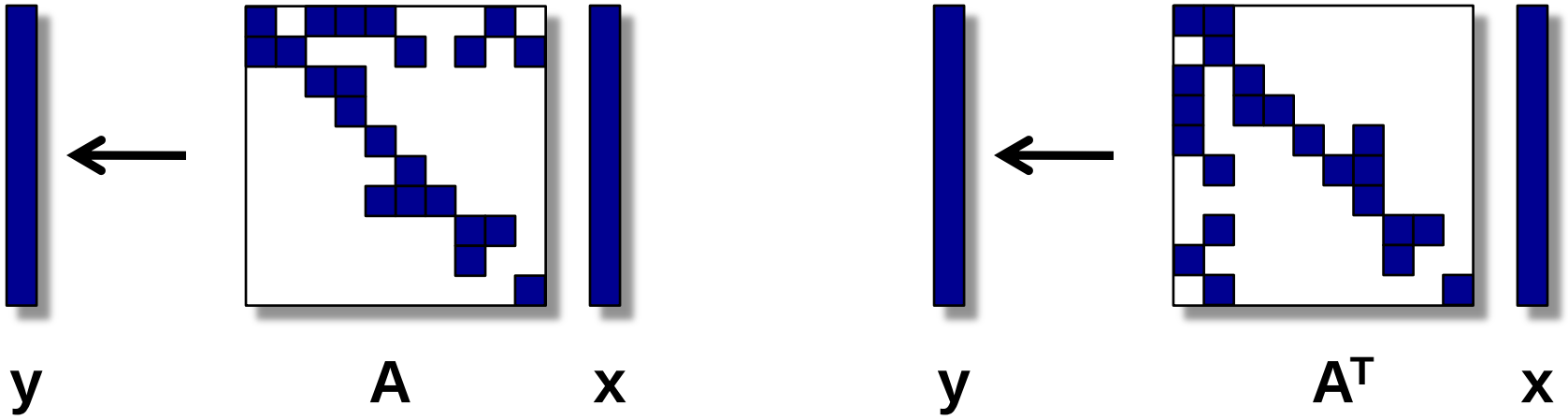



Parallel Sparse Matrix-Vector and Matrix-Transpose-Vector Multiplication using Compressed Sparse Blocks

Aydın Buluç, UCSB
Jeremy T. Fineman (MIT)
Matteo Frigo (Cilk Arts)
John R. Gilbert (UCSB)
Charles E. Leiserson (MIT & Cilk Arts)

Sparse Matrix-Dense Vector Multiplication (SpMV)

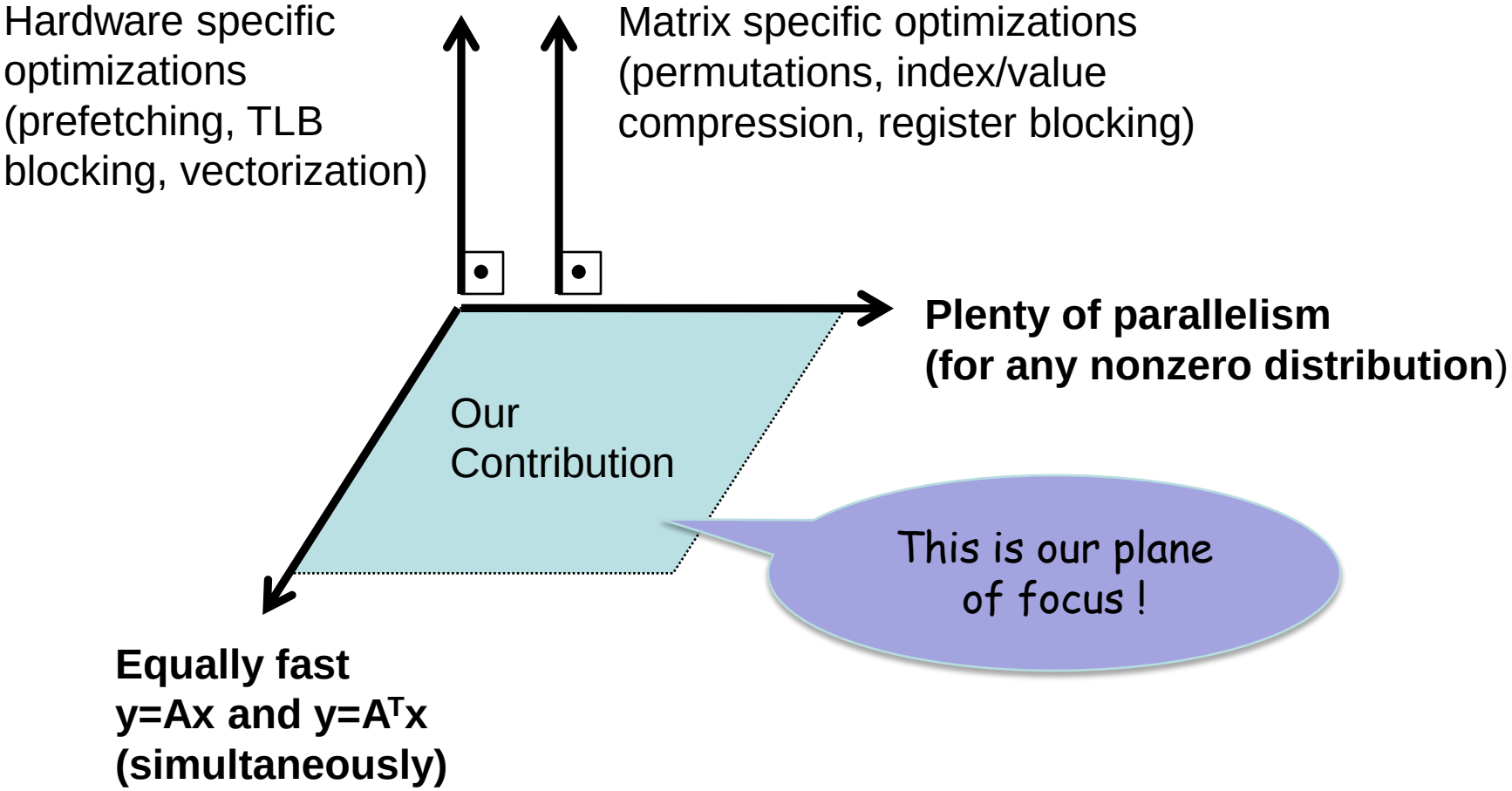


A is an **n-by-n sparse** matrix with **nnz** << n^2 nonzeros

Applications:

- **Iterative methods for solving linear systems**: Krylov subspace methods based on Lanczos biorthogonalization: Biconjugate gradients (BCG) & quasi-minimal residual (QMR)
- Graph analysis: Betweenness centrality computation

