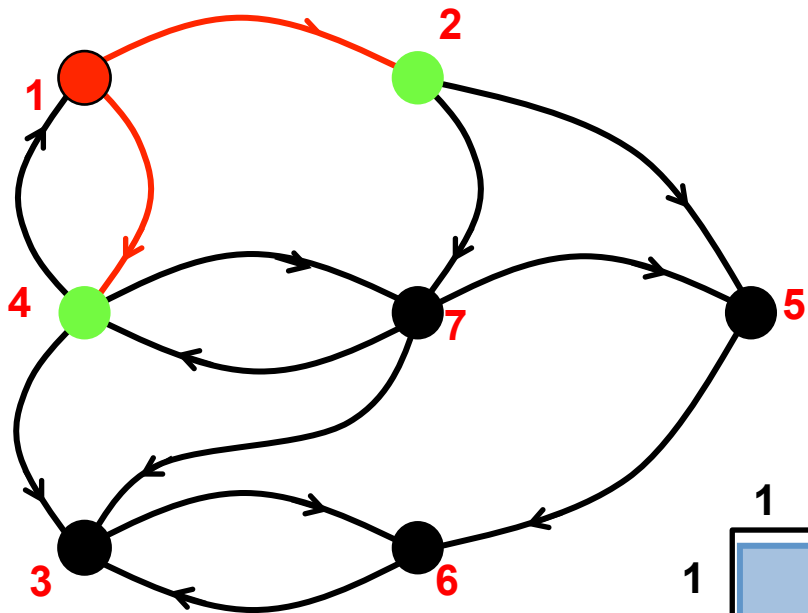


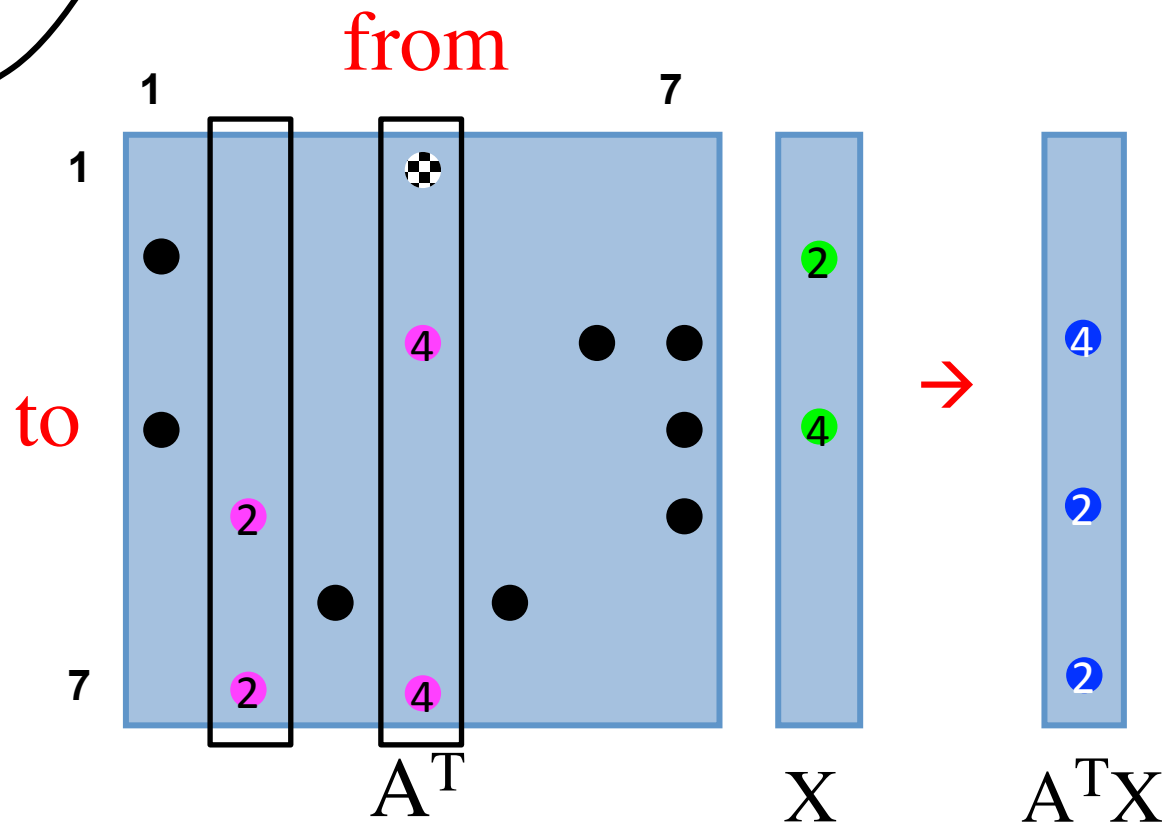
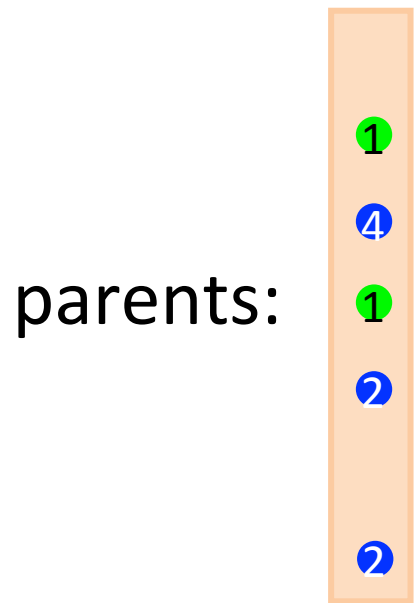
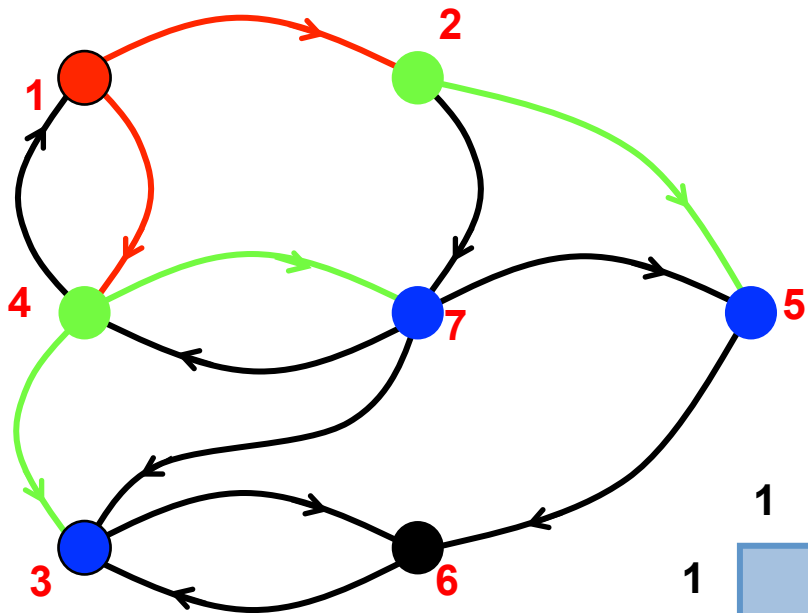
- Adjacency matrix: sparse array w/ nonzeros for graph edges
- Multiply by adjacency matrix  $\rightarrow$  step to neighbor vertices

