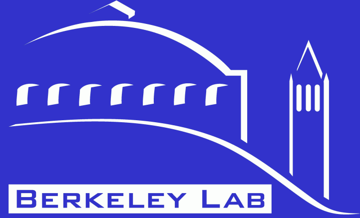


Tuning and communication reduction for graph and sparse matrix computations

Aydın Buluç
Lawrence Berkeley National Laboratory

Kamesh Madduri, Lenny Oliker, Sam Williams (LBNL)
John Gilbert (UCSB), James Demmel (UCB)

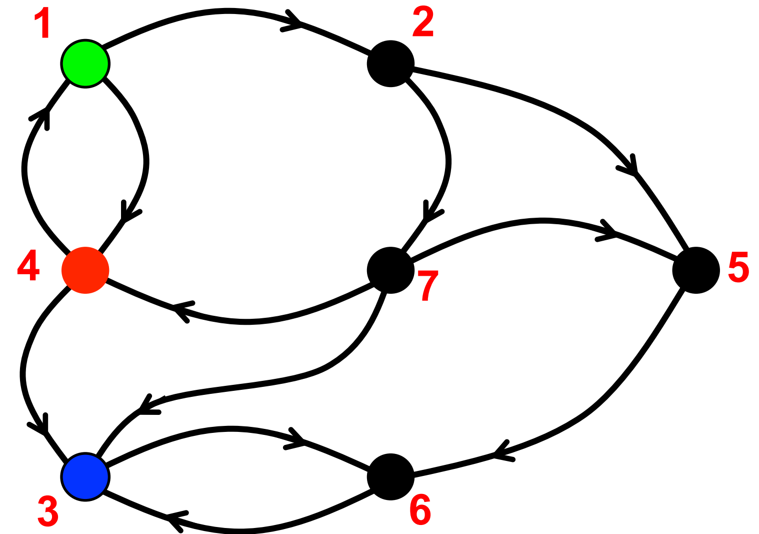
CScADS Autotuning workshop
August 9, 2011

Floating-point vs. graphs, June 2011

8.1 Petaflops

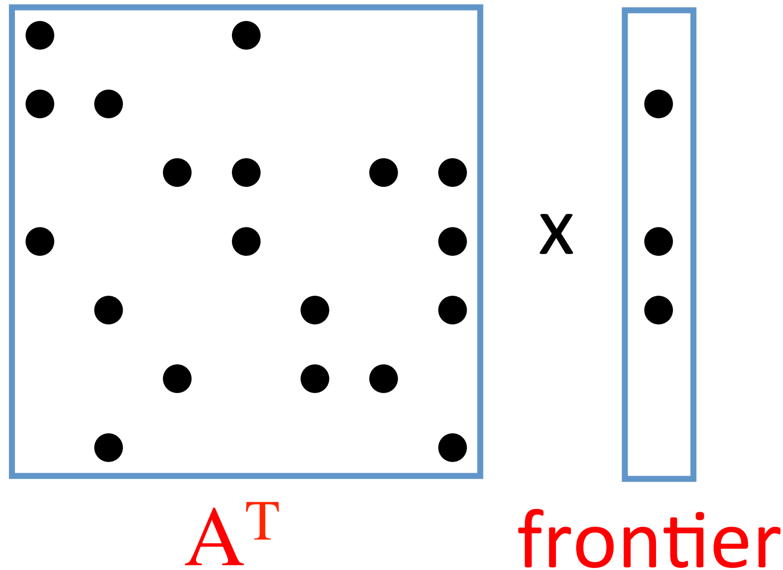
P A = L x U

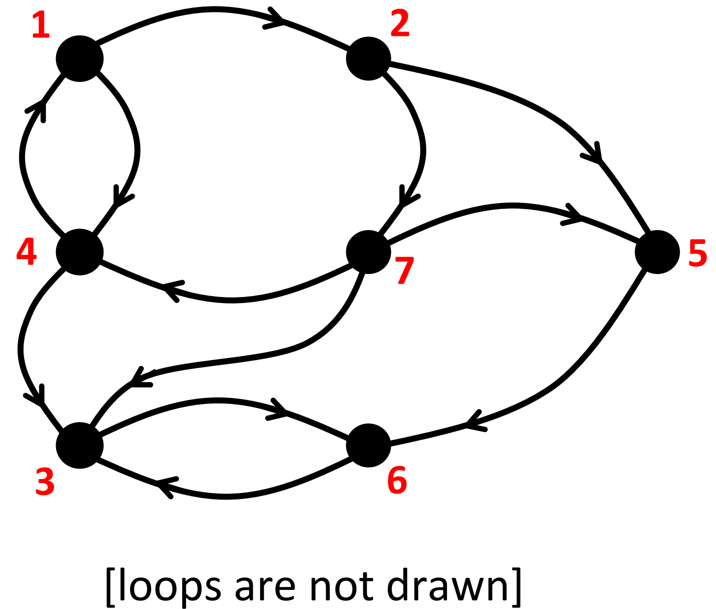
43 Gigateps



8.1 Peta / 43 Giga is about 190,000!

Sparse adjacency matrix and graph

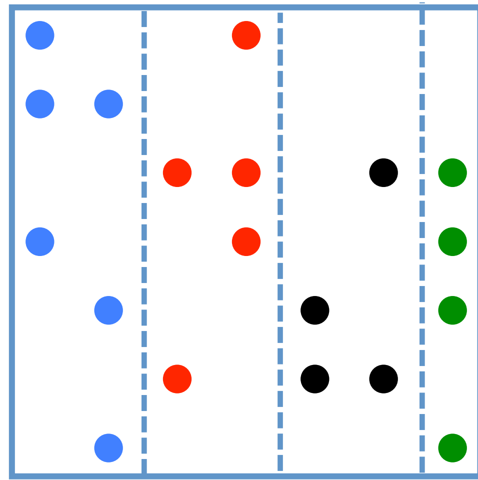




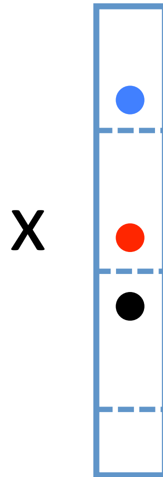
[loops are not drawn]

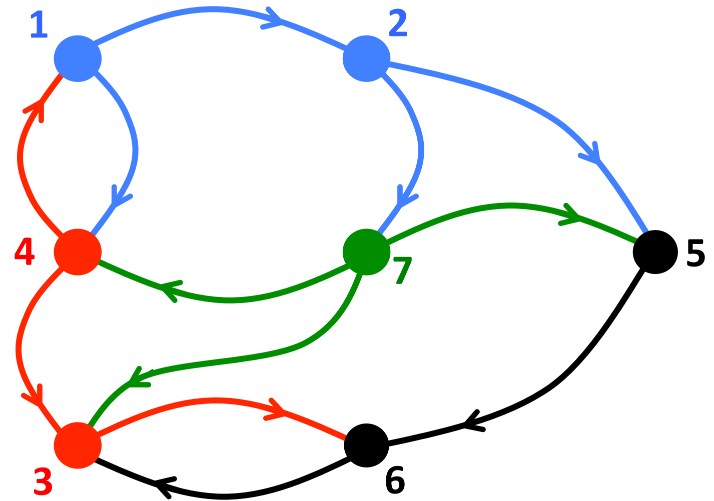
- Every graph is a sparse matrix and vice-versa
- Adjacency matrix: sparse array w/ nonzeros for graph edges
- Storage-efficient implementation from sparse data structures

1D Parallel BFS algorithm



x





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]

1D Parallel BFS 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:

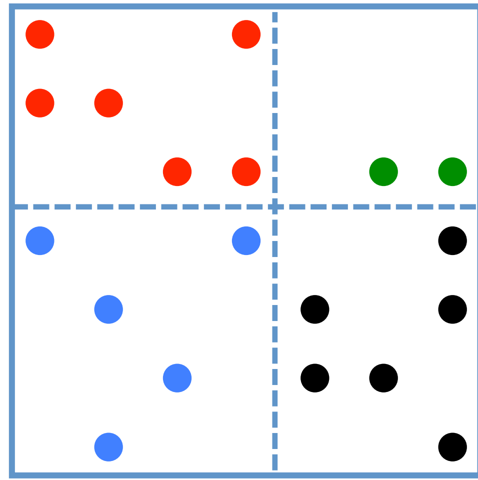
$$\beta_{N,a2a}(p)\frac{m}{p} + \alpha_N p$$

All-to-all remote bandwidth with p participating processors

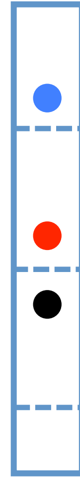
ALGORITHM:

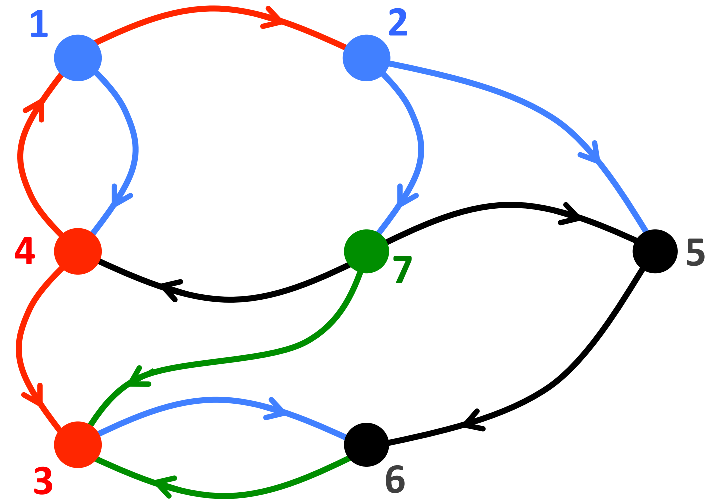
1. Find owners of the current frontier's adjacency [computation]
2. Exchange adjacencies via all-to-all. **[communication]**
3. Update distances/parents for unvisited vertices. [computation]

2D Parallel BFS algorithm



x

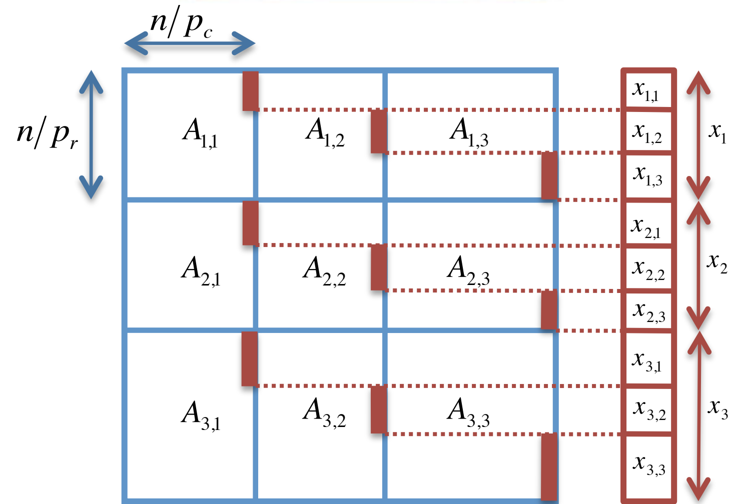


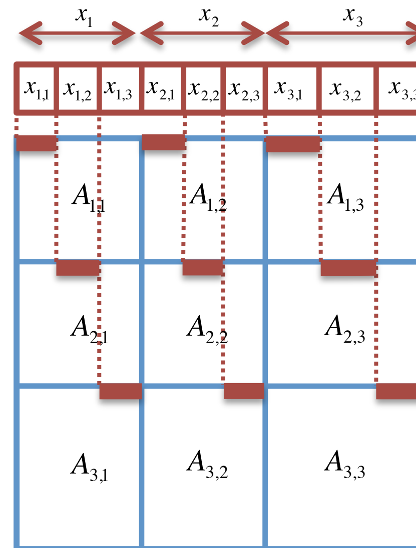


ALGORITHM:

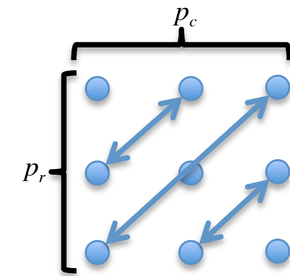
1. Gather vertices in *processor column* **[communication]**
2. Find owners of the current frontier's adjacency [computation]
3. Exchange adjacencies in *processor row* **[communication]**
4. Update distances/parents for unvisited vertices. [computation]

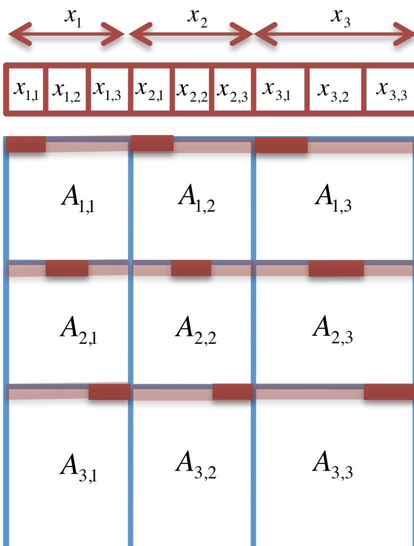
2D algorithm: Communication



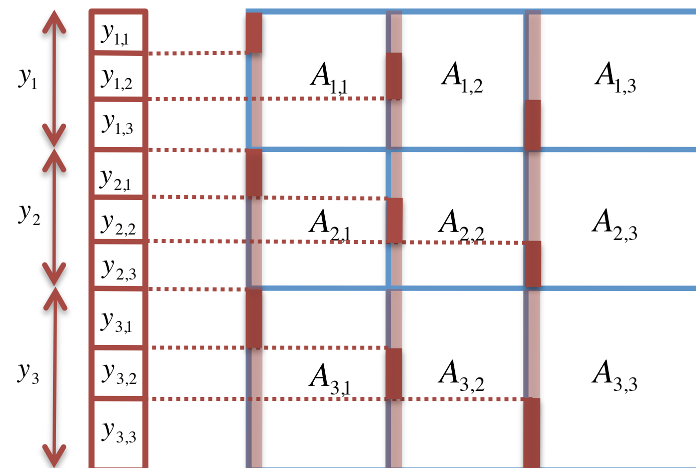


Pairwise
SendRecv
$O(n/p)$



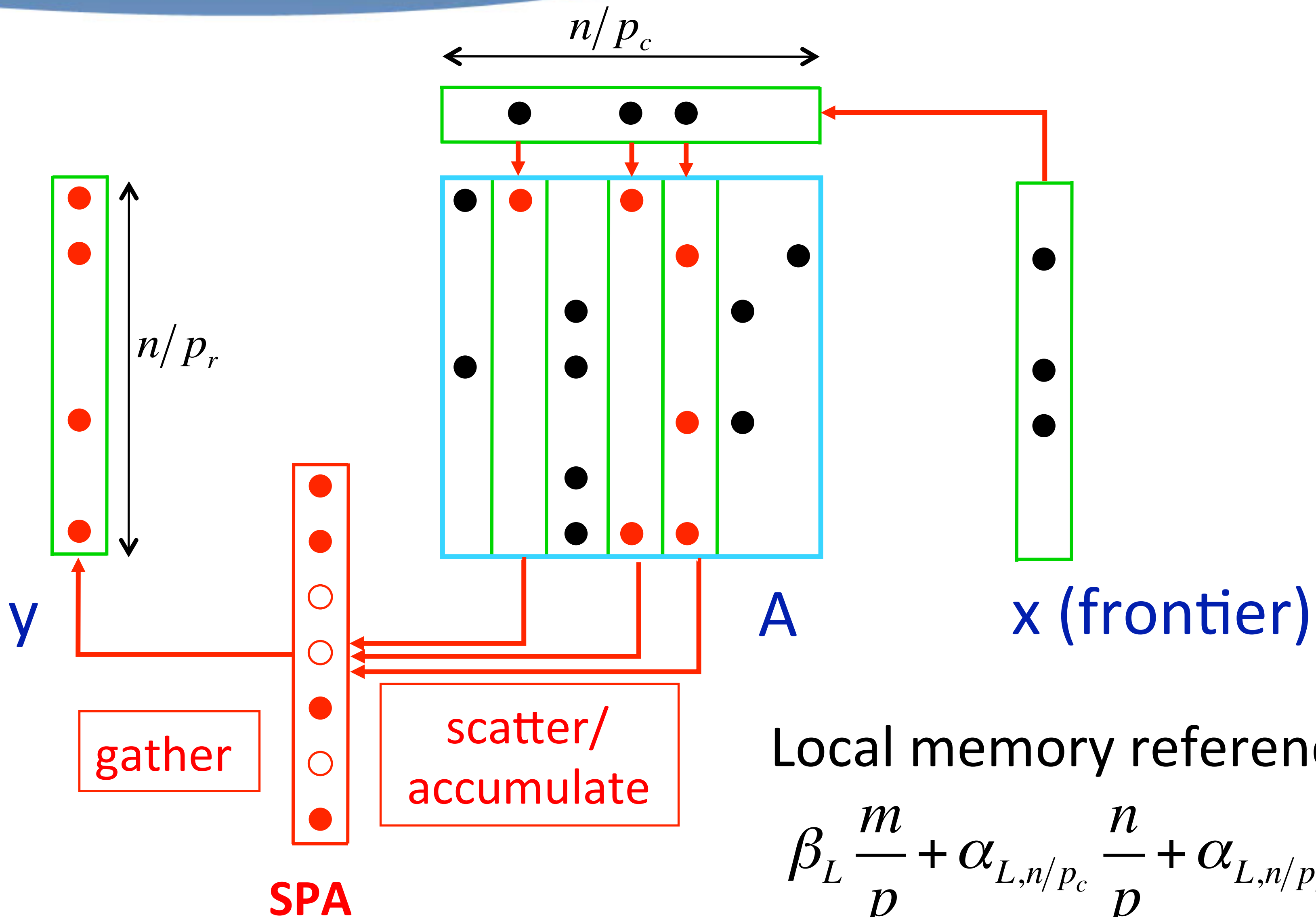


Columnwise
Allgatherv
$O(n/p_c)$



Rowwise
Alltoallv
$O(m/p)$

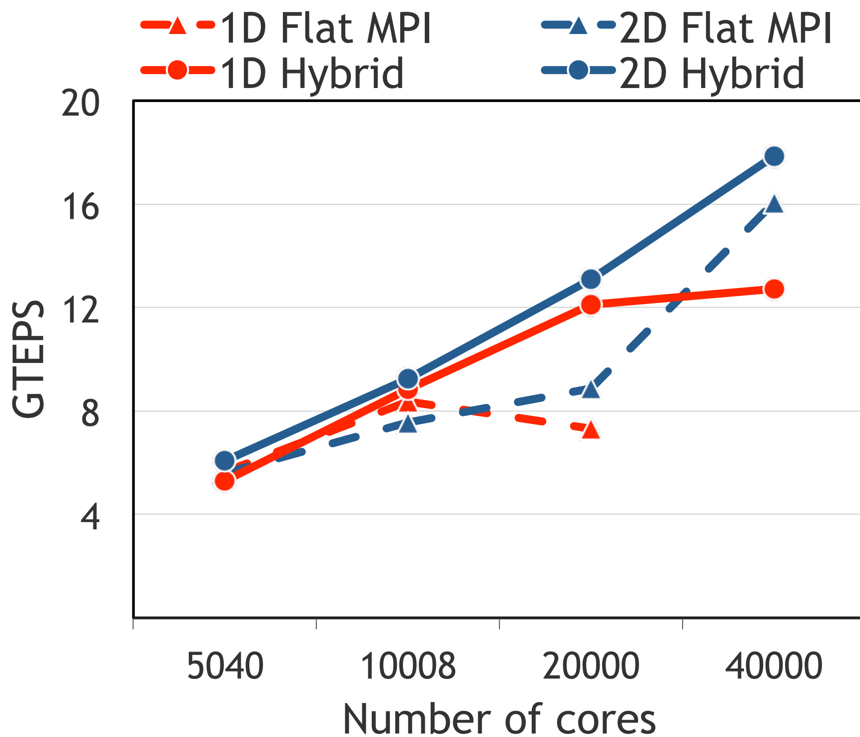
2D algorithm: Computation

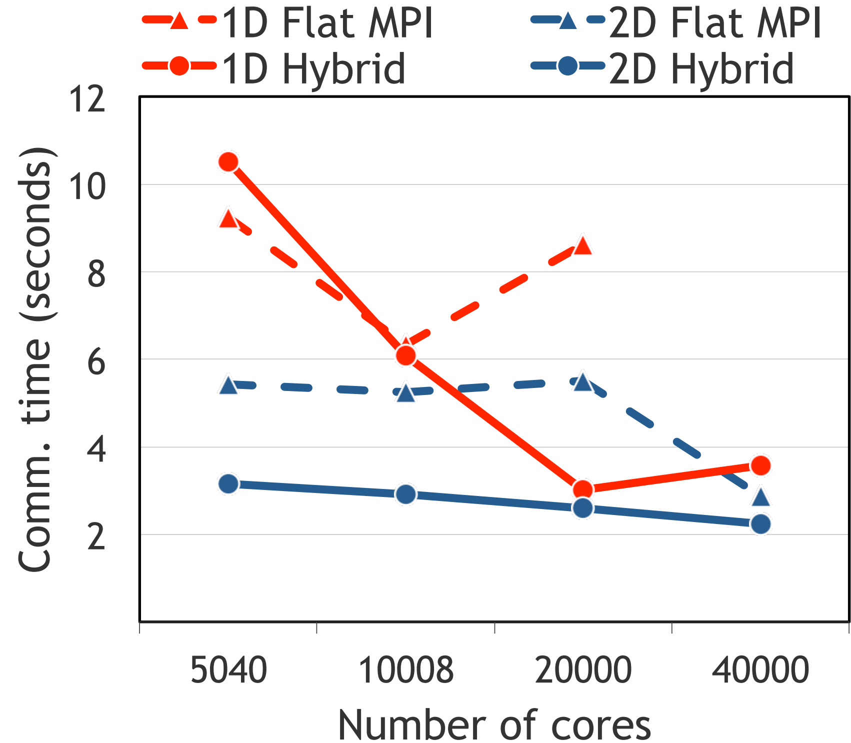


Local memory references:

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

Performance results (May 2011)





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

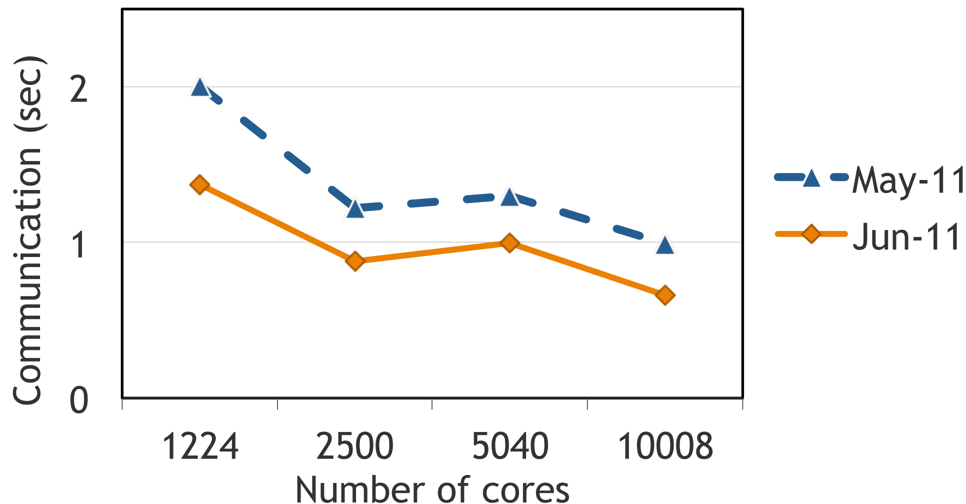
Recent improvements (July 2011)