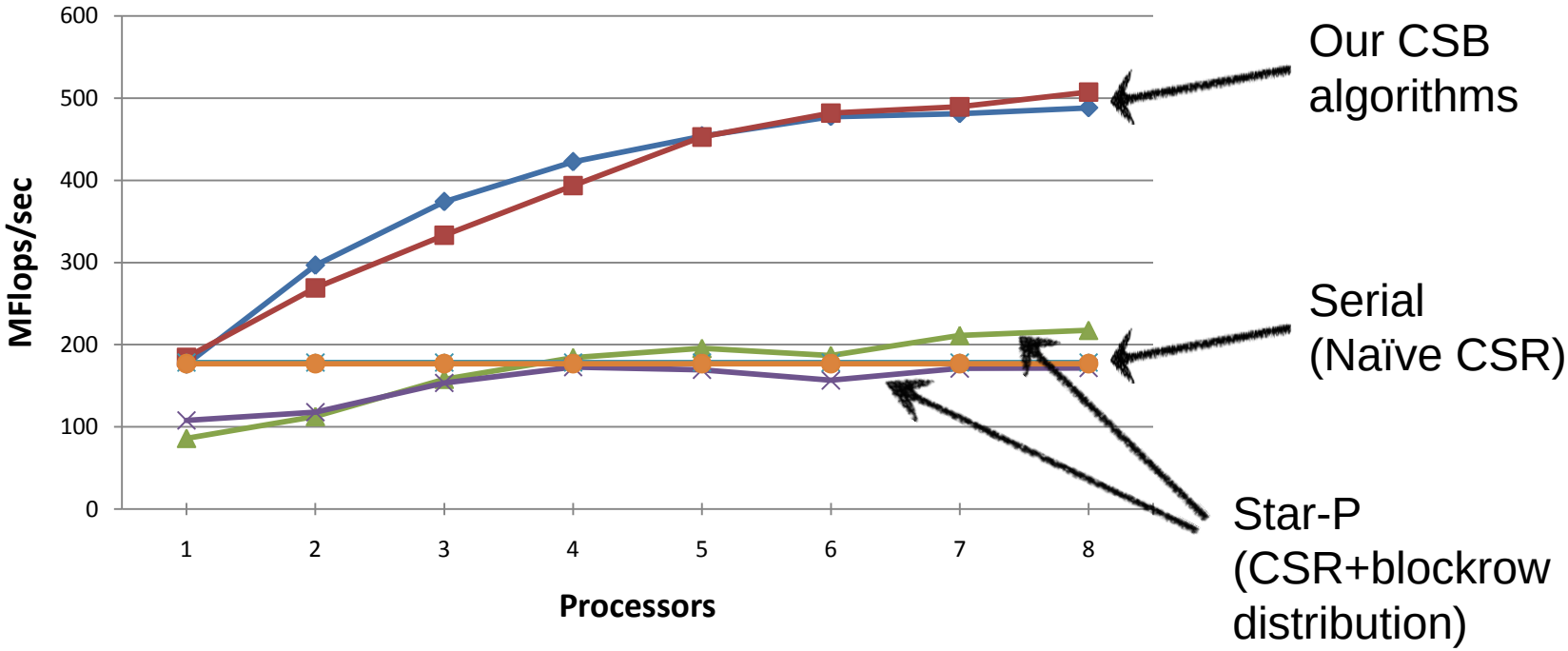
The Landscape: Where does our work fit?



Theoretical and Experimental: Main Results

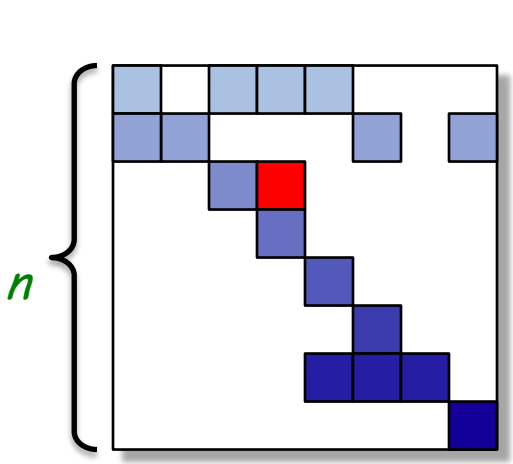
Our parallel algorithms for $y \leftarrow Ax$ and $y \leftarrow A^T x$ using the new ***compressed sparse blocks* (*CSB*)** layout has

- $\Theta(\sqrt{n} \lg n)$ span, and $\Theta(nnz)$ work,
- yielding $\Theta(nnz / \sqrt{n} \lg n)$ parallelism.

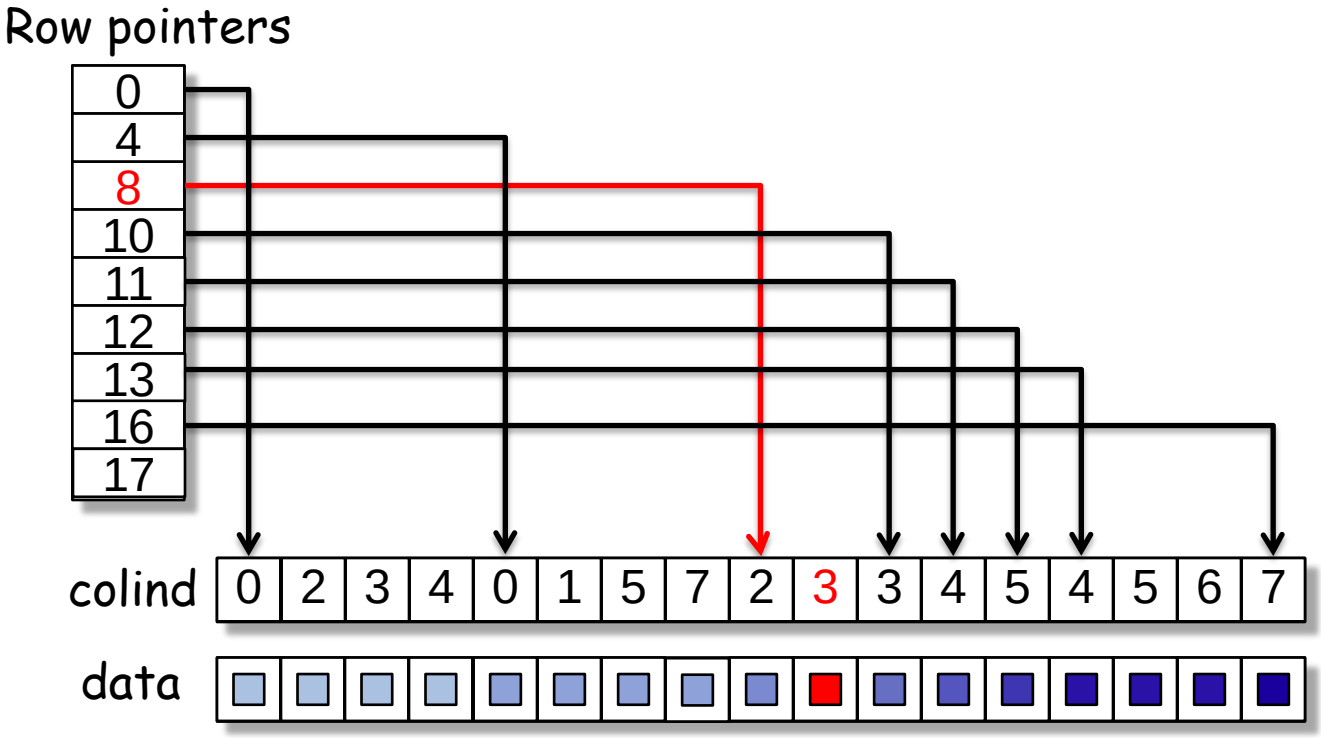


Compressed Sparse Rows (CSR): A Standard Layout

Dense collection of "sparse rows"