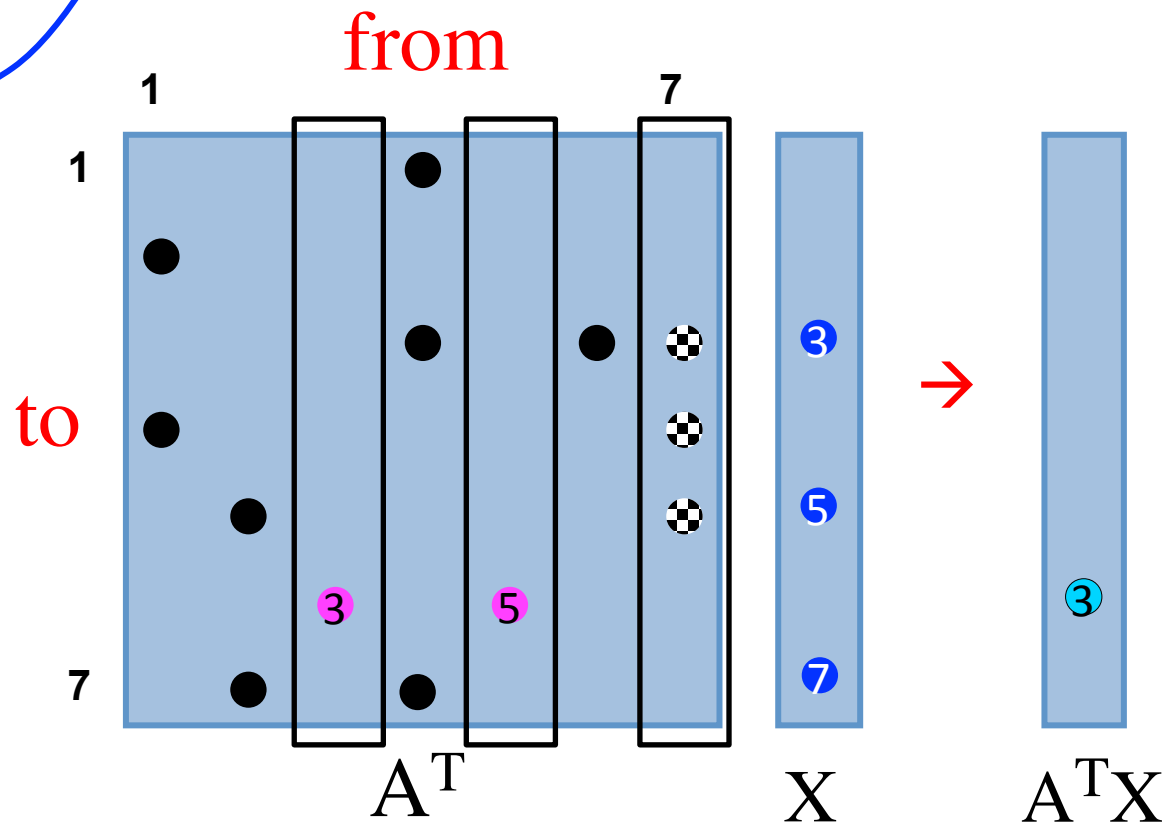
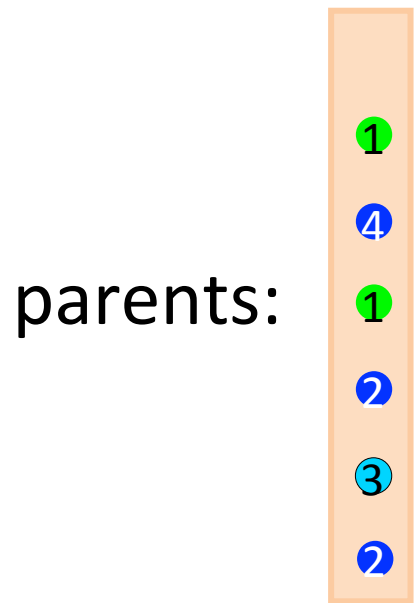
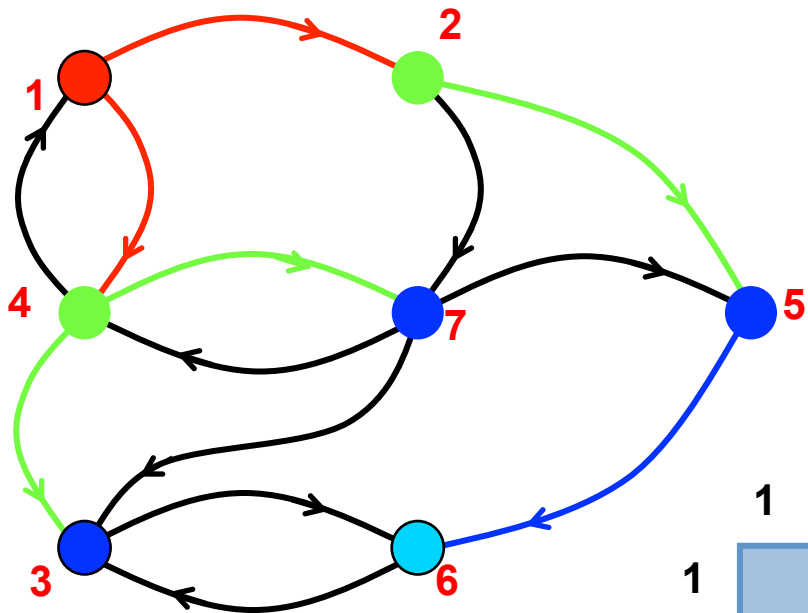