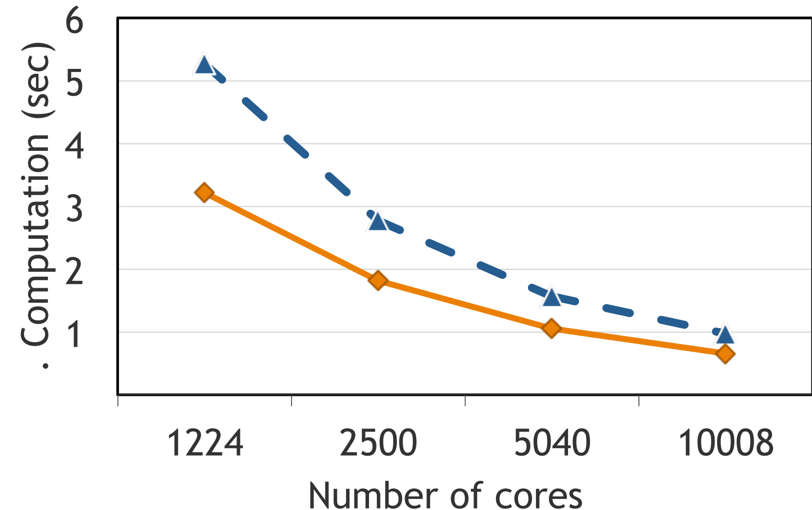
Two bottlenecks for 2D algorithm:

1) Local computation is expensive due to $O(n/p_r)$ working set
2) Communication involves 64-bit indices

Remedies:

1) Avoid the dense vector (SPA) in local SpMSV:
   - Select vertex with minimum label as parent (a bit vector suffices)
2) Local indices are 32-bit addressable in 2 out of 3 cases:
   - The receiver applies the correct sender "offset"





BFS autotuning parameters

Inputs:

1. Desired concurrency
2. Number of vertices (n) and edges (m)
3. Estimated via micro-benchmarks:
   a) Network parameters (e.g. $\beta_{N,a2a}$, $\beta_{N,ag}$), as a function of participating processors.
   b) Microprocessor parameters (e.g. $\alpha_{L,x}$, β_{L}), as a function of working set |x|.

Variables to tune:

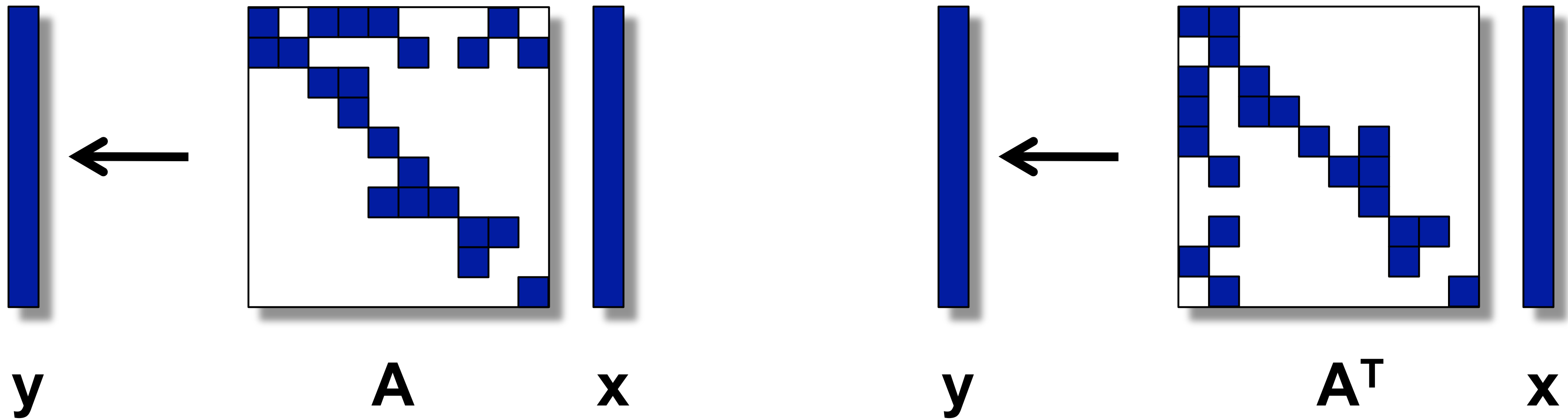
1. Best data distribution (1D vs 2D) and processor grid dimensions ($p=p_r \times p_c$)
2. To use intra-node threading or not.

Smooth transition in remote communication costs:

$$\beta_{N,a2a}(p_c)\frac{m}{p} + \beta_{N,ag}(p_r)\frac{n}{p} + \alpha_N p \quad \xrightarrow{p_r=1,\ p_c=p} \quad \beta_{N,a2a}(p)\frac{m}{p} + \alpha_N p$$

2D → **1D**

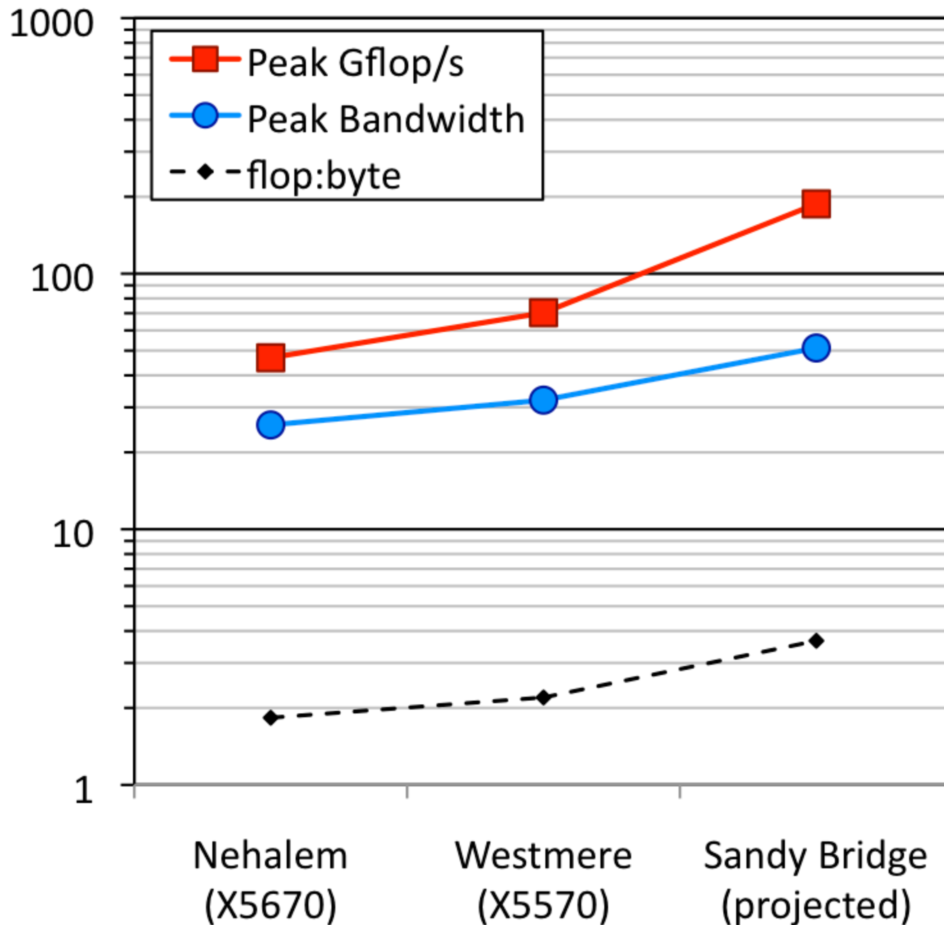
Sparse matrix-dense vector multiplication (SpMV)



A is an **n-by-n sparse** matrix with **nnz** << n^2 nonzeros

- *Iterative methods for solving linear systems*
  - *Symmetric SpMV (CG) can potentially go twice as well.*
  - *Some unsymmetric methods need transpose (QMR, BiCG)*
- *Principal component analysis, eigenvalues (pagerank)*

Future of SpMV on multicores



More flops/s, because:

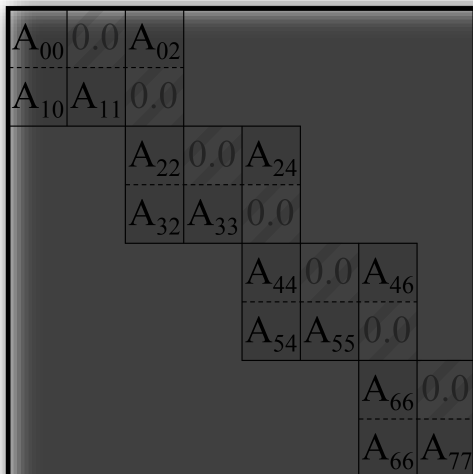
- Wider SSE Registers
  - x *Totally wasted*
- More cores
  - ✓ *Partially used*

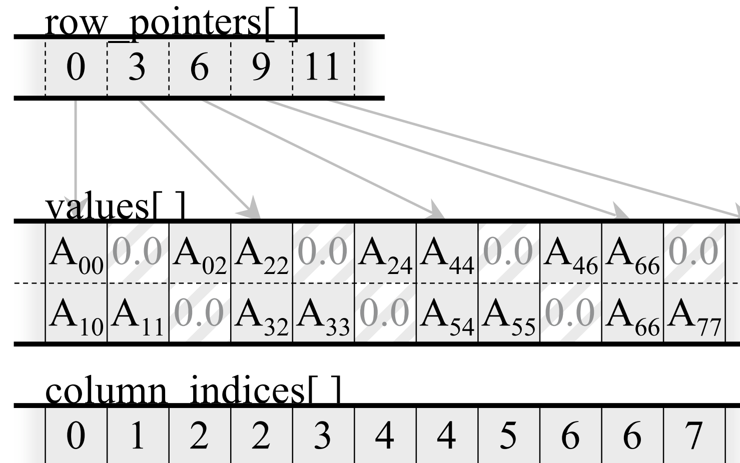
Problem: SpMV is bandwidth-bound

i. Save bandwidth.
ii. Retain parallelism.
iii. Use ILP at your disposal.

Reducing index bandwidth

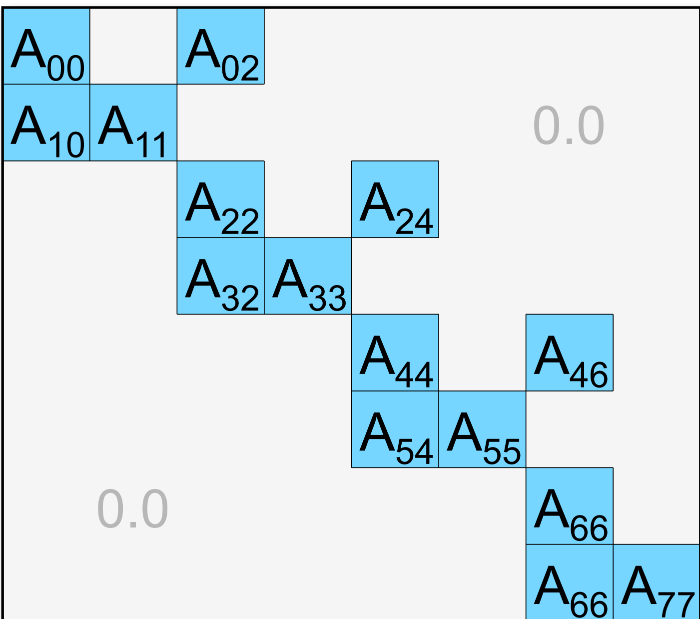
- Canonical approach: Register blocking
- Example data structure: BCSR
  - Nonzeros are grouped into dense r x c blocks
  - Some explicit zeros are added
  - Performance can hurt especially in DP as streaming explicit zeros is wasteful