$n \times n$ matrix with nnz nonzeroes



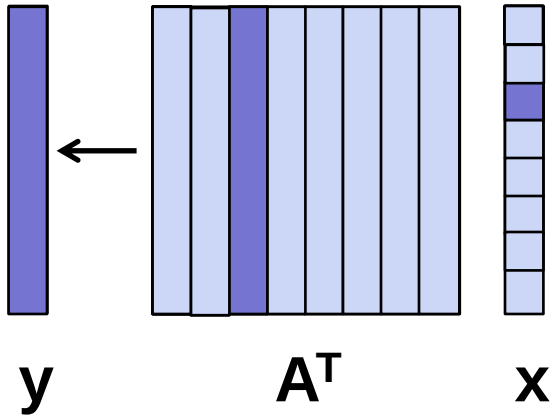
- Stores entries in row-major order
- Uses $n \lg nnz + nnz \lg n$ bits of index data.
- Reading rows in parallel is easy, but columns is hard.

Parallelizing SpMV_T is hard using the standard CSR format

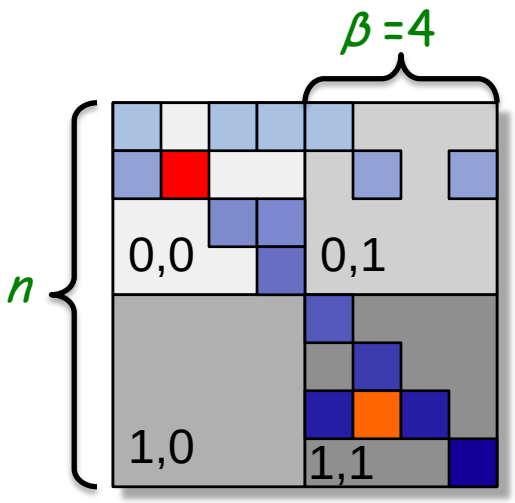
```
CSR_SPMV_T(A,x,y)
for i←0 to n-1
    do for k←A.rowptr[i] to A.rowptr[i+1]-1
       do y[A.colind[k]] ← y[A.colind[k]] + A.data[k]·x[i]
```

1. Parallelize the outer loop?
   - ✕ Race conditions on vector ***y***.
   - a. Locking on ***y*** is not scalable.
   - b. Using ***p*** copies of ***y*** is work-inefficient
2. Parallelize the inner loop?
   - ✕ Span is $\Theta(n)$,

   thus parallelism at most $O(nnz/n)$

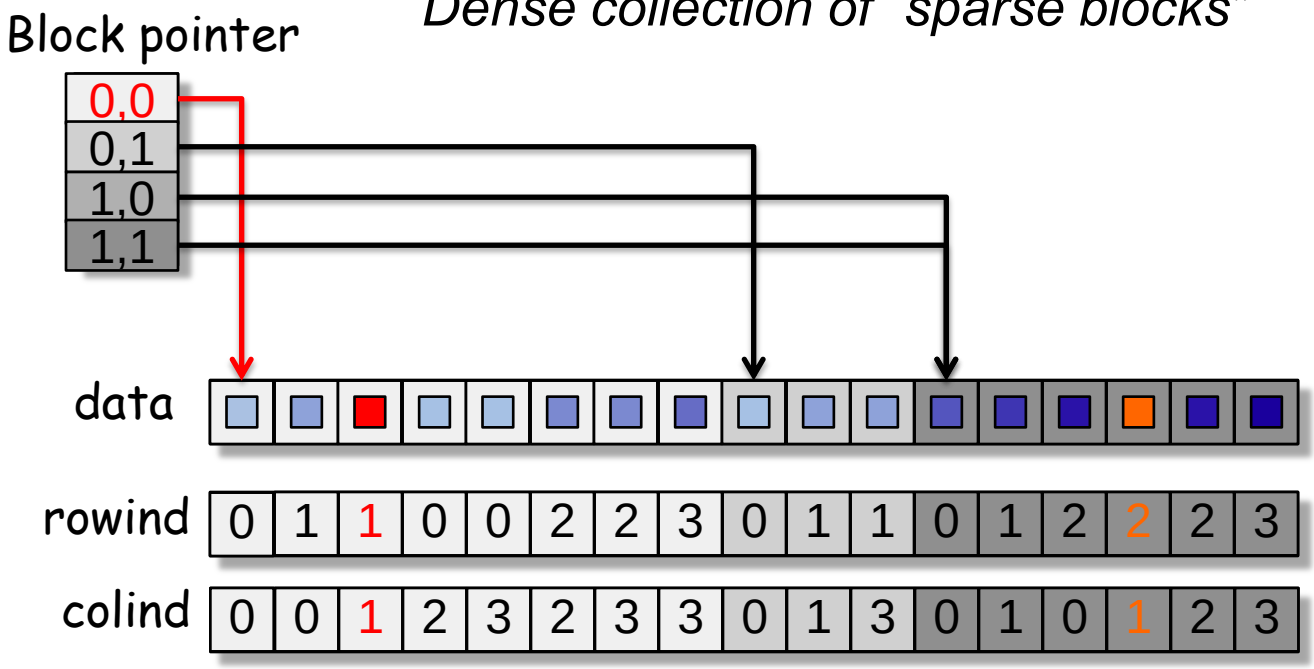


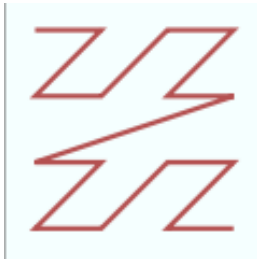
Compressed Sparse Blocks (CSB)



$n \times n$ matrix with nnz nonzeroes, in $\beta \times \beta$ blocks

Dense collection of "sparse blocks"

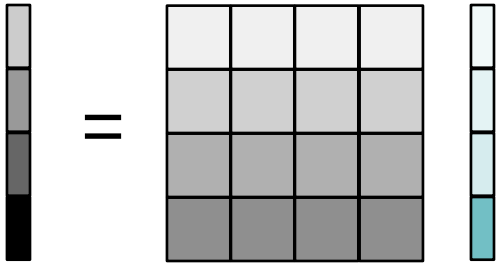




- Store blocks in row-major order (*)
- Store entries within block in Z-morton order (*)
- For $\beta = \sqrt{n}$, matches CSR on storage.

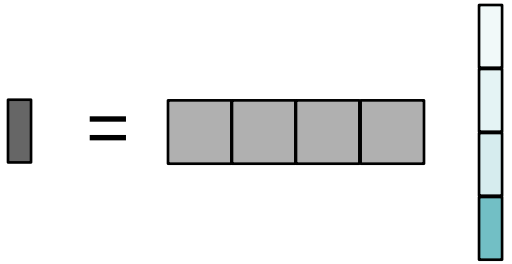
Reading blockrows or blockcolumns in parallel is now easy.