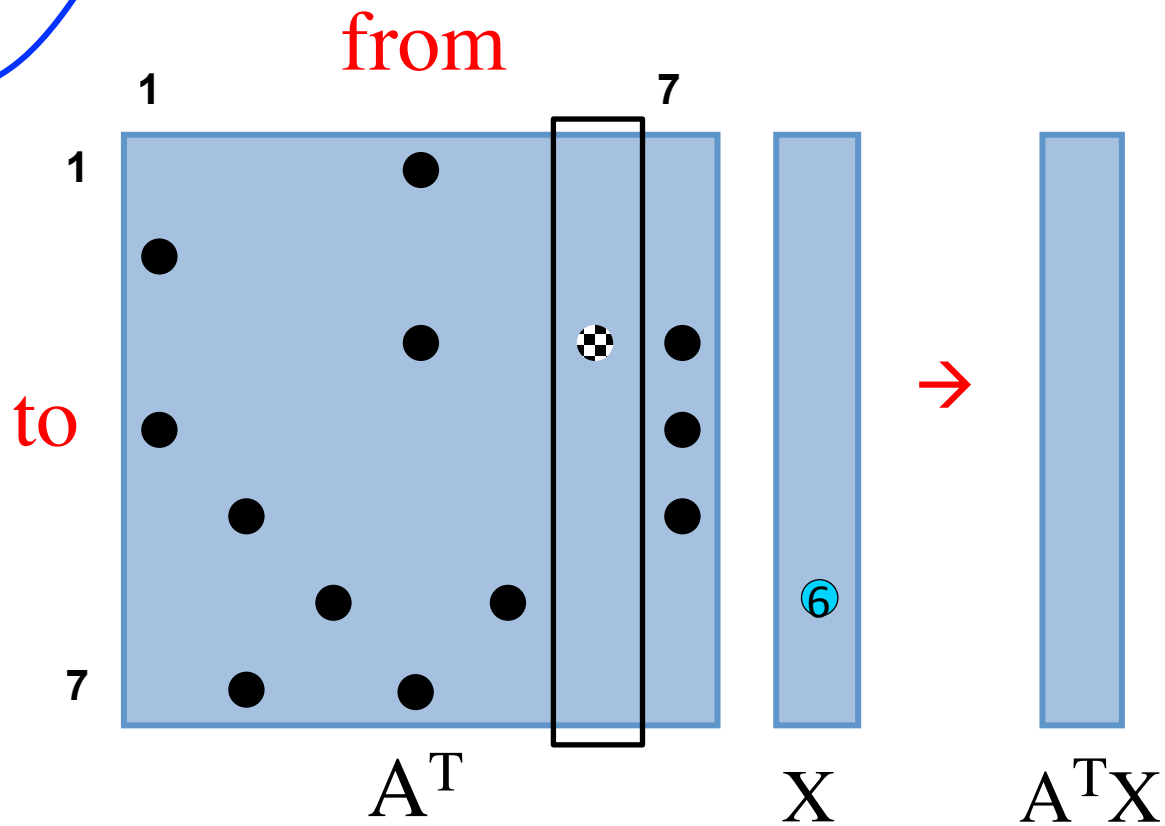
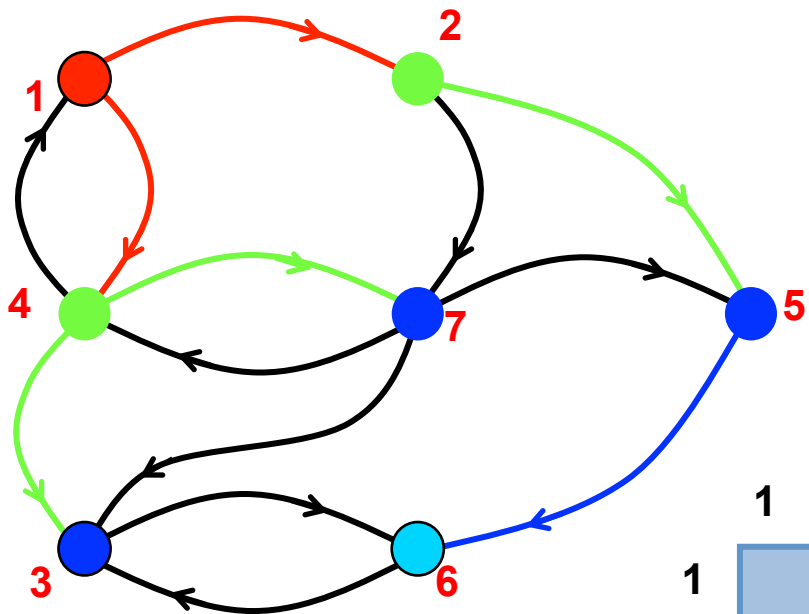
BFS as sparse matrix-  
 sparse vector multiply  
 Semiring: (select, min)









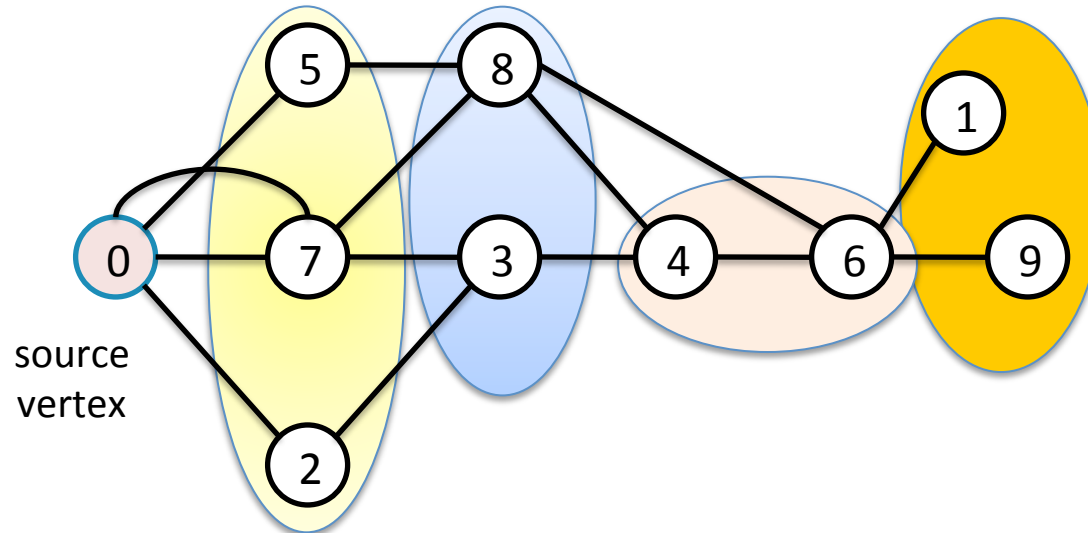


# Outline

- BFS overview and applications
- BFS as sparse matrix-sparse vector multiply
- **Parallel BFS: 1D and 2D approaches.**
- Experimental results and insight
- Conclusions / Contributions
- Future Directions