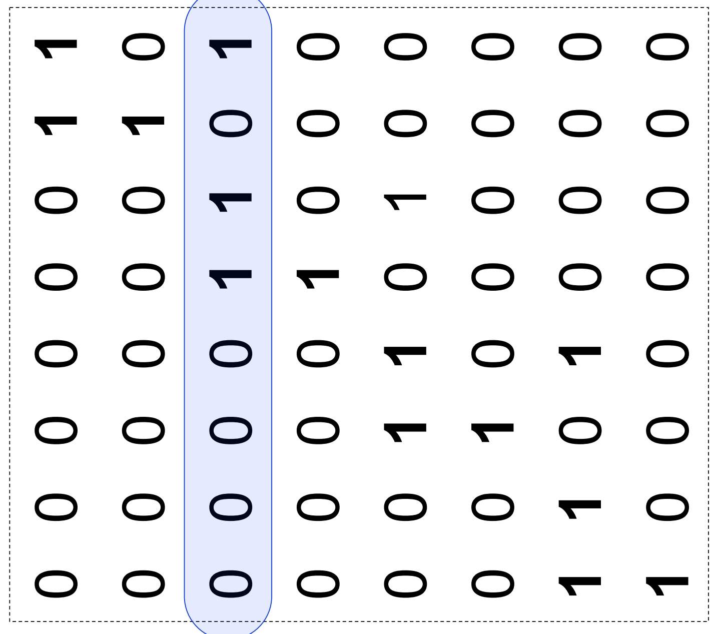


Bitmasked register blocks

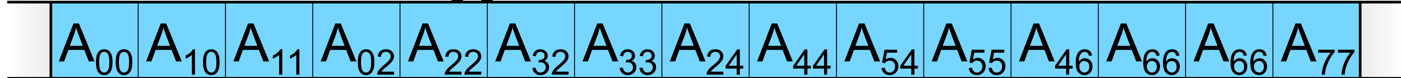
visualization of register block



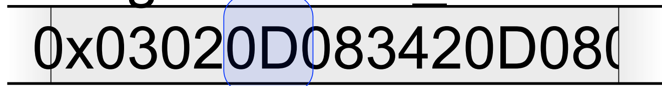
64-bit bitmask of nonzero positions



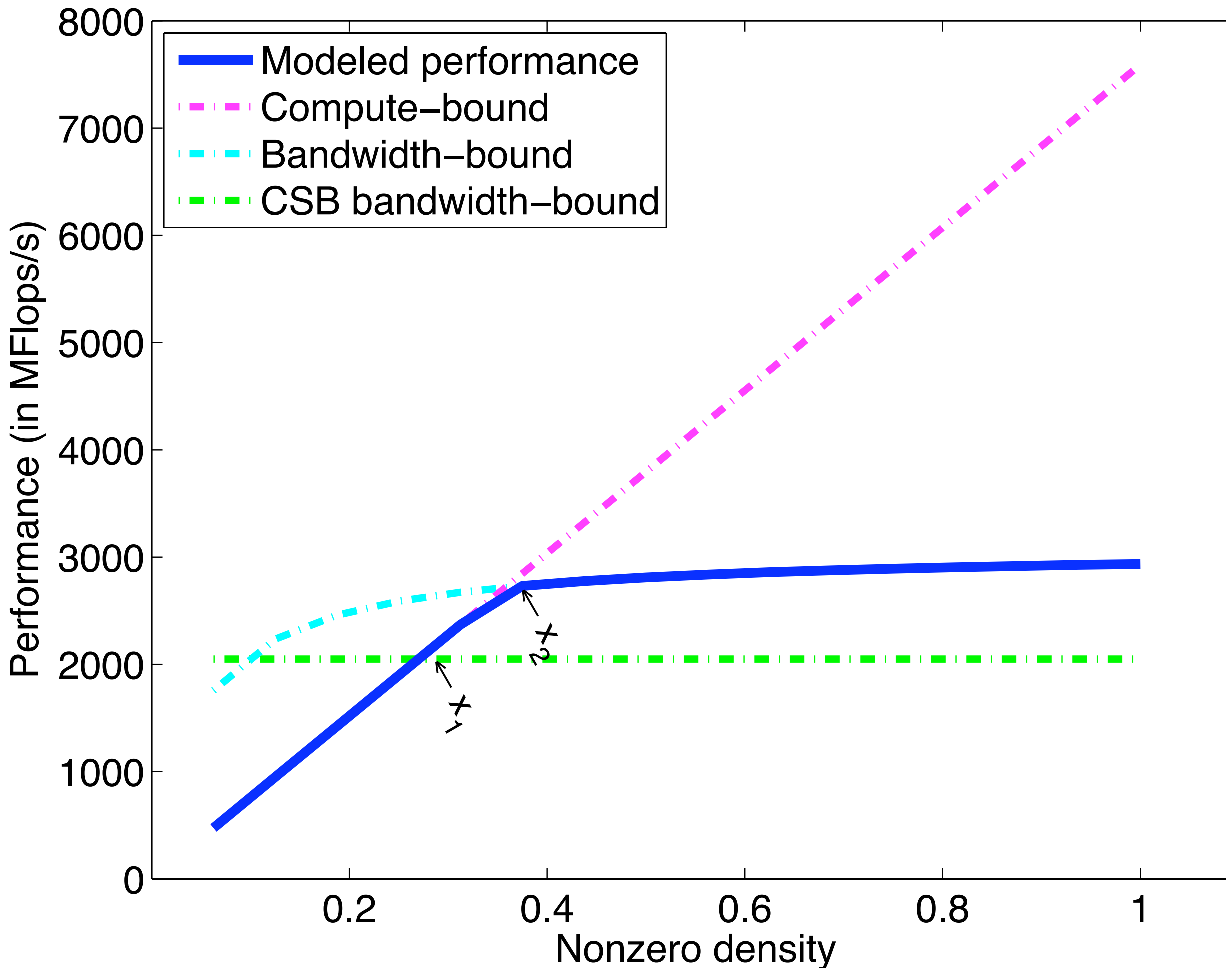
NonzeroValues[ ]



RegisterBlock_Bitmasks[ ]



Performance Model



Architecture specific

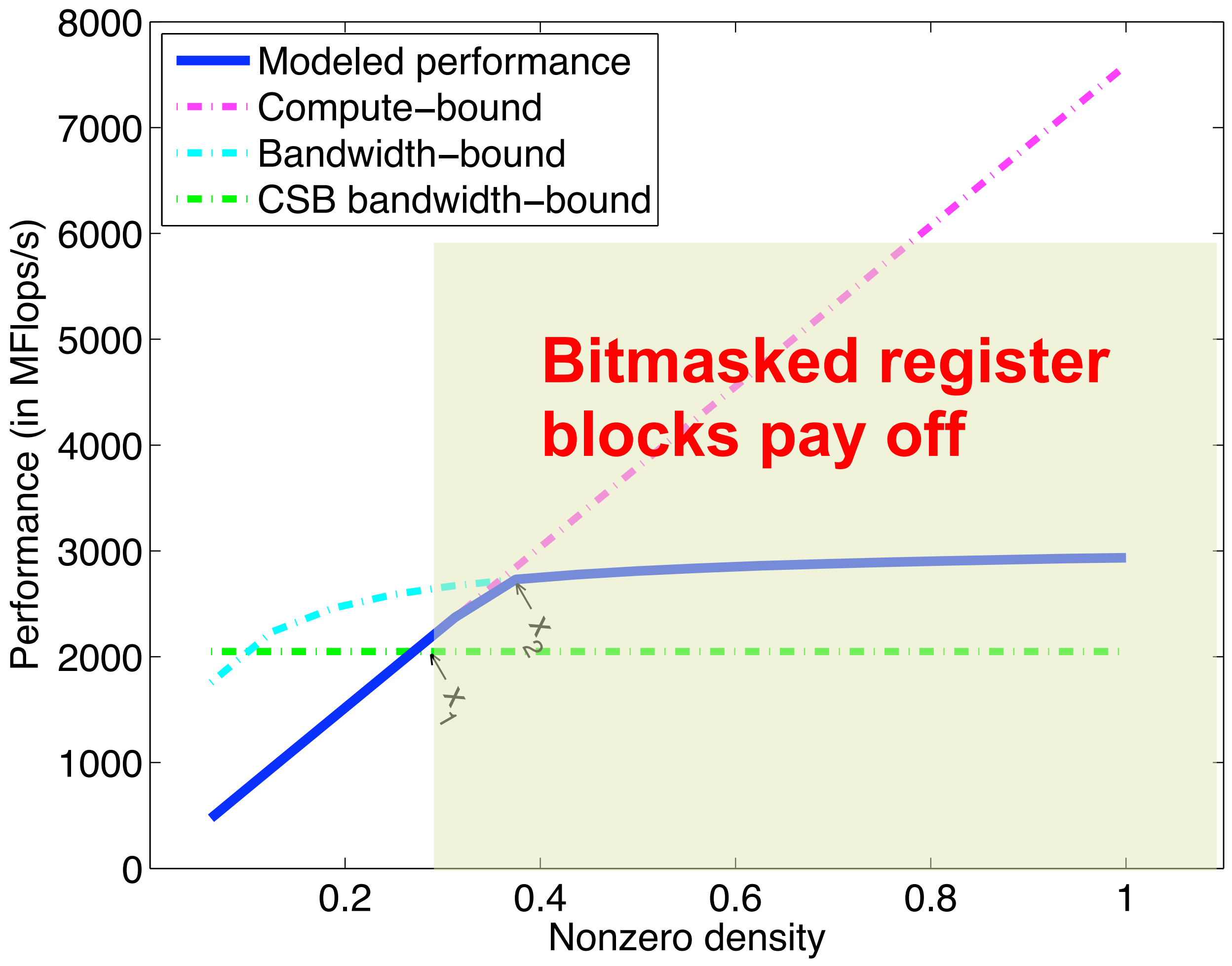
- *Nehalem-EX*
- *16 threads*
- *8 cores*