CSB Matrix-Vector Multiplication

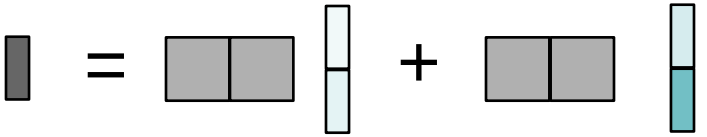


Our algorithm uses three levels of parallelism

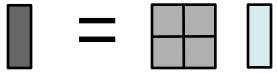
1) Multiply each blockrow in parallel, each writing to a disjoint output subvector.



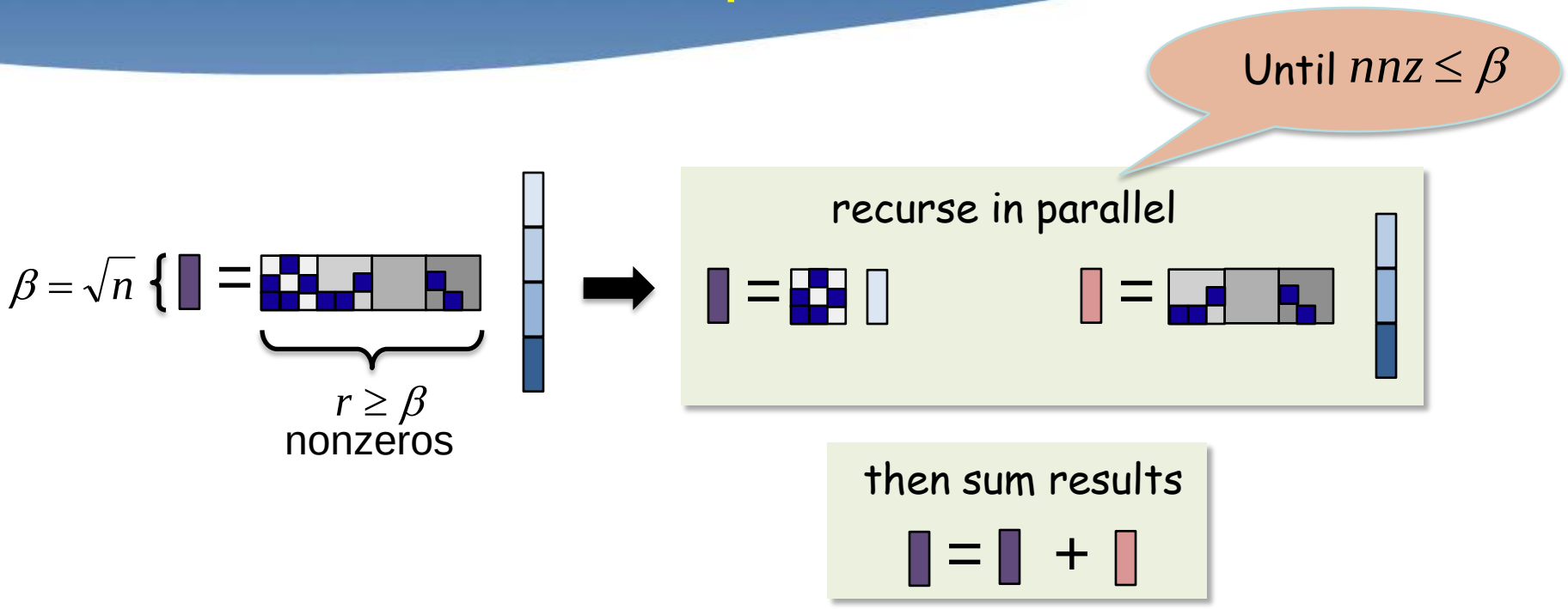
2) If a blockrow is "dense," parallelize the blockrow multiplication.



3) If a single block is dense, parallelize the block multiplication.

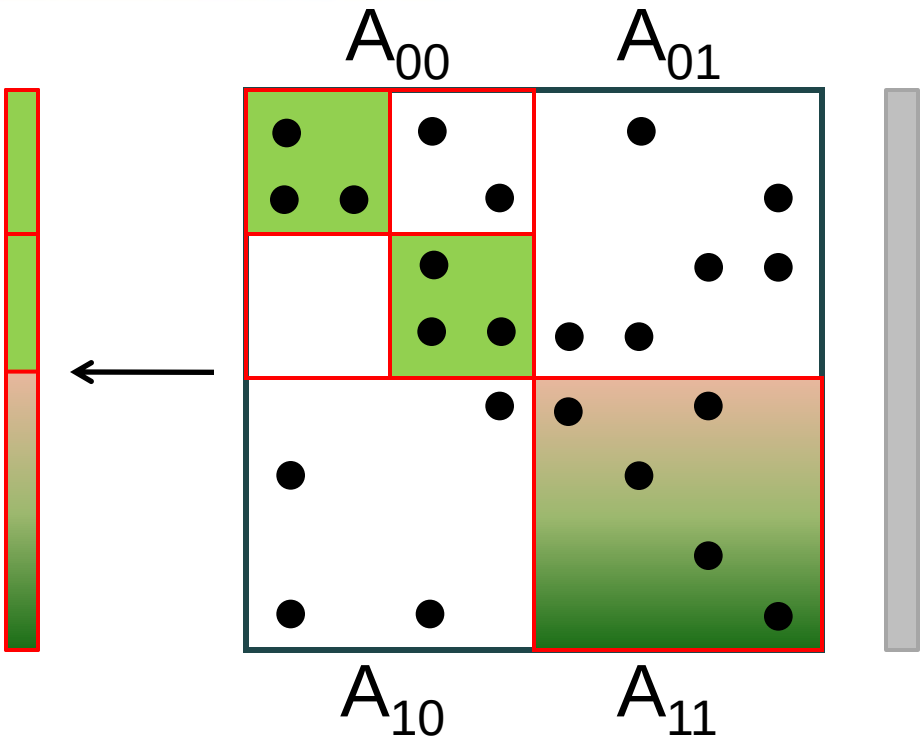


Blockrow-Vector Multiplication



- Divide-and-conquer based on the nonzero count, not spatial.
- Allocation & accumulation costs of temporary vectors are amortized.
- **Lemma:** For $\beta = \sqrt{n}$, our parallel blockrow-vector multiplication has work $\Theta(r)$ and span $O(\sqrt{n}\lg n)$ on a $\beta \times n$ blockrow containing $r \ge \beta$ nonzeroes.