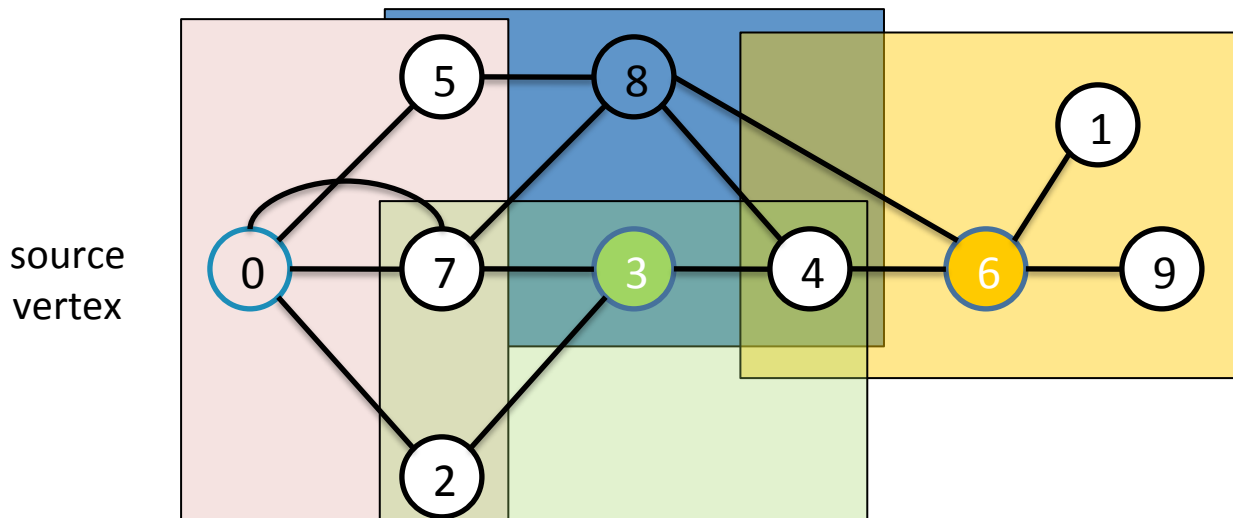
# Parallel BFS strategies

1. Expand current frontier (**level-synchronous** approach, suited for **low diameter** graphs)



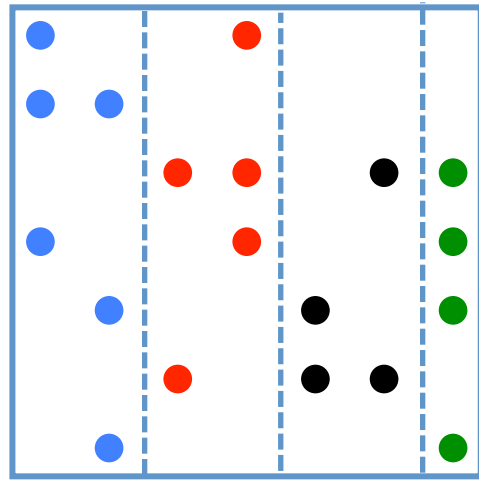
- $O(D)$  parallel steps
- Adjacencies of all vertices in current frontier are visited in parallel

2. Stitch multiple concurrent traversals (Ullman-Yannakakis, for **high-diameter** graphs)

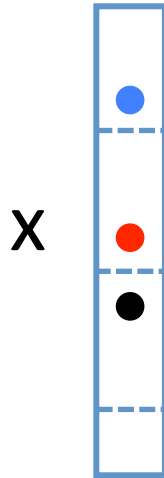


- path-limited searches from “super vertices”
- APSP between “super vertices”

# 1D parallel BFS algorithm

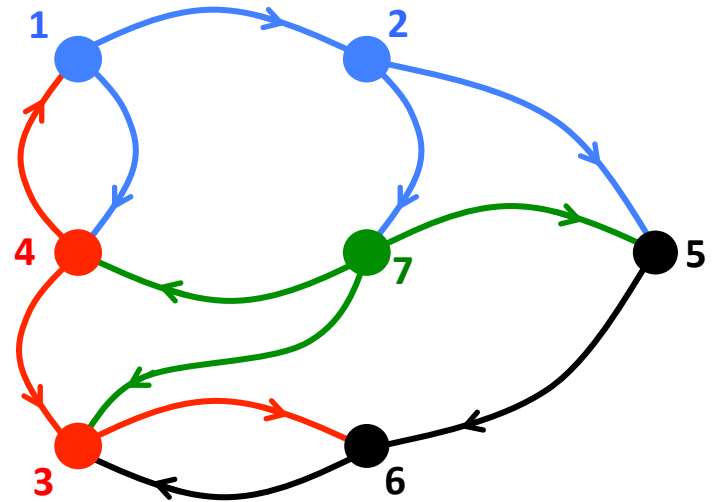


$A^T$



X

frontier



## ALGORITHM:

1. Find owners of the current frontier's adjacency [computation]
2. Exchange adjacencies via all-to-all. **[communication]**
3. Update distances/parents for unvisited vertices. [computation]

# Communication in 1D algorithm

**Local memory references:**

$$\beta_L \frac{m}{p} + \alpha_{L,n/p} \frac{n+m}{p}$$

Inverse local  
RAM bandwidth

Local latency on  
working set  $|n/p|$

**Remote communication:**

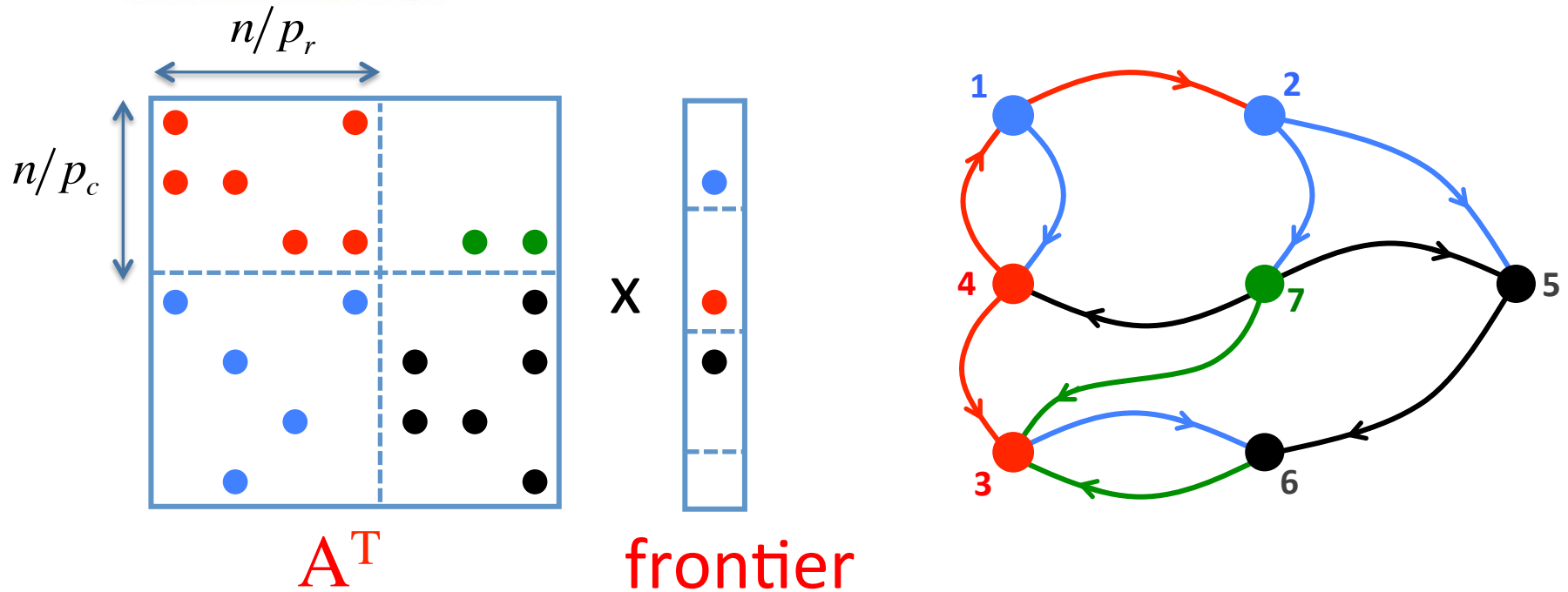
$$\beta_{N,a2a}(p) \frac{m}{p} + \alpha_N p$$