For a particular register block size

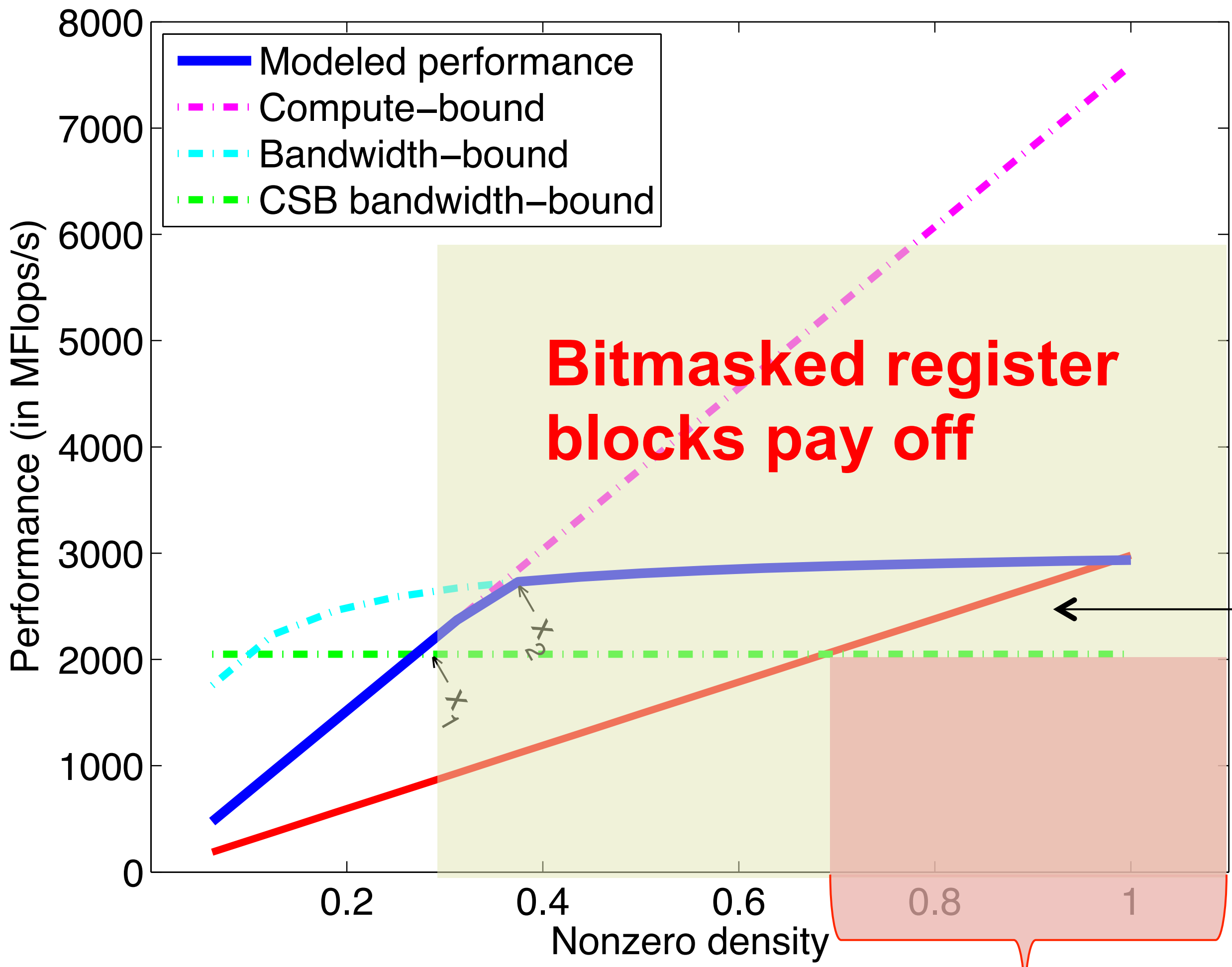
- *r* = *4*

Matrix independent model construction

Performance Model



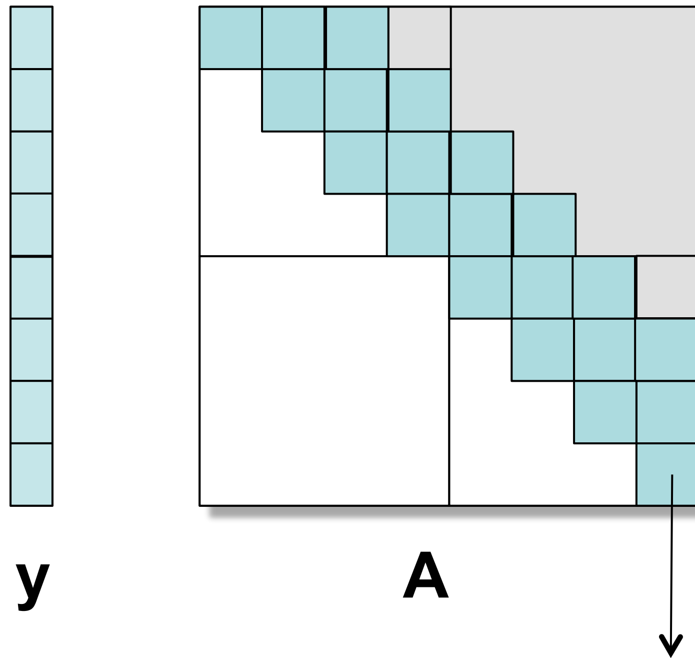
Performance Model



Bandwidth-bound performance using regular register blocks (with explicit fill)

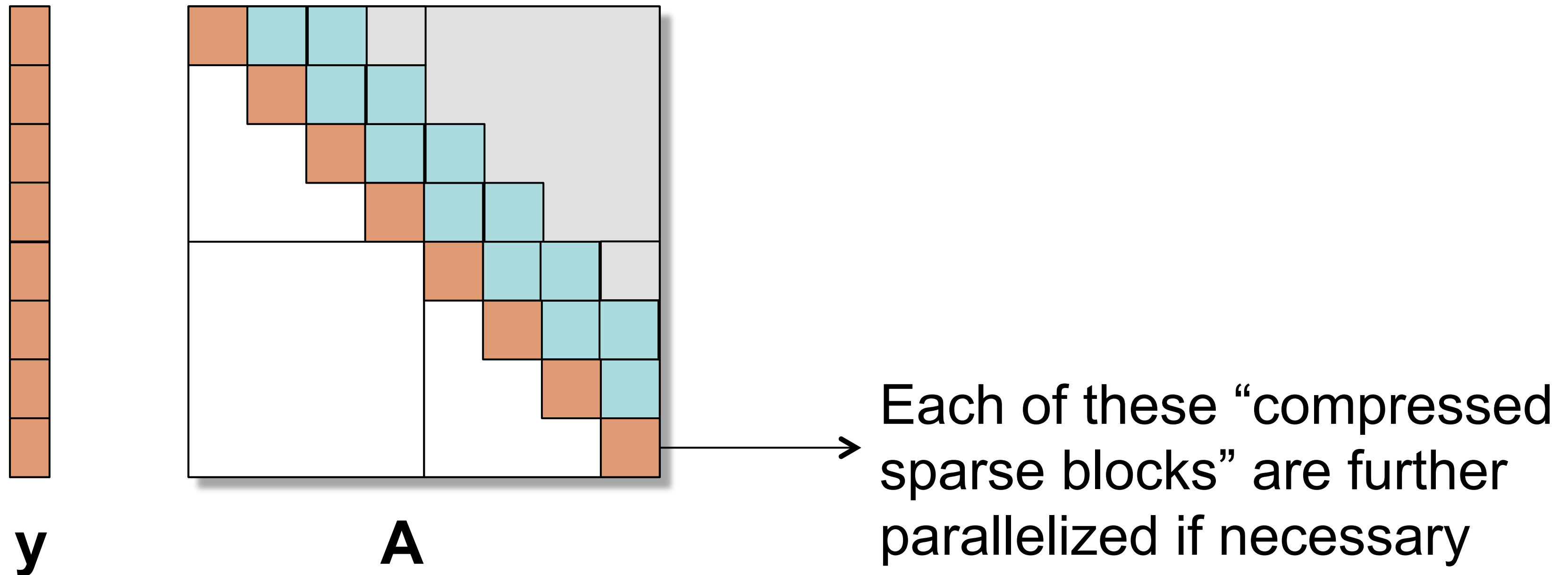
Regular register blocks pay off

Symmetric algorithm



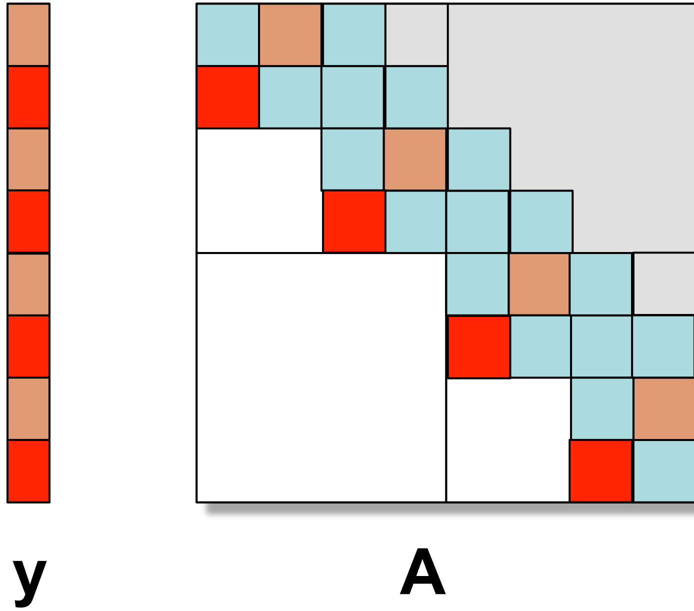
- Each of these boxes is a βxβ "compressed sparse block" for $\beta \cong \sqrt{n}$,
- Nonzeros are stored in Z-morton order