Block-Vector Multiplication

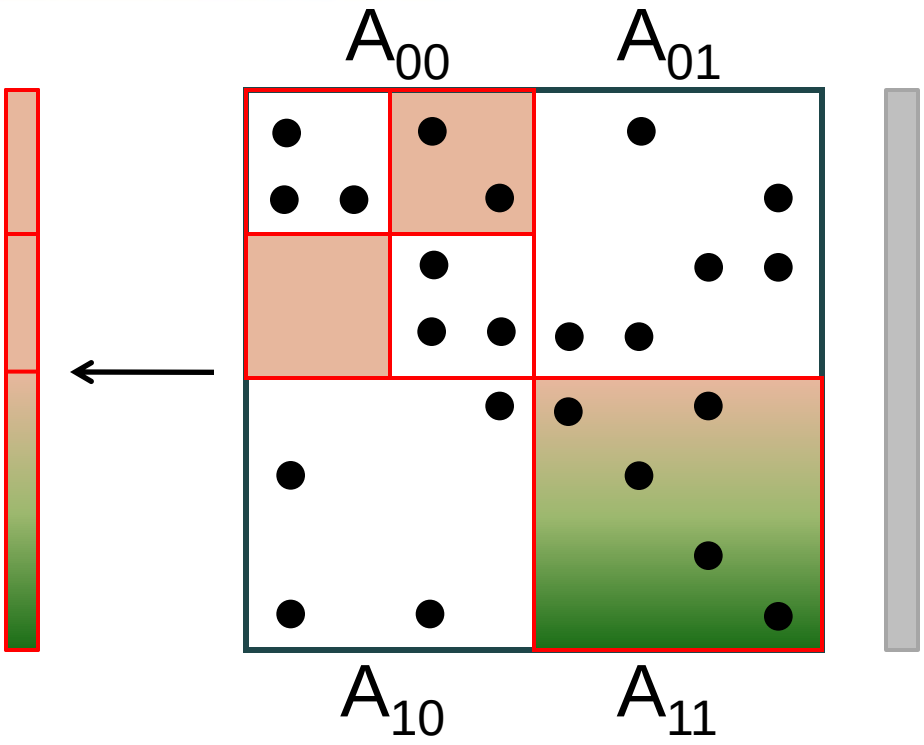


For any (sub)block, first perform A_{00} and A_{11} in parallel; then A_{01} and A_{10} in parallel.

Updates on **y** are **race-free**

- With Z-morton ordering, spatial division to quadrants takes $\lg dim$ time on a $dim \times dim$ (sub)block using three binary searches
- **Lemma:** For $\beta = \sqrt{n}$, our parallel block-vector multiplication has work $\Theta(r)$ and span $O(\sqrt{n})$ on a block with r nonzeroes.

Block-Vector Multiplication

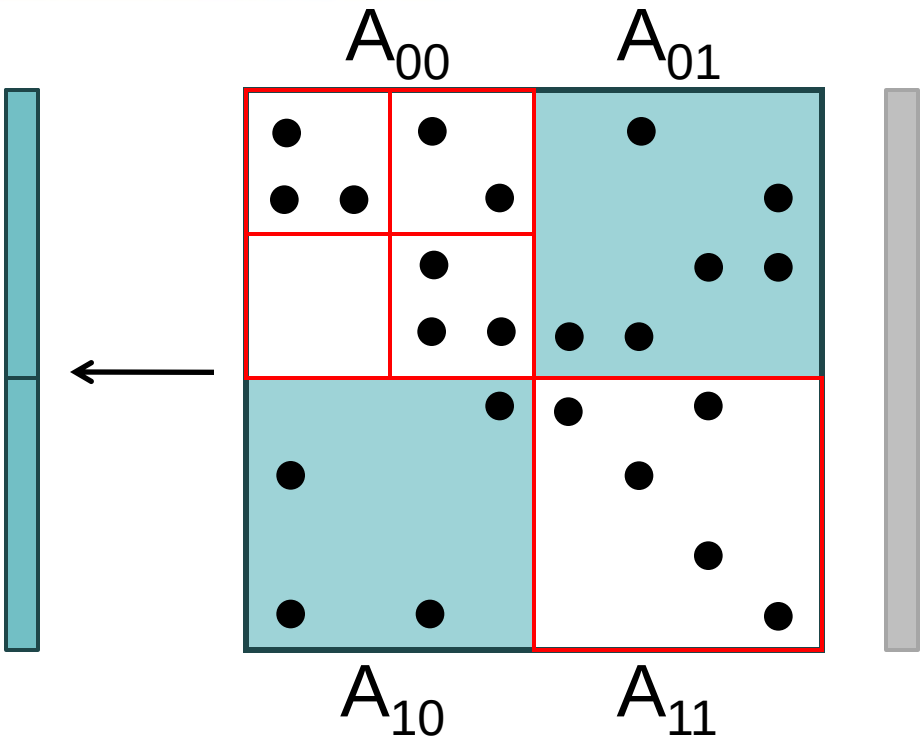


For any (sub)block, first perform A_{00} and A_{11} in parallel; then A_{01} and A_{10} in parallel.

Updates on **y** are **race-free**

- With Z-morton ordering, spatial division to quadrants takes $\lg dim$ time on a $dim \times dim$ (sub)block using three binary searches
- **Lemma:** For $\beta = \sqrt{n}$, our parallel block-vector multiplication has work $\Theta(r)$ and span $O(\sqrt{n})$ on a block with r nonzeroes.

Block-Vector Multiplication



For any (sub)block, first perform A_{00} and A_{11} in parallel; then A_{01} and A_{10} in parallel.

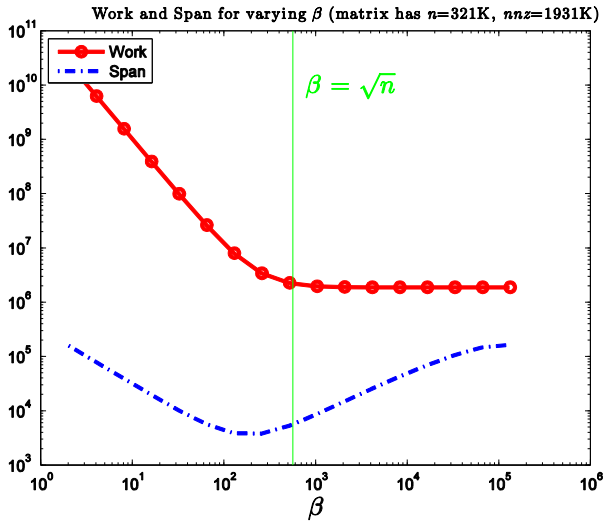
Updates on **y** are **race-free**

- With Z-morton ordering, spatial division to quadrants takes $\lg dim$ time on a $dim \times dim$ (sub)block using three binary searches.
- **Lemma:** For $\beta = \sqrt{n}$, our parallel block-vector multiplication has work $\Theta(r)$ and span $O(\sqrt{n})$ on a block with r nonzeroes.

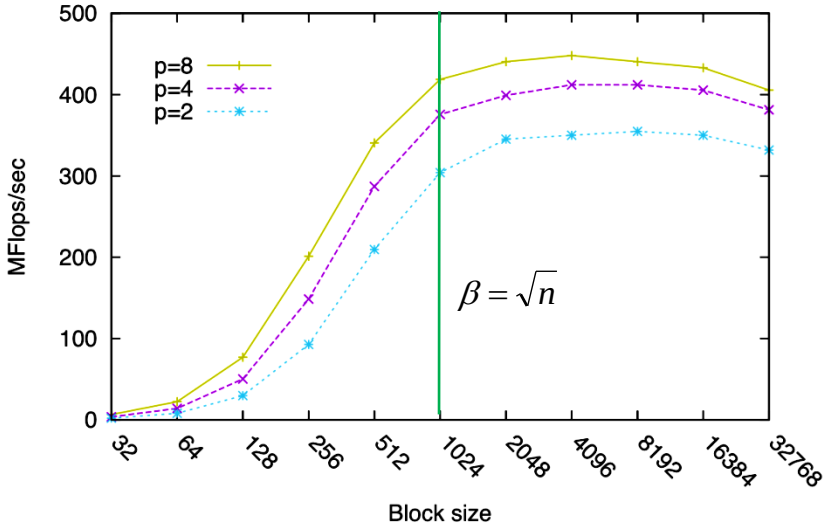
Main Theorem and The Choice of β

Theorem: Our parallel matrix-vector multiplication has work $O(n^2/\beta^2 + nnz)$ and span $O(\beta \lg(n/\beta) + n/\beta)$, yielding on an $n \times n$ CSB matrix containing $nnz \geq n$ nonzeroes.