All-to-all remote bandwidth  
with  $p$  participating processors

**ALGORITHM:**

1. Find owners of the current frontier's adjacency [computation]
2. Exchange adjacencies via all-to-all. **[communication]**
3. Update distances/parents for unvisited vertices. [computation]

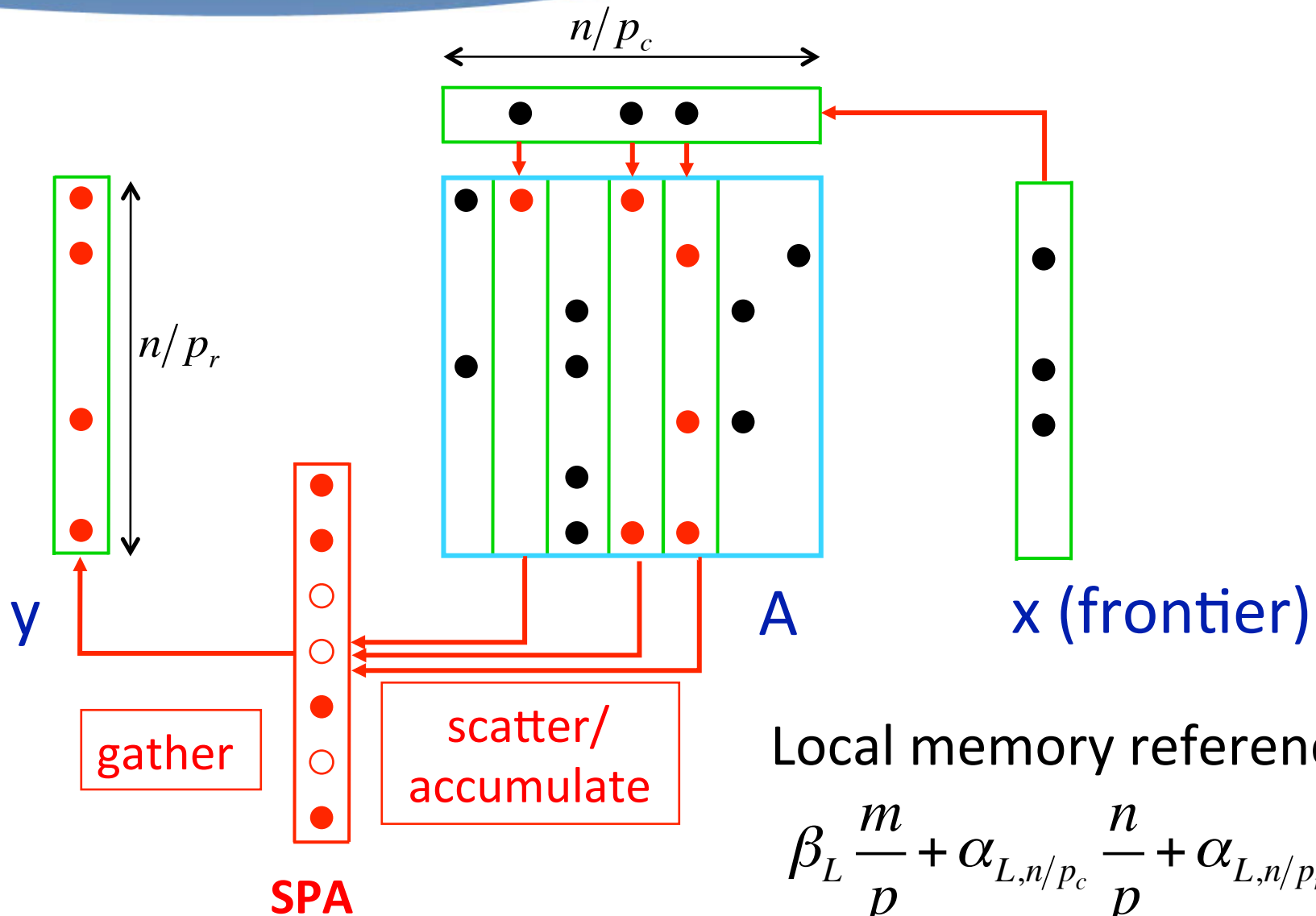
# 2D parallel BFS algorithm



## ALGORITHM:

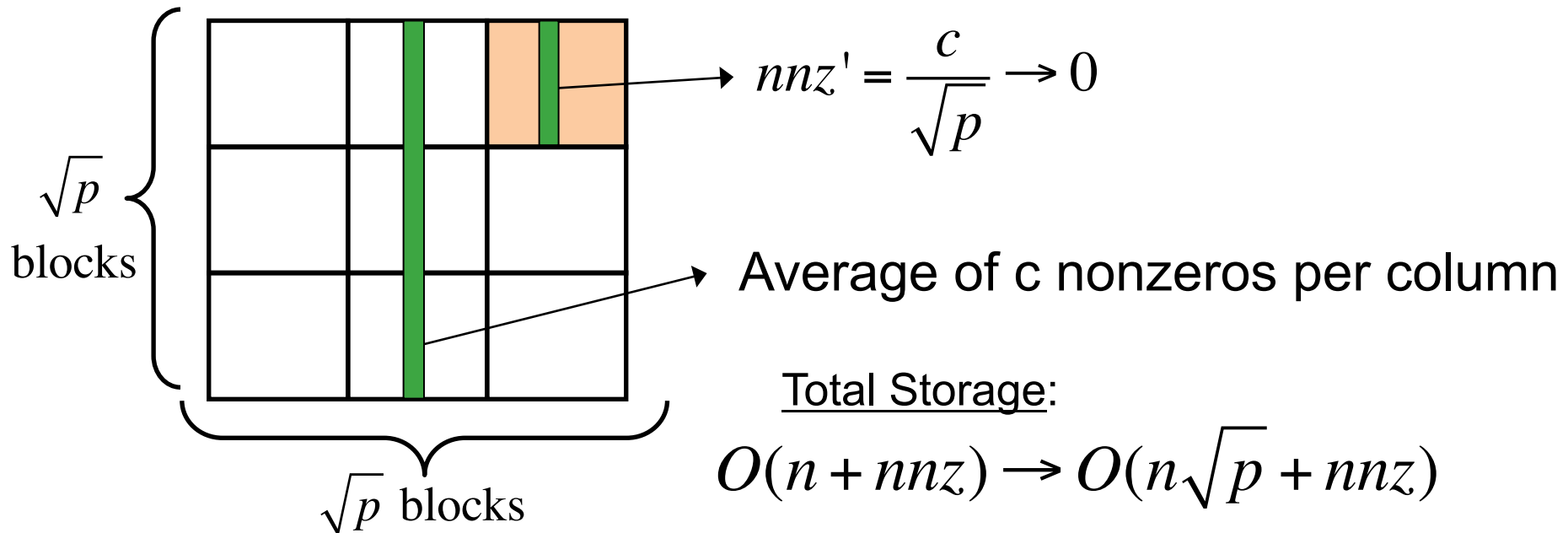
1. Gather vertices in *processor column* [**communication**]
2. Find owners of the current frontier's adjacency [computation]
3. Exchange adjacencies in *processor row* [**communication**]
4. Update distances/parents for unvisited vertices. [computation]