Symmetric algorithm

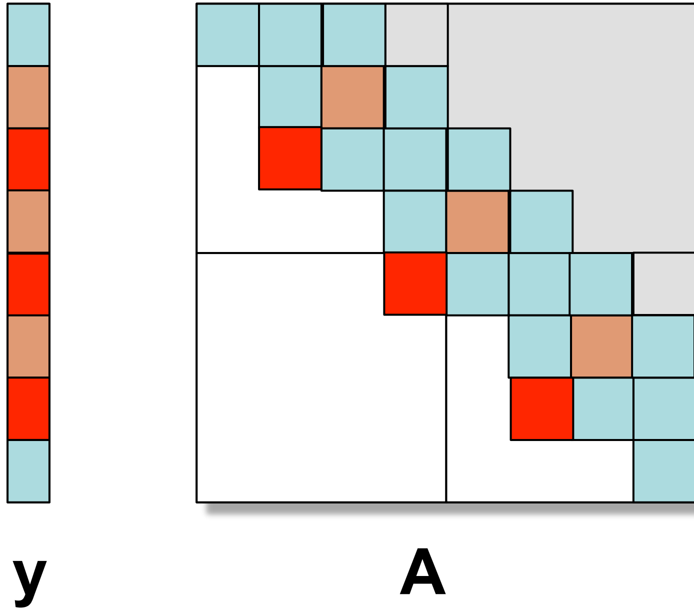


- Dynamically scheduled nested parallelism
- Via work stealing

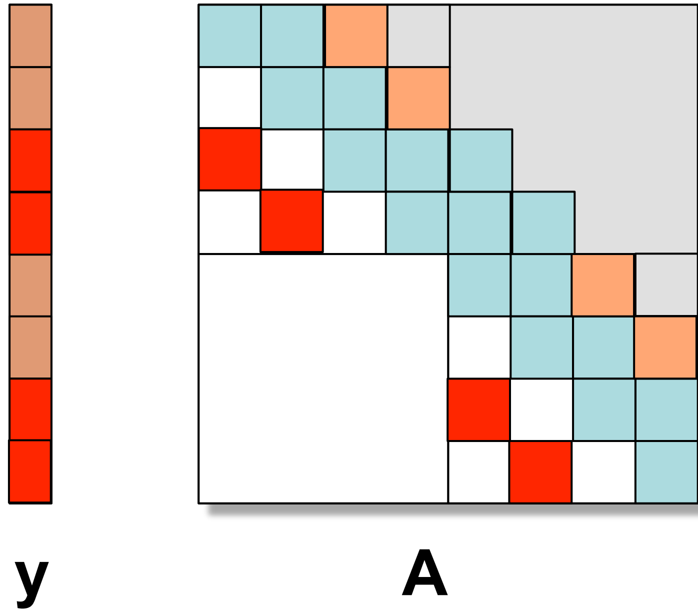
Symmetric algorithm



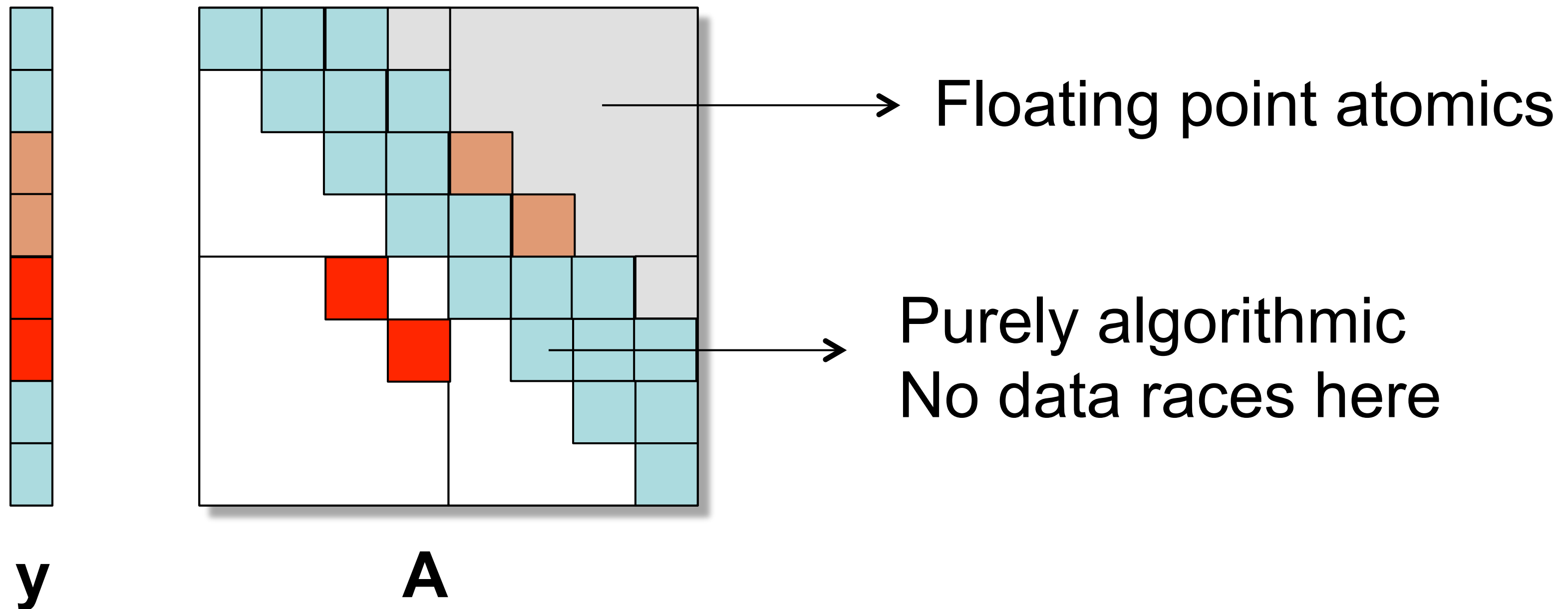
Symmetric algorithm



Symmetric algorithm

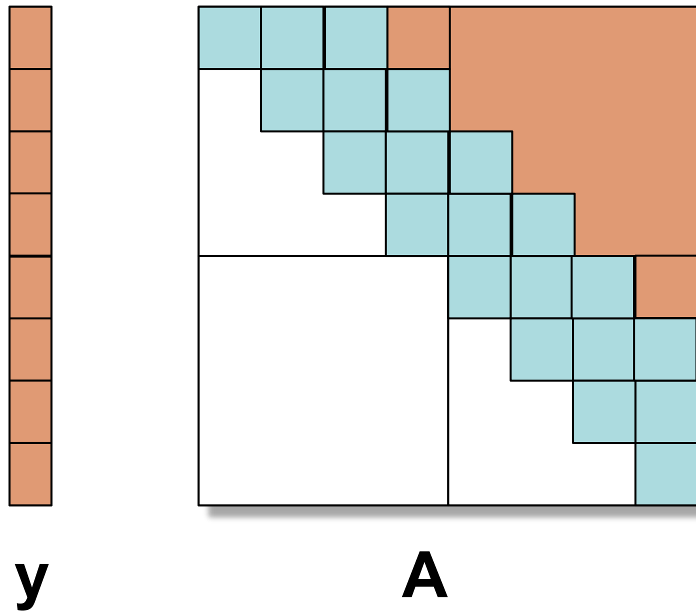


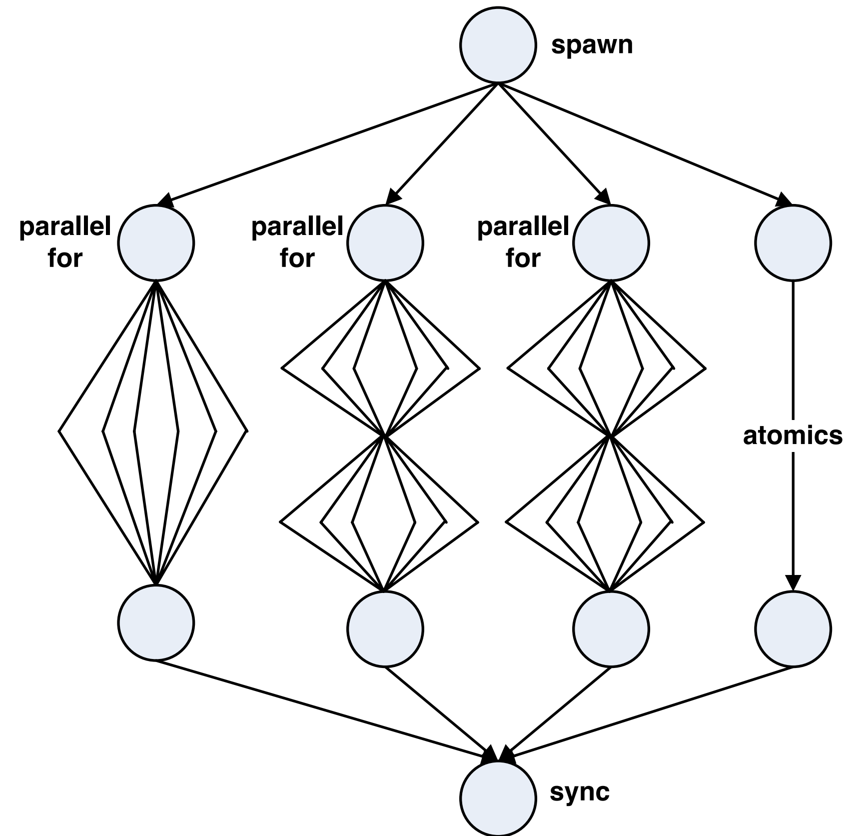
Symmetric algorithm



- Too many diagonal “levels” might limit parallelism.
- Optimal number of levels is *(1) architecture, (2) matrix, (3) run-time* dependent

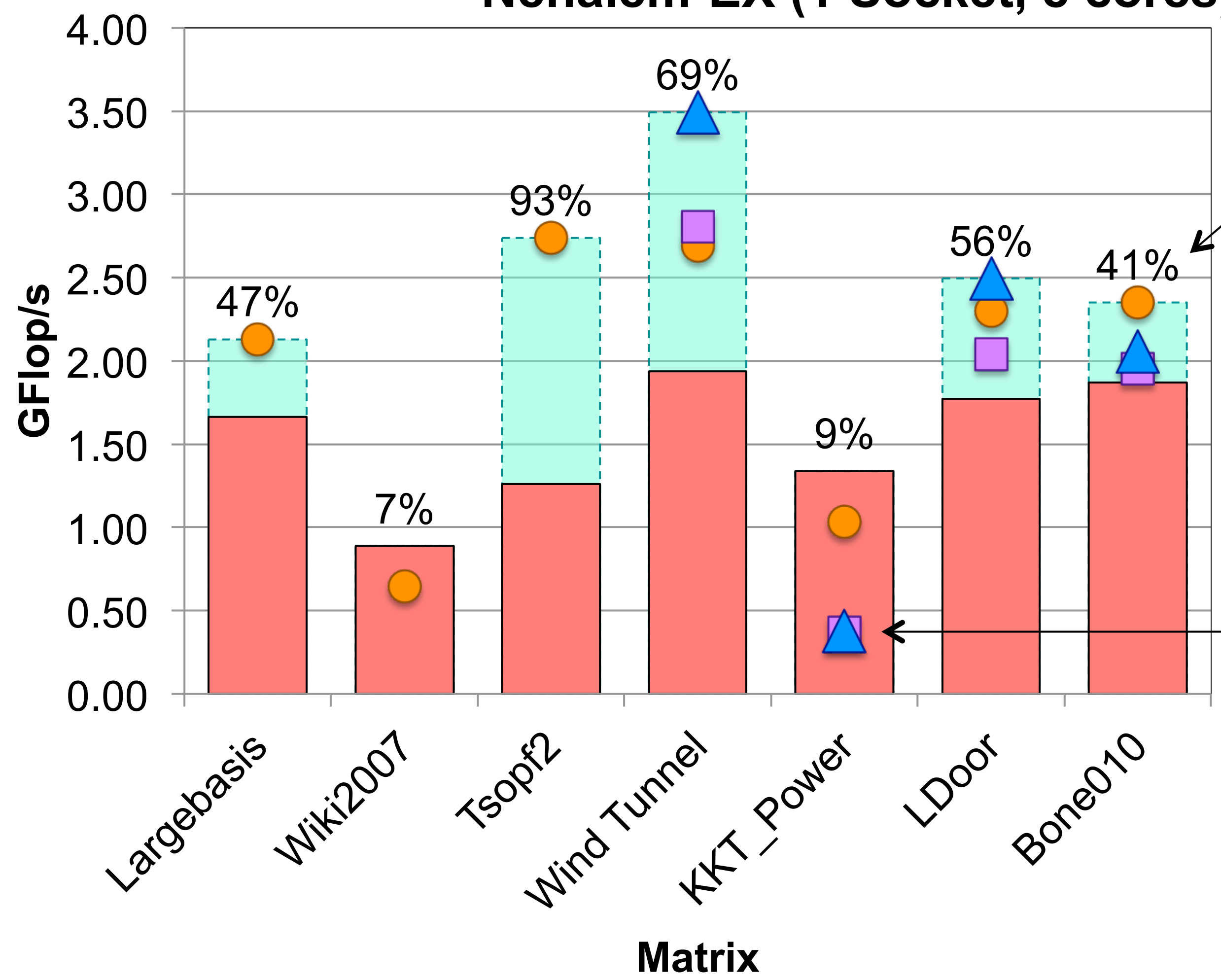
Symmetric algorithm





- Experiments and diagram shown on three diagonal levels
- Strategy is extendible (guided by *auto-tuning*)

Performance results



Nonzero density for r = 4 *[6.25 – 100%]*

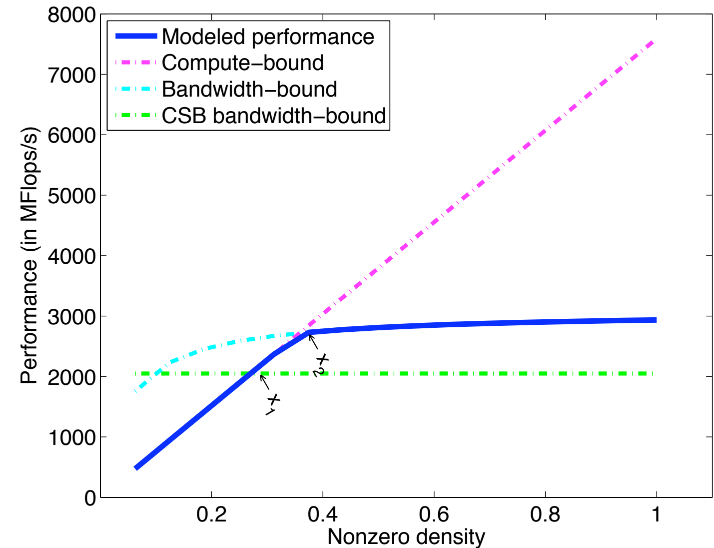
Large matrix bandwidth

Baseline: Paralel CSB with Intel Cilk

SpMV autotuning parameters

Main Inputs:

- #{hardware threads} available.
- Sparse matrix (and its RCM ordering)



for bitmasked register blocking:

for symmetric algorithm:

- Amount of parallelism on each block diagonal (via cilkscreen)
- Atomics performance (determines the number of block diagonals)

Variables to tune:

1. #{block diagonals treated algorithmically}
2. #{temporary vectors} → #{synchronizations between diagonals)
3. (potentially variable) register block dimensions

Future Work

Communication avoiding algorithms for BFS:

- Even for 2D Hybrid, communication takes 58.18% on 40K cores.
- Avoid the post-multiply communication phase.
- Replicate a bitmask array of $O(n/p_r)$ along the processor row.

"Fastest SpMV (and SpMV_T) in the West" via:

- full autotuning
- variable splitting
- AVX instructions

More 2D graph algorithms in the language of linear algebra:

- SSSP via delta stepping
- Maximal independent sets via Luby's algorithm

References

Graph work:

Aydın Buluç and Kamesh Madduri. Parallel breadth-first search on distributed memory systems. To appear, Supercomputing (SC'11).