For $\beta = \sqrt{n}$,this yields a parallelism of $\Theta(nnz/\sqrt{n}\lg n)$
On our test matrices, parallelism ranges from 186 to 3498

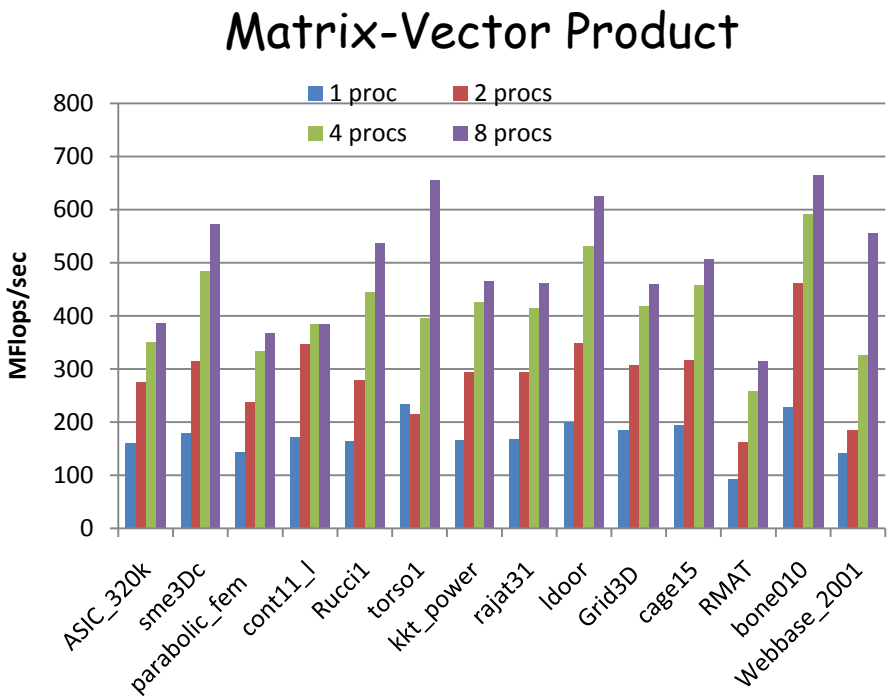


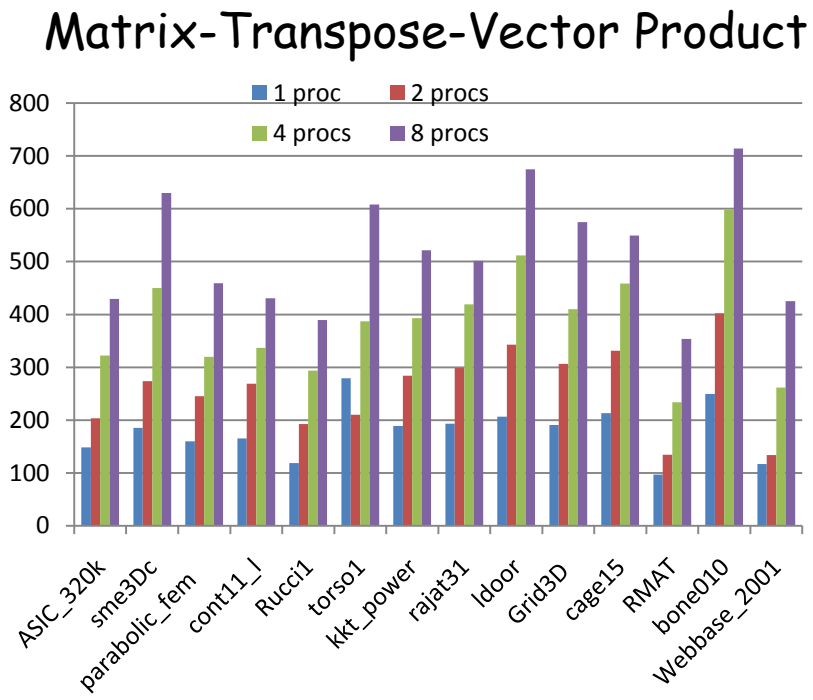
Sensitivity to β in theory



Sensitivity to β in practice

$\mathbf{Ax}$ and $\mathbf{A^Tx}$ perform equally well

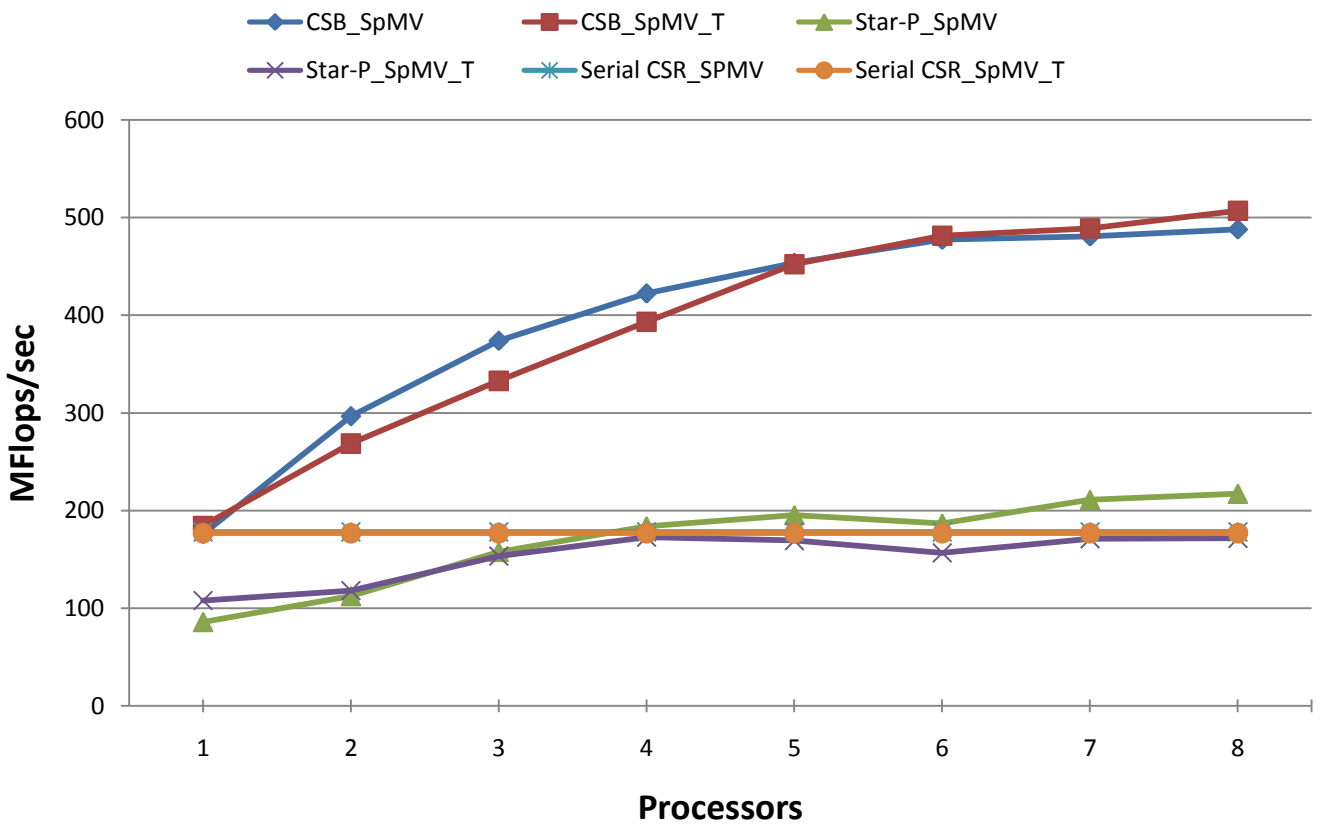


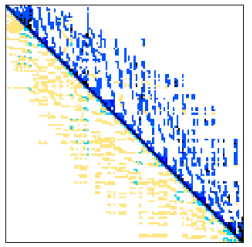


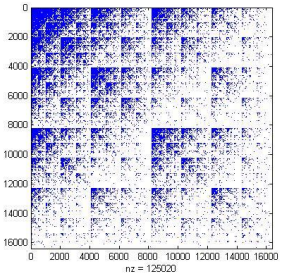
4-socket dual-core 2.2GHz AMD Opteron 8214

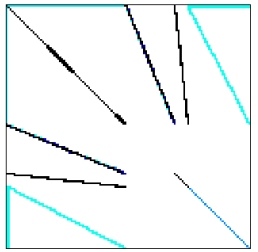
Most test matrices had ***similar performance*** when multiplying by the matrix and its transpose.

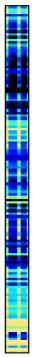
Test Matrices and Performance Overview

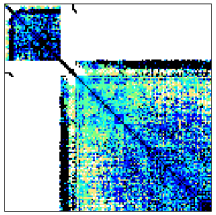


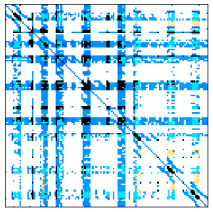


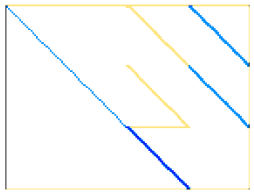