# 2D algorithm: Local computation



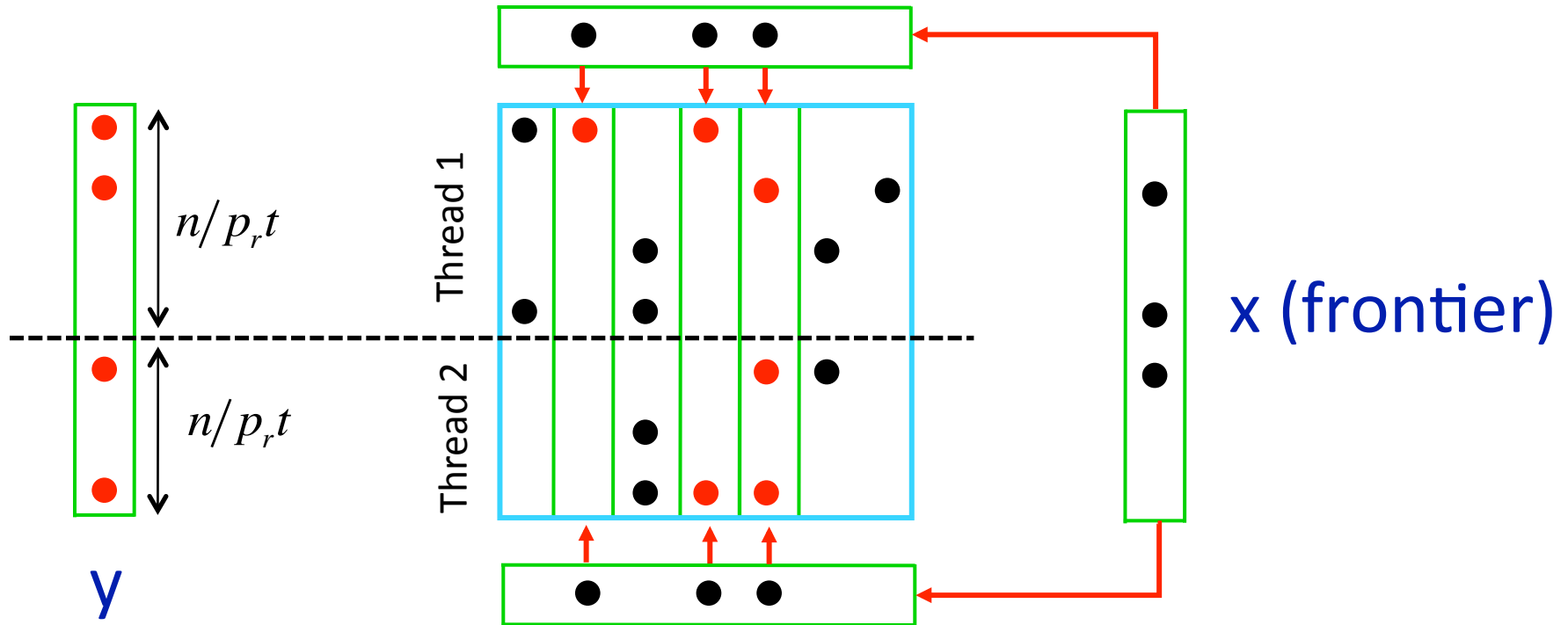
# Submatrix storage

Submatrices are “hypersparse” (i.e.  $nnz \ll n$ )



- A data structure or algorithm that depends on matrix dimension  $n$  (e.g. CSR or CSC) is asymptotically too wasteful for submatrices
- Use doubly-compressed (DCSC) data structures instead.

# 2D hybrid parallelism

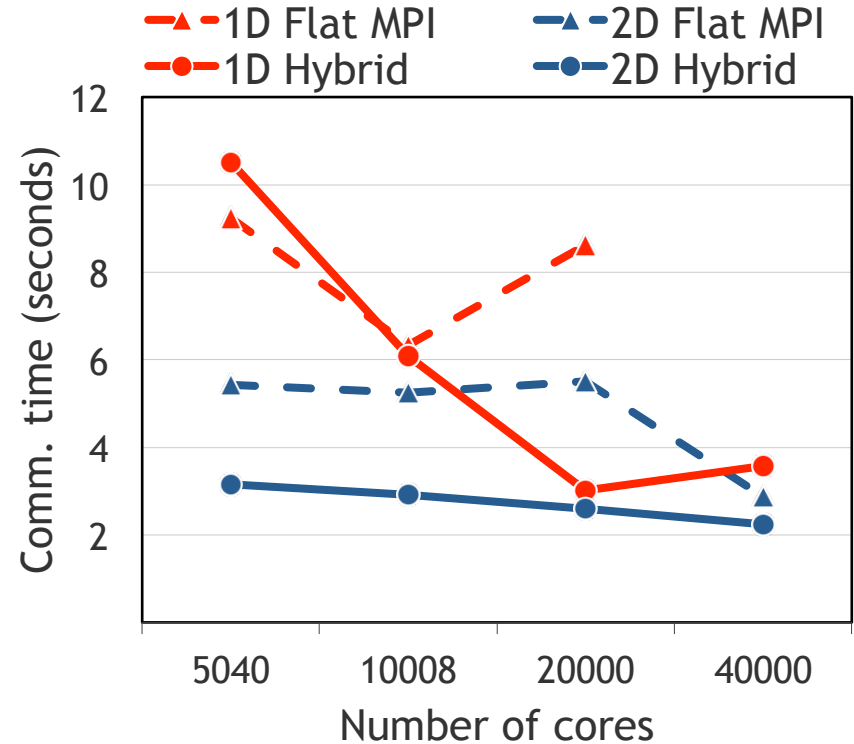
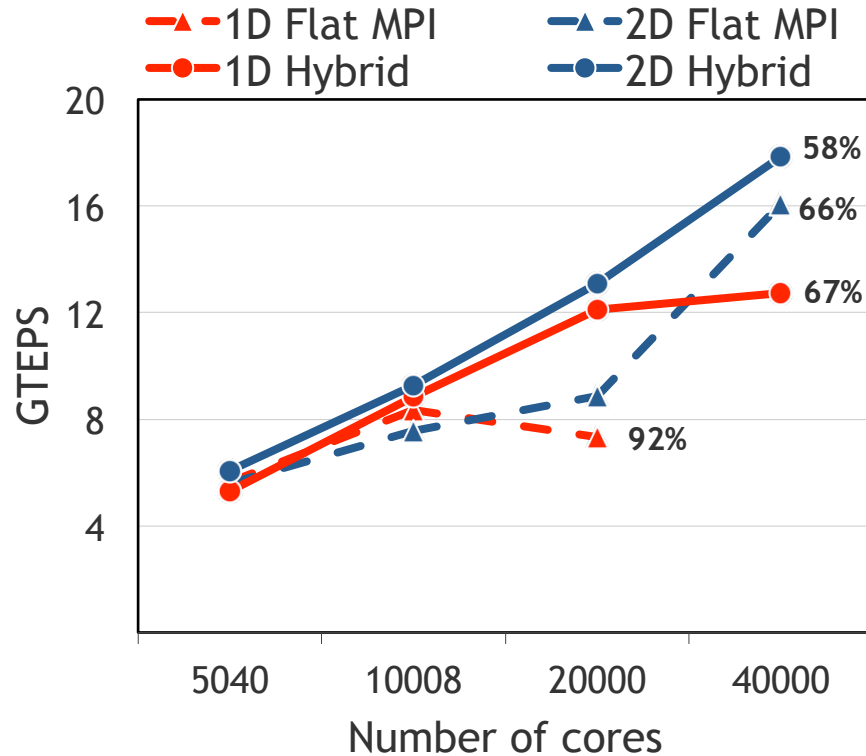