SpMV work:

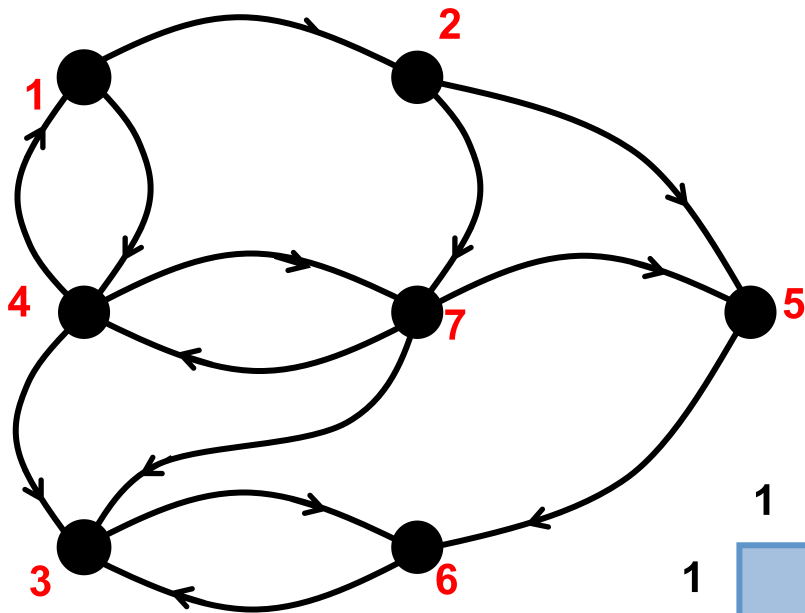
Aydın Buluç, Samuel Williams, Leonid Oliker, and James Demmel. Reduced-bandwidth multithreaded algorithms for sparse matrix-vector multiplication. In *Proc. IPDPS*, 2011

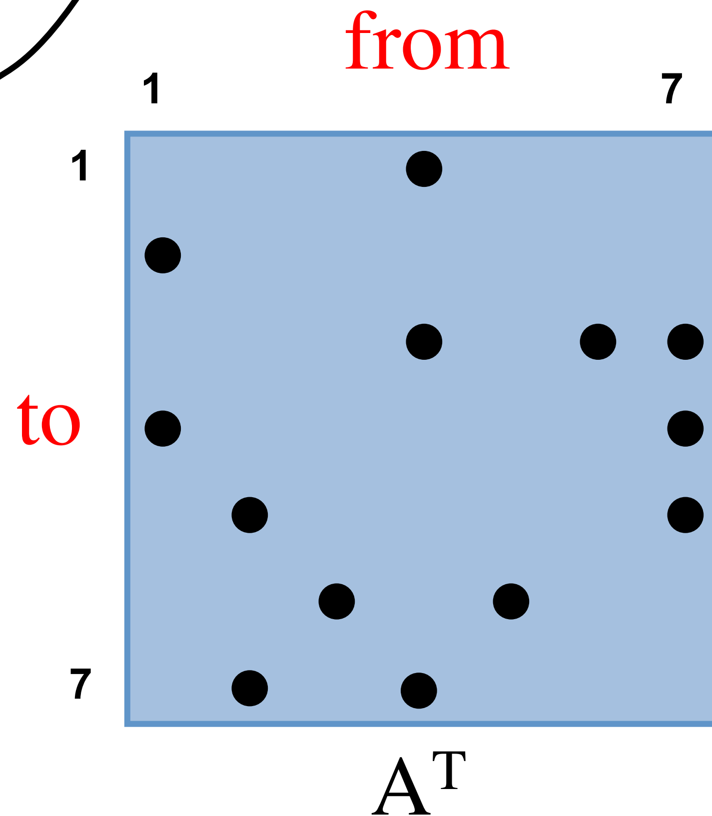
BFS optimizations and 2D algorithms incorporated into the Combinatorial BLAS, a parallel MPI library featuring graph primitives:
http://gauss.cs.ucsb.edu/~aydin/CombBLAS/html/index.html

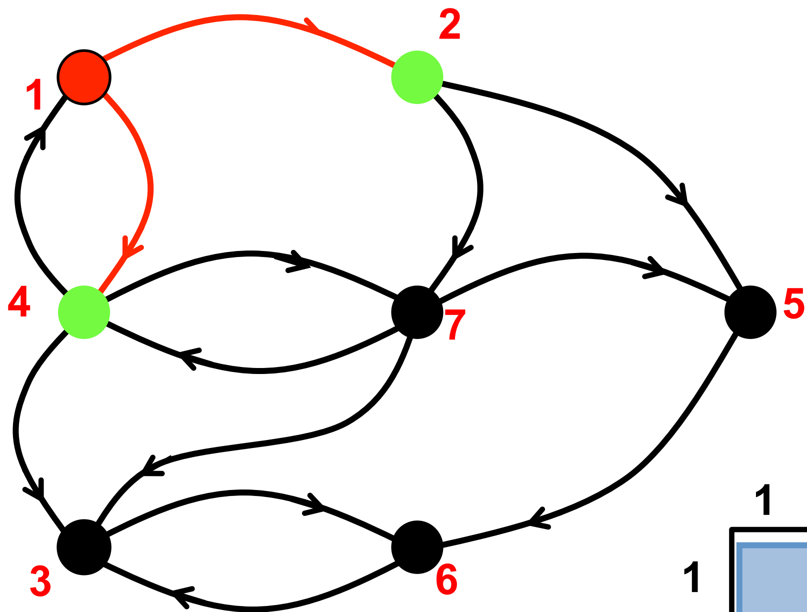
Aydın Buluç and John R. Gilbert. The Combinatorial BLAS: Design, implementation, and applications. *The International Journal of High Performance Computing Applications*, 2011.