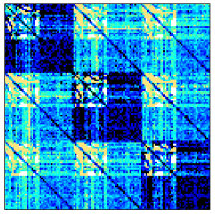


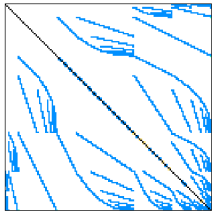


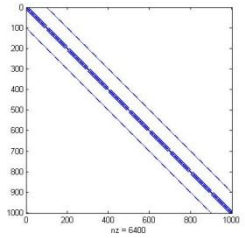












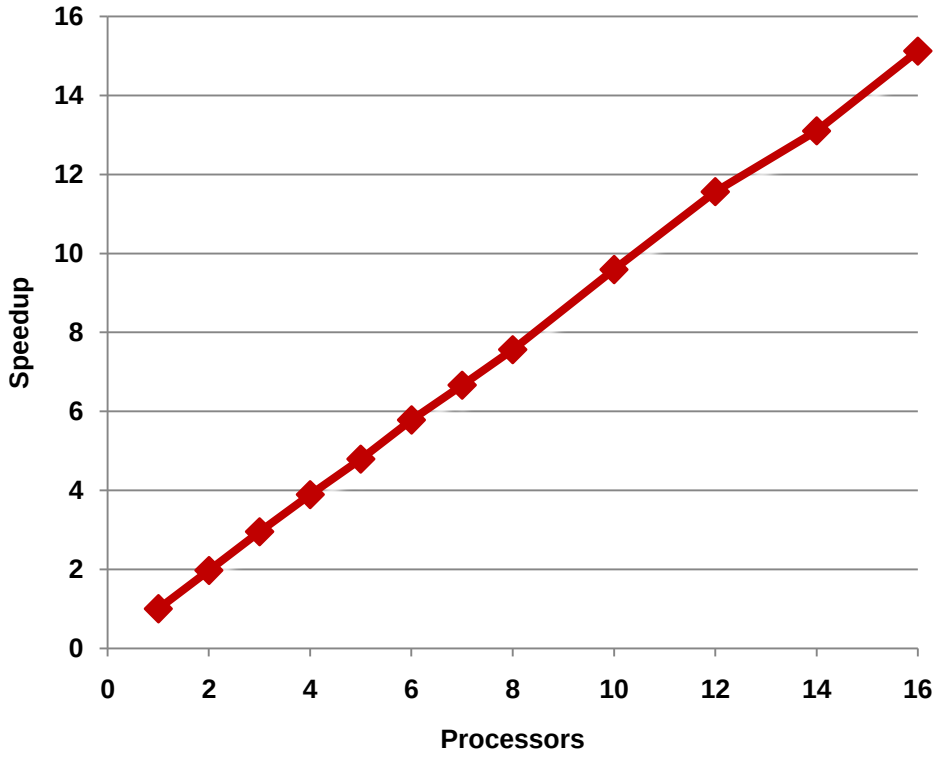
Reality Check and Related Work

FACT: Sparse matrix-dense vector multiplication (and the transpose) is ***bandwidth limited***.

This work: motivated by ***multicore/manycore*** architectures where parallelism and memory bandwidth are key resources.

- Previous work mostly focused on reducing communication volume in distributed memory, often by using graph or hypergraph partitioning [Catalyurek & Aykanat '99].
- Great optimization work for SpMV on multicores by Williams, et al.'09, but without parallelism guarantees or multiplication with the transpose (SpMV_T).
- Blocking for sparse matrices is not new, but mostly for cache performance, not parallelism [Im et al.'04, Nishtala et al. '07].