- Explicitly split submatrices to  $t$  (#threads) pieces along the rows.
- Local working set is smaller by a factor of  $\sqrt{t}$   
(not a factor of  $t$ , because  $p_r$  is now a factor of  $\sqrt{t}$  smaller as well)

# Outline

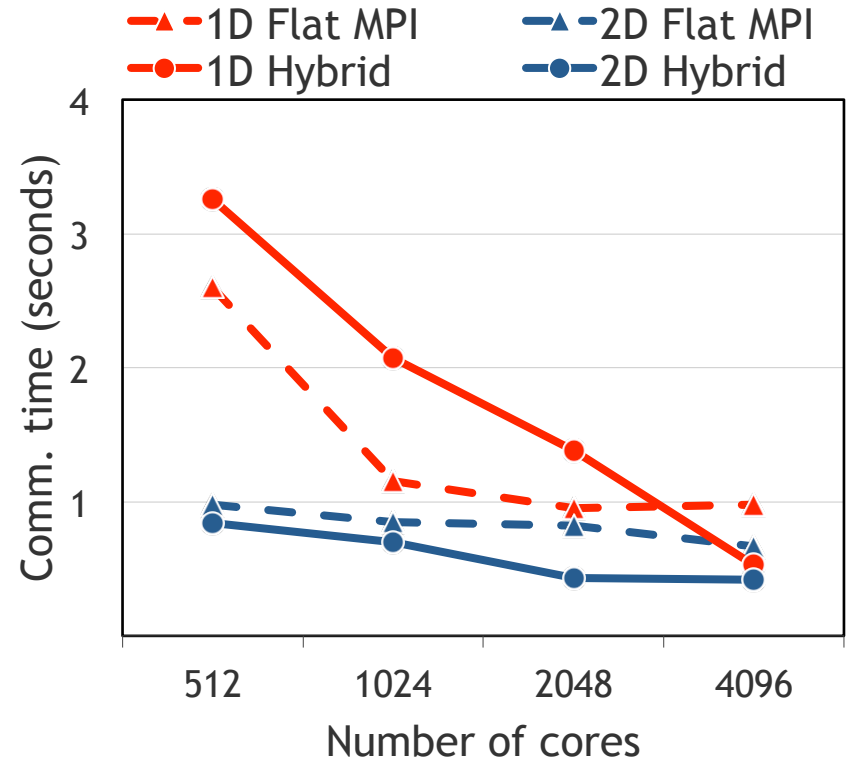
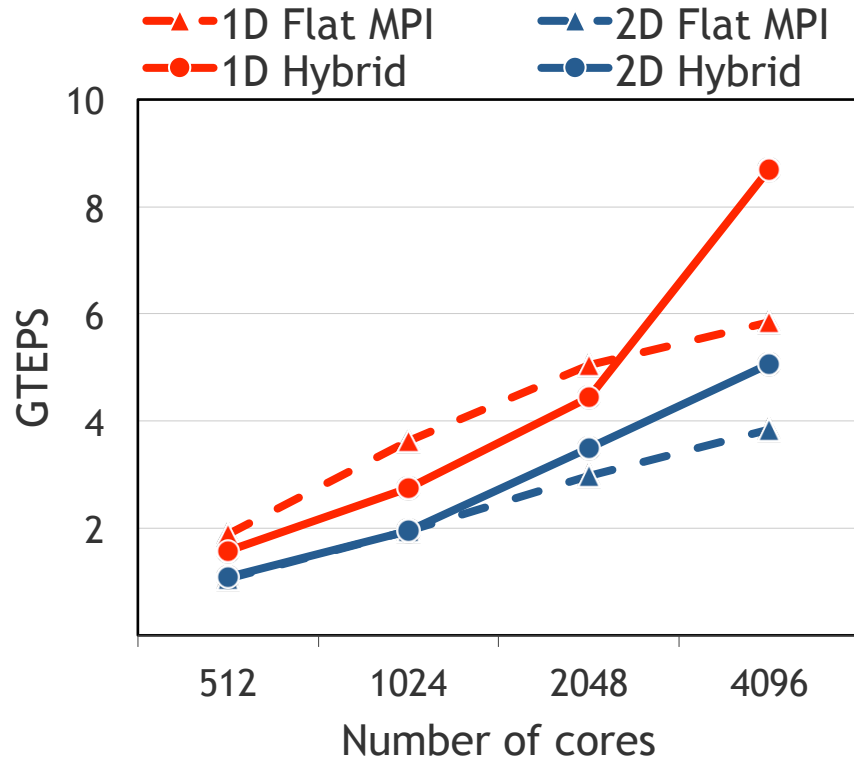
- BFS overview and applications
- BFS as sparse matrix-sparse vector multiply
- Parallel BFS: 1D and 2D approaches
- **Experimental results and insight**
- Conclusions / Contributions
- Future Directions

# Hopper strong scaling



- NERSC Hopper (Cray XE6, Gemini interconnect AMD Magny-Cours)
- Hybrid: In-node 6-way OpenMP multithreading
- Graph500: Scale 32, R-MAT with edgefactor=16