Extra slides for SPA free SpMSV

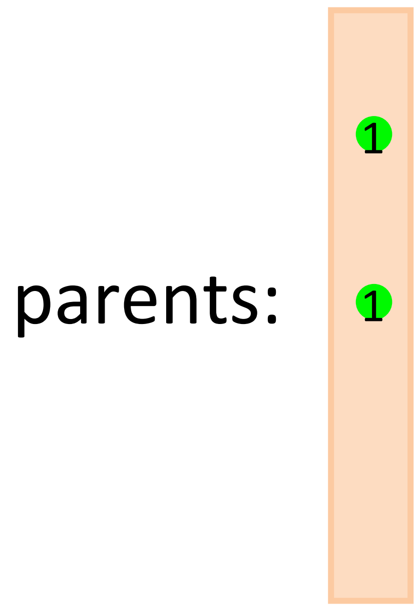
BFS sparse array implementation



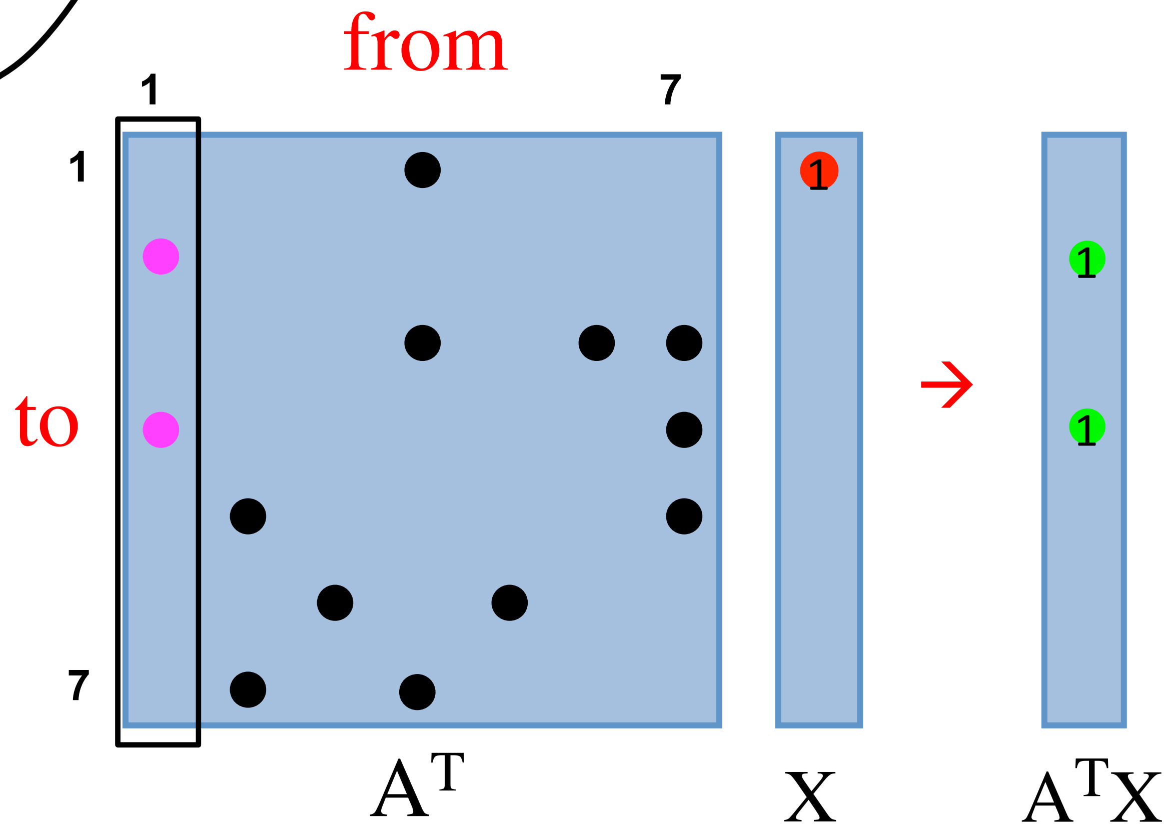




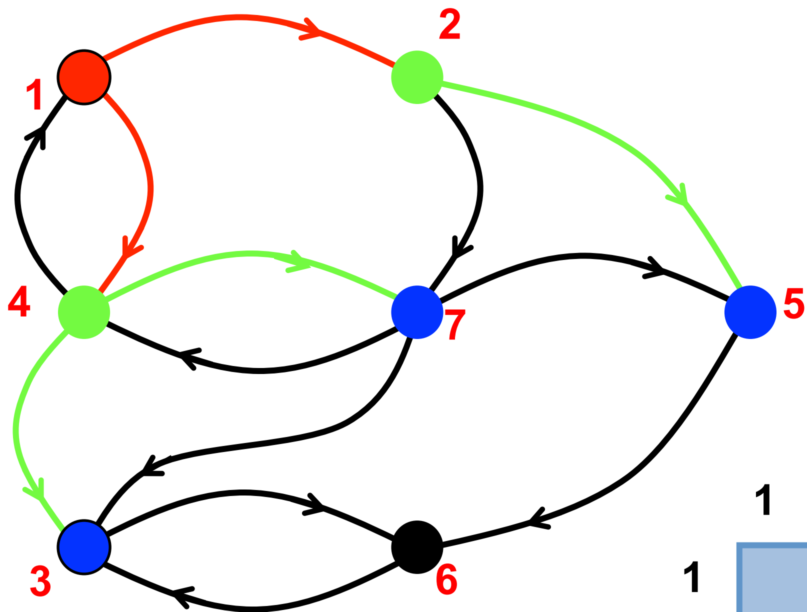
1
2
4
7
5
3
6



parents:
1
1

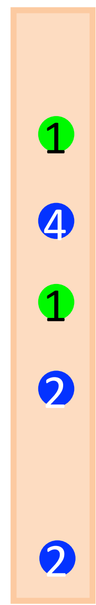


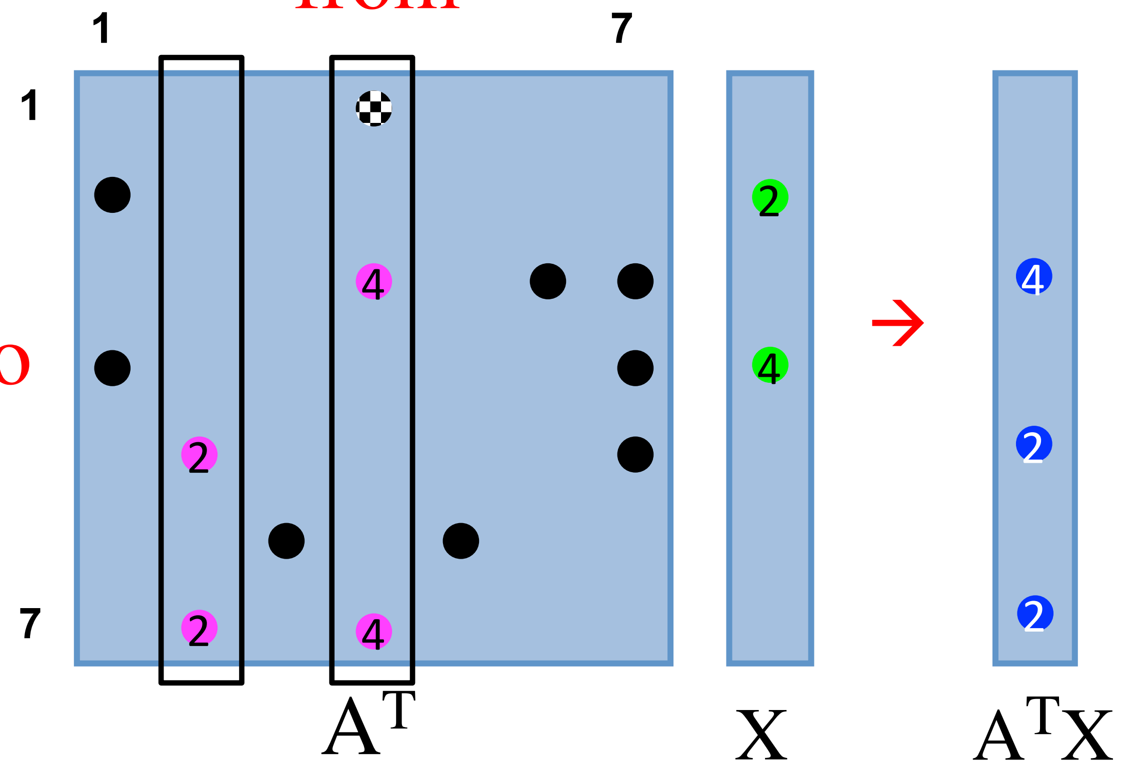
from
1
7
1
to
7
1
1
1
→
A^T
X
A^TX

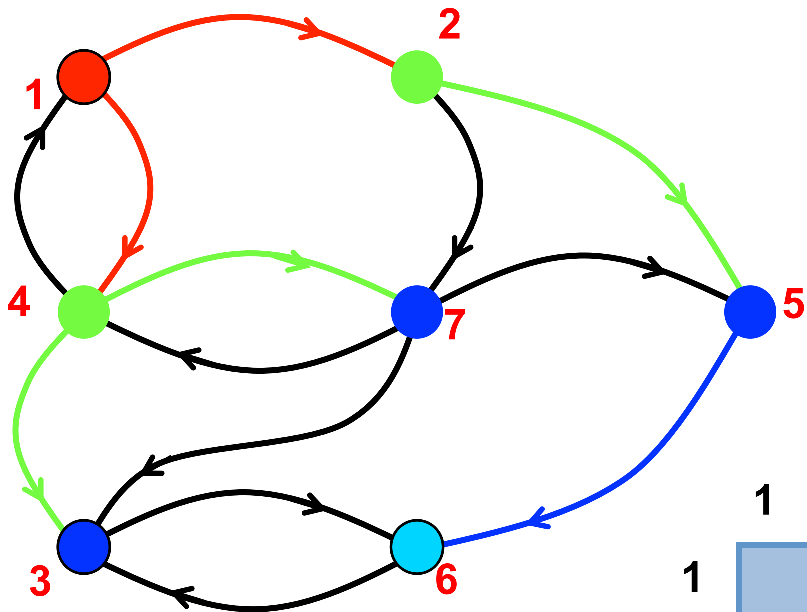


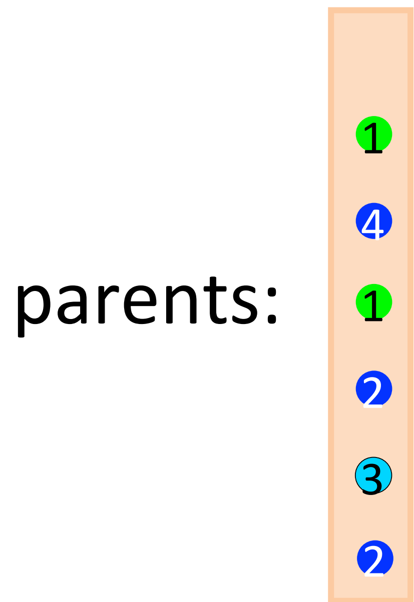
Select vertex with minimum label as parent (no need to keep SPA)

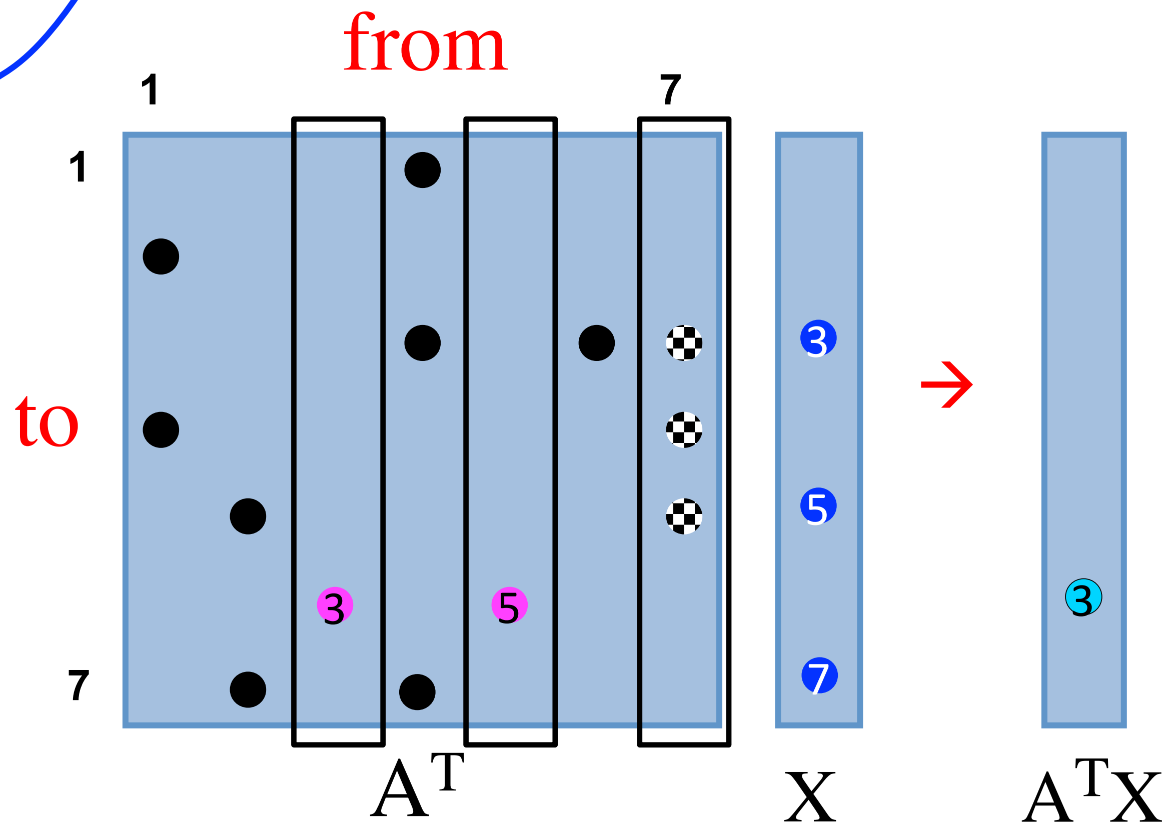
parents:

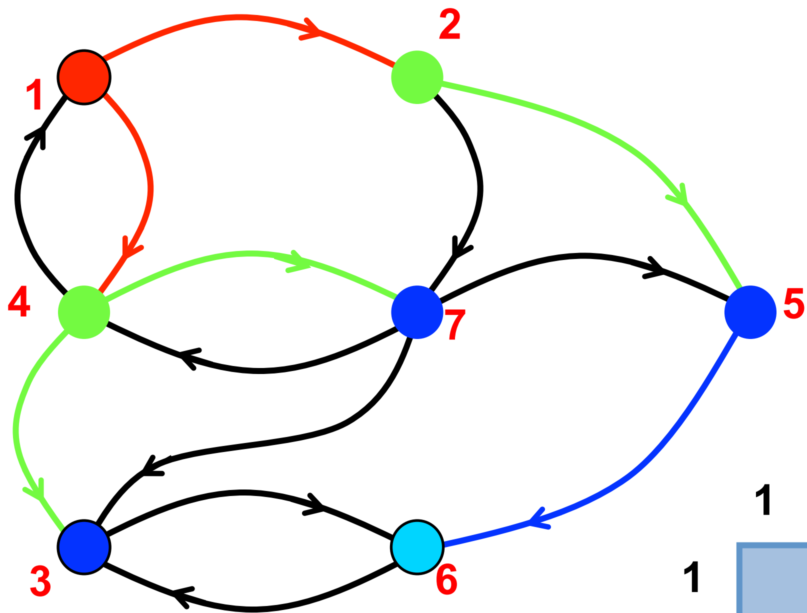


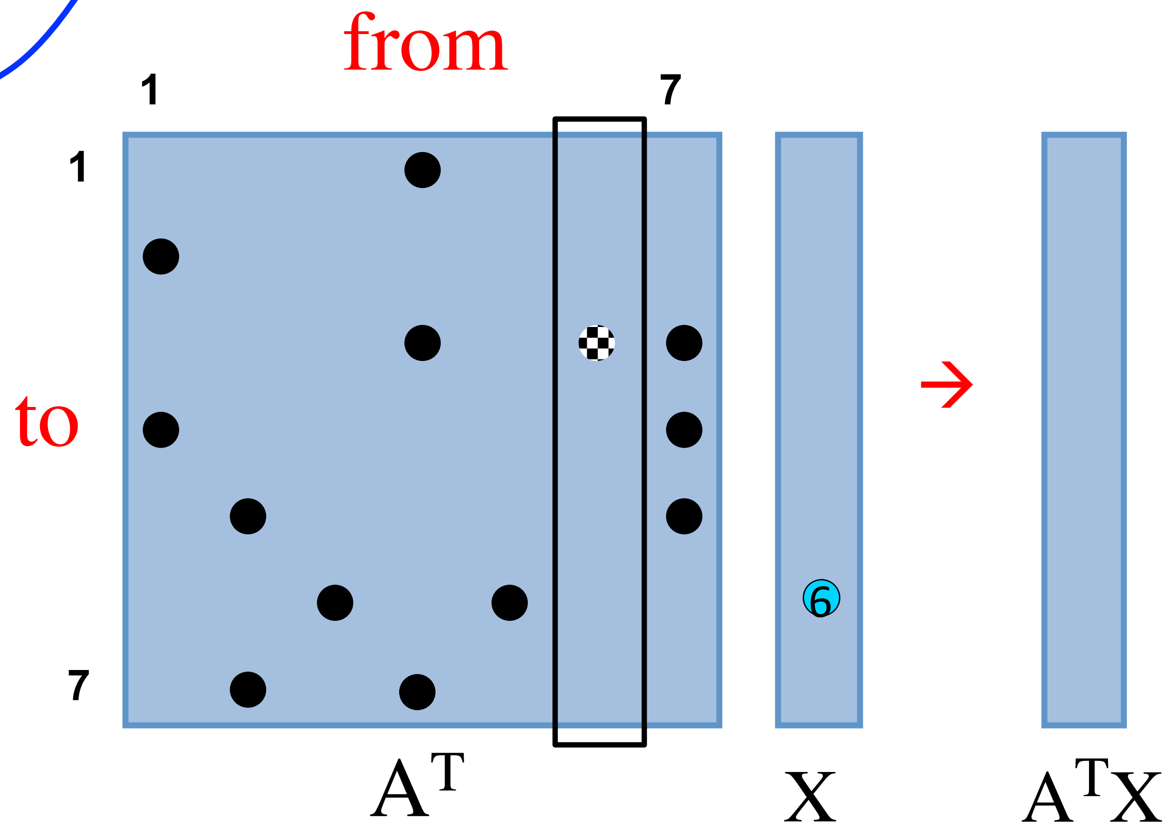



1
2
4
7
5
3
6


parents:
1
4
1
2
3
2


from
to
1
7
1
7
3
5
3
5
7
3
→
A^T
X
A^TX


1
2
4
7
5
3
6


from
1
7
1
7
to
6
→
A^T
X
A^TX