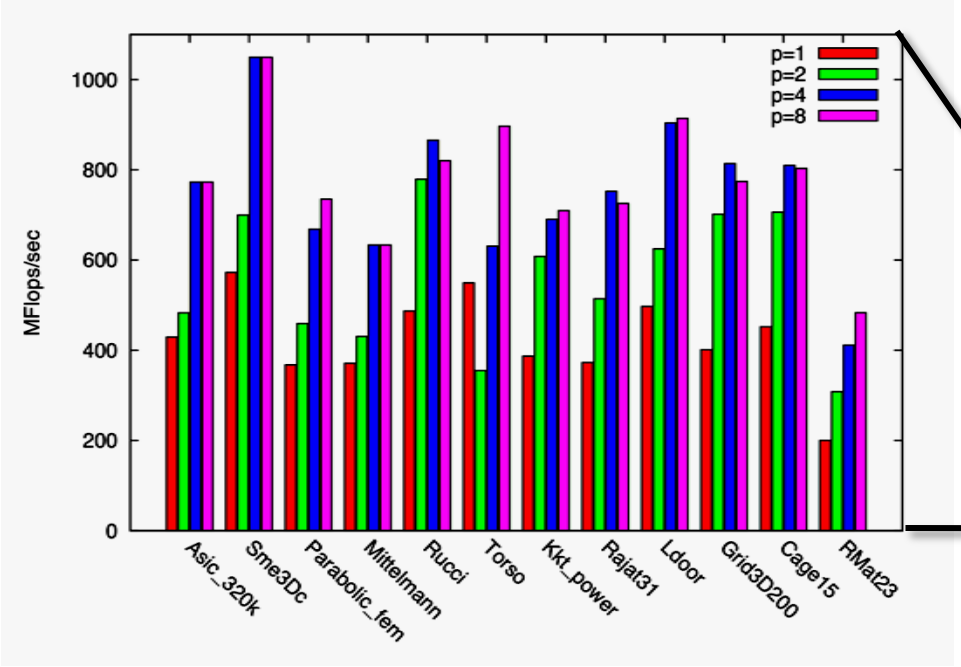
Good Speedup Until Bandwidth Limit

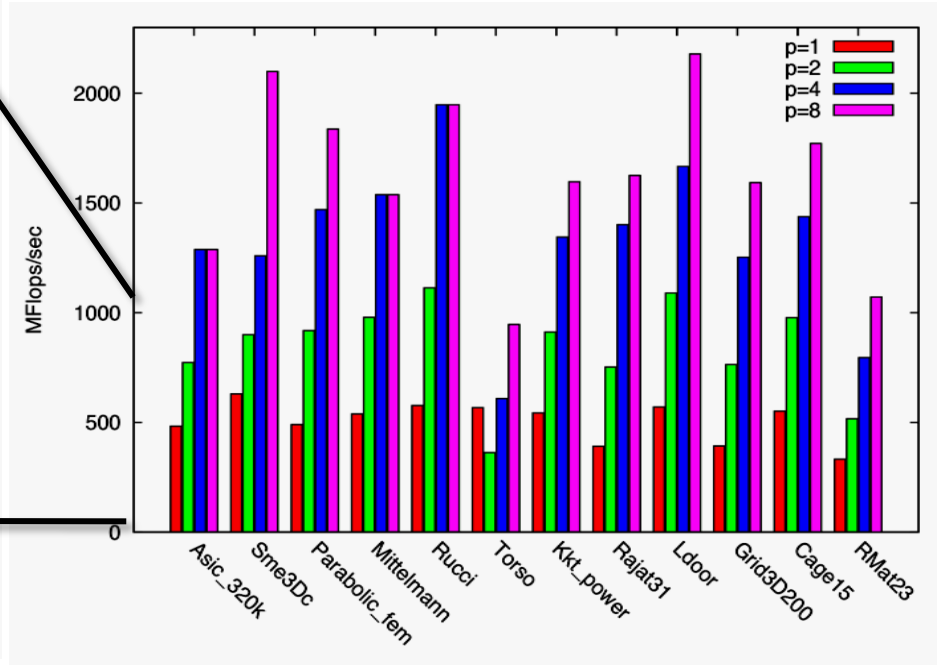


- Slowed down processors (artificially introduced extra instructions) for test to hide memory issues.
- Shows algorithm scales well given sufficient memory bandwidth.

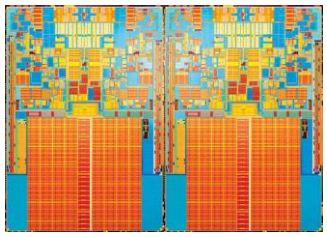
Ran on the smallest (and one of the most irregular) test matrix

All about Bandwidth: Harpertown vs Nehalem

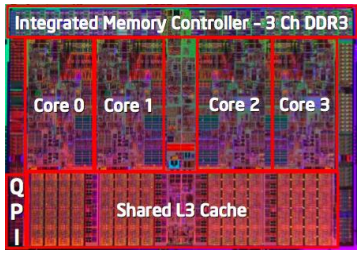




Intel Xeon X5460 @3.16Ghz
Dual-socket Quad-core
FSB @1333Mhz



Intel Core i7 920 @2.66Ghz
Single-socket Quad-core
Quickpath+Hyperthreading



Conclusions & Future Work

- CSB allows for efficient multiplication of a sparse matrix *and its transpose* by a dense vector.

Future Work:

- Does CSB work well with other computations? Sparse LU decomposition? Sparse matrix-matrix multiplication?
- For a symmetric matrix, need only store upper triangle of matrix. Can we multiply with one read (i.e., ½ the bandwidth)?

Code (in C++ and Cilk++) available from:

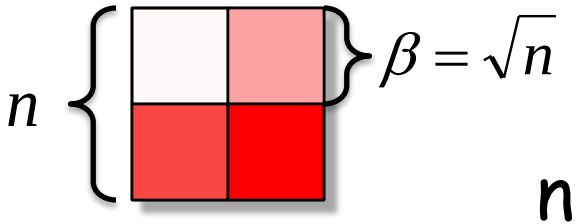
http://gauss.cs.ucsb.edu/~aydin/software.html

Thank You !

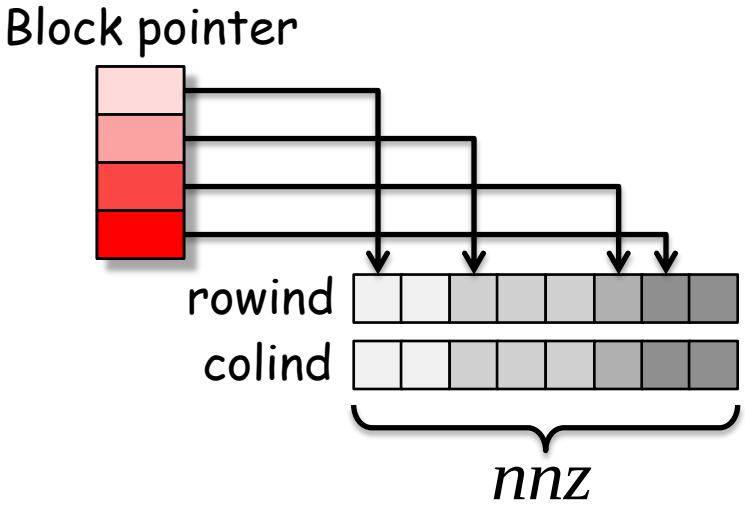
CSB Space Usage

Lemma: For $\beta = \sqrt{n}$, CSB uses $n \lg nnz + nnz \lg n$ bits of index data, matching CSR.

Proof:



n blocks => n block pointers.



- Each block pointer uses $\lg nnz$ bits, for $n \lg nnz$ total
- Each row (or column) offset requires $\lg \beta = \lg \sqrt{n}$ bits, for $(nnz/2) \lg n$ total.