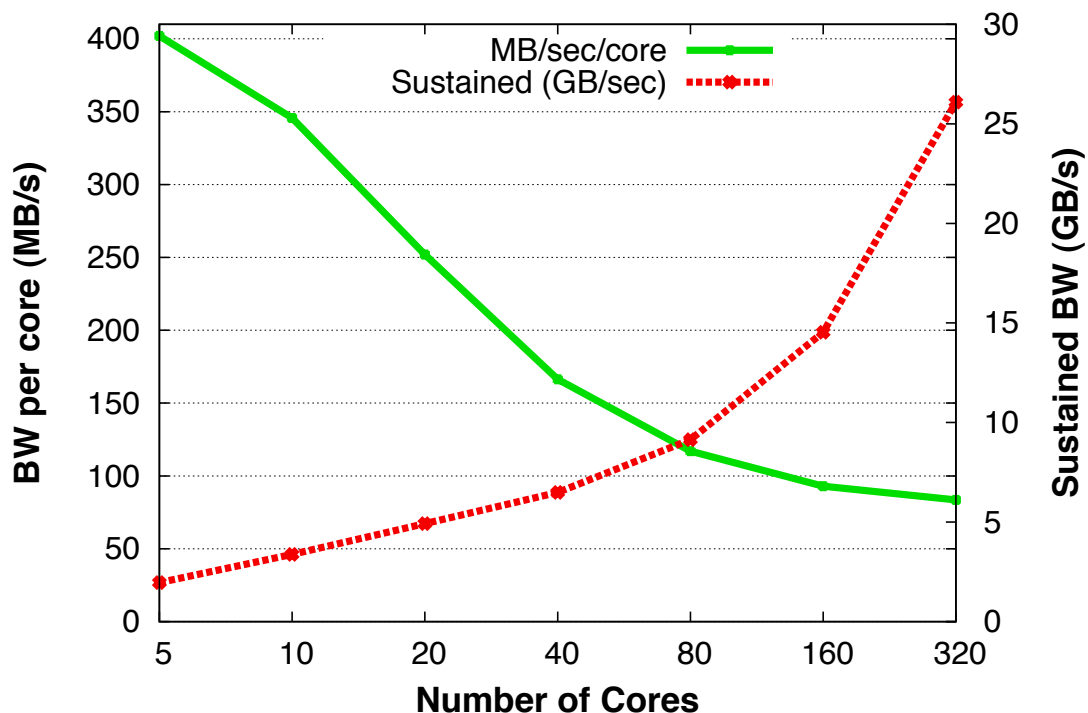
# Franklin strong scaling



- NERSC Franklin (Cray XT4, Seastar interconnect AMD Budapest)
- Hybrid: In-node 4-way OpenMP multithreading
- Graph500: Scale 29, R-MAT with edgefactor=16

# Bandwidth as a function of p

- The network parameter  $\beta_{N,a2a}$  is a function of participating processors.
- Micro-benchmark imitates 2D algorithm's communication pattern.



$$\beta_{N,a2a}(p_c) \frac{m}{p} + \beta_{N,ag}(p_r) \frac{n}{p_c}$$

**2D**

$$\beta_{N,a2a}(p) \frac{m}{p}$$

**1D**

\*: Latency costs not listed

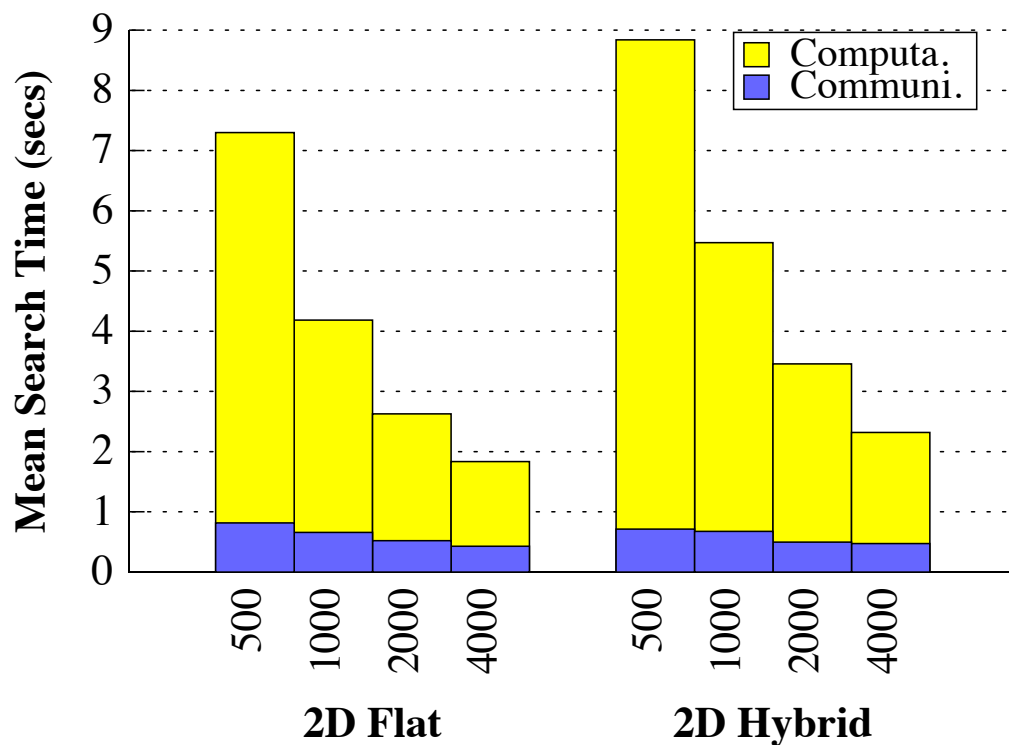
# Communication breakdown (2D)

Core count	Problem scale	Edge factor	BFS time (secs)	Allgatherv (percent.)	Alltoallv (percent.)
1024	27	64	2.67	7.0%	6.8%
	29	16	4.39	11.5%	7.7%
	31	4	7.18	16.6%	9.1%
2025	27	64	1.56	10.4%	7.6%
	29	16	2.87	19.4%	9.2%
	31	4	4.41	24.3%	9.0%
4096	27	64	1.31	13.1%	7.8%
	29	16	2.23	20.8%	9.0%
	31	4	3.15	30.9%	7.7%

$$\beta_{N,a2a}(p_c) \frac{m}{p} + \beta_{N,ag}(p_r) \frac{n}{p_c}$$

- i. Allgather becomes the bottleneck as concurrency increases.
- ii. Allgather sensitive to sparsity.

# A higher diameter graph



Lower is better

Vertices: 133M

Edges: 5.5B

Diameter  $\geq 140$

Run on Hopper

- Union of multiple 2005 crawls of the .uk domain
- Speedup: 4X when going from 500 to 4000 cores
- Hybrid is slower since remote communication is not the bottleneck.

# Conclusions / Contributions

- In-depth analysis and evaluation of all 4 combinations of parallel BFS on distributed memory

1D Flat MPI	2D Flat MPI
1D Hybrid MPI+OpenMP	2D Hybrid MPI+OpenMP

- Novel 2D hybrid approach: Up to 3.5X reduction in communication time compared to 1D
- Scaling to 40K cores on “scale-free” graphs
- A performance model that captures both local and non-local memory accesses, as well as collectives

# Some future work

1- Optimal processor grid dimensions ( $p = p_r \times p_c$ ) depends on:

- Graph size
- Graph density
- Desired concurrency
- Target architecture

*Great opportunity for autotuning.*

2- Performance depends heavily on collectives performance.

- Non-torus partitions -> unpredictable performance
- Topology aware collectives (Edgar Solomonik's talk at 4:30 today)

3- Graph/hypergraph partitioning for reducing communication

4- Prospect of using PGAS languages