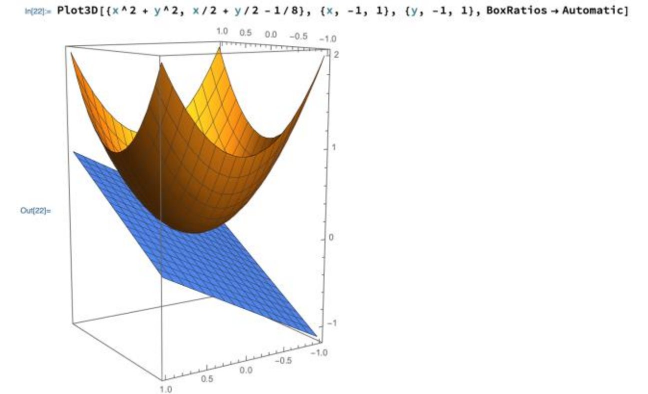
Contiguous Partitioning: Registers, Caches, and Distributed Memories

Peter Ahrens

Matrix-Vector Multiply And Iterative Solvers

- Everything is linear to a first approximation
- Goal: Solve $Ax = b$
- A is an $m \times n$ matrix with N nonzeros
- Iterative methods work to improve an initial guess
  - Multiply by A many times to find the error in each step

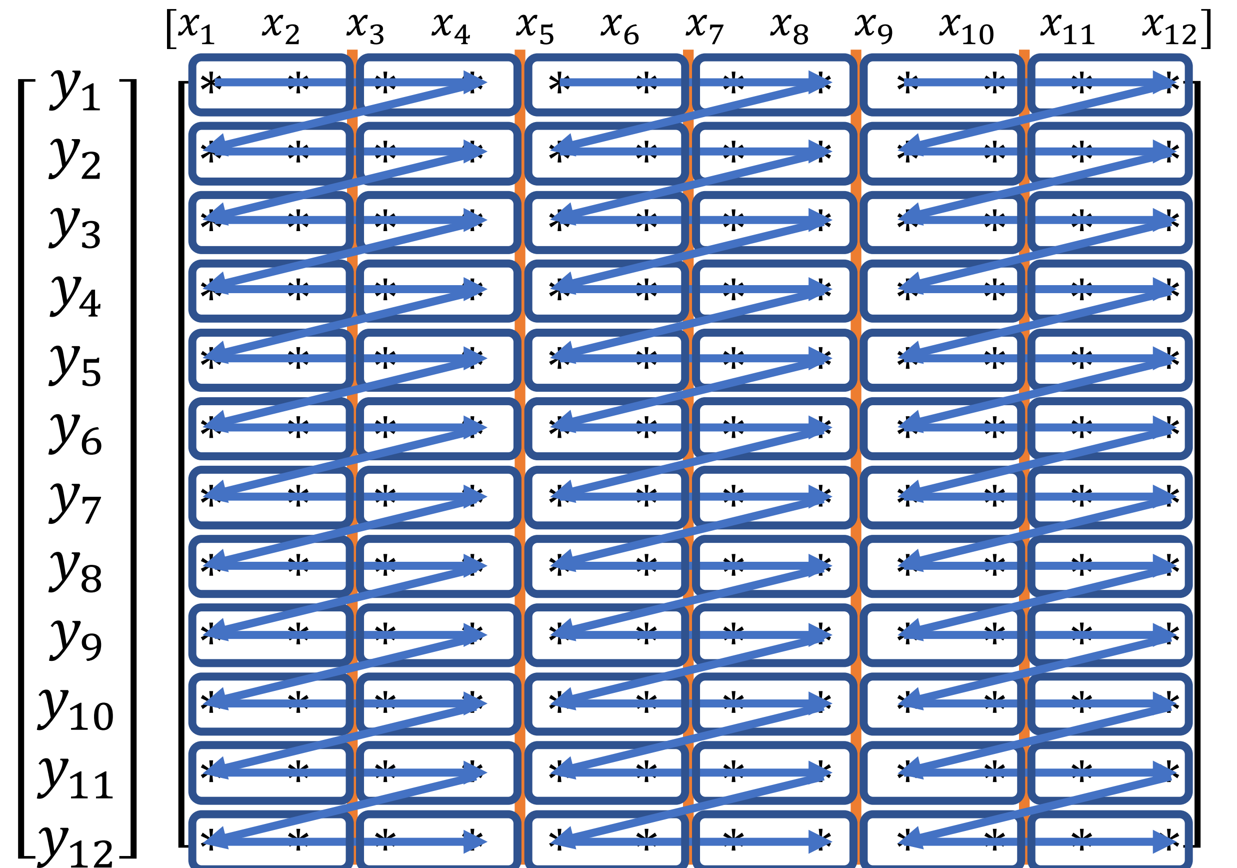


$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} \boldsymbol{a_{11}} x_1 + a_{12} x_2 + a_{13} x_3 \\ a_{21} x_1 + \boldsymbol{a_{22}} x_2 + a_{23} x_3 \\ a_{31} x_1 + a_{32} x_2 + \boldsymbol{a_{33}} x_3 \end{bmatrix}$$

$m, n < N$

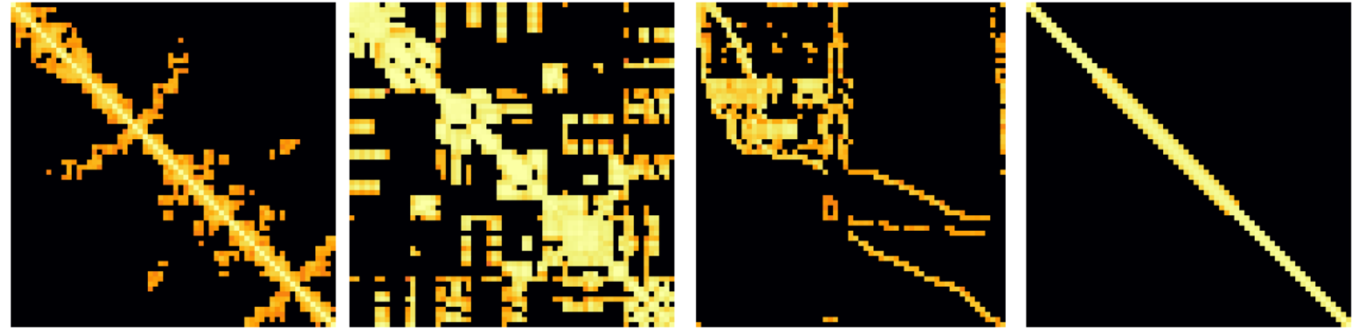
Dense Matrix-Vector Multiply

- On NERSC Cori KNL Supercomputer
  - Theoretical Peak Compute Throughput: 1.5TB/s
  - Memory Bandwidth: 101GB/s
  - Measured MPI Bandwidth (NERSC Cori): 11GB/s
- Register Blocking
- Cache Blocking
- Distributed Memory



Sparsity

- Sparsity is beneficial
  - Potential to save work
  - Potential to save messages
- Sparsity is a challenge!
  - Memory cost
  - Computational cost
  - Communication structure



$$[x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad x_7 \quad x_8 \quad x_9 \quad x_{10} \quad x_{11}]$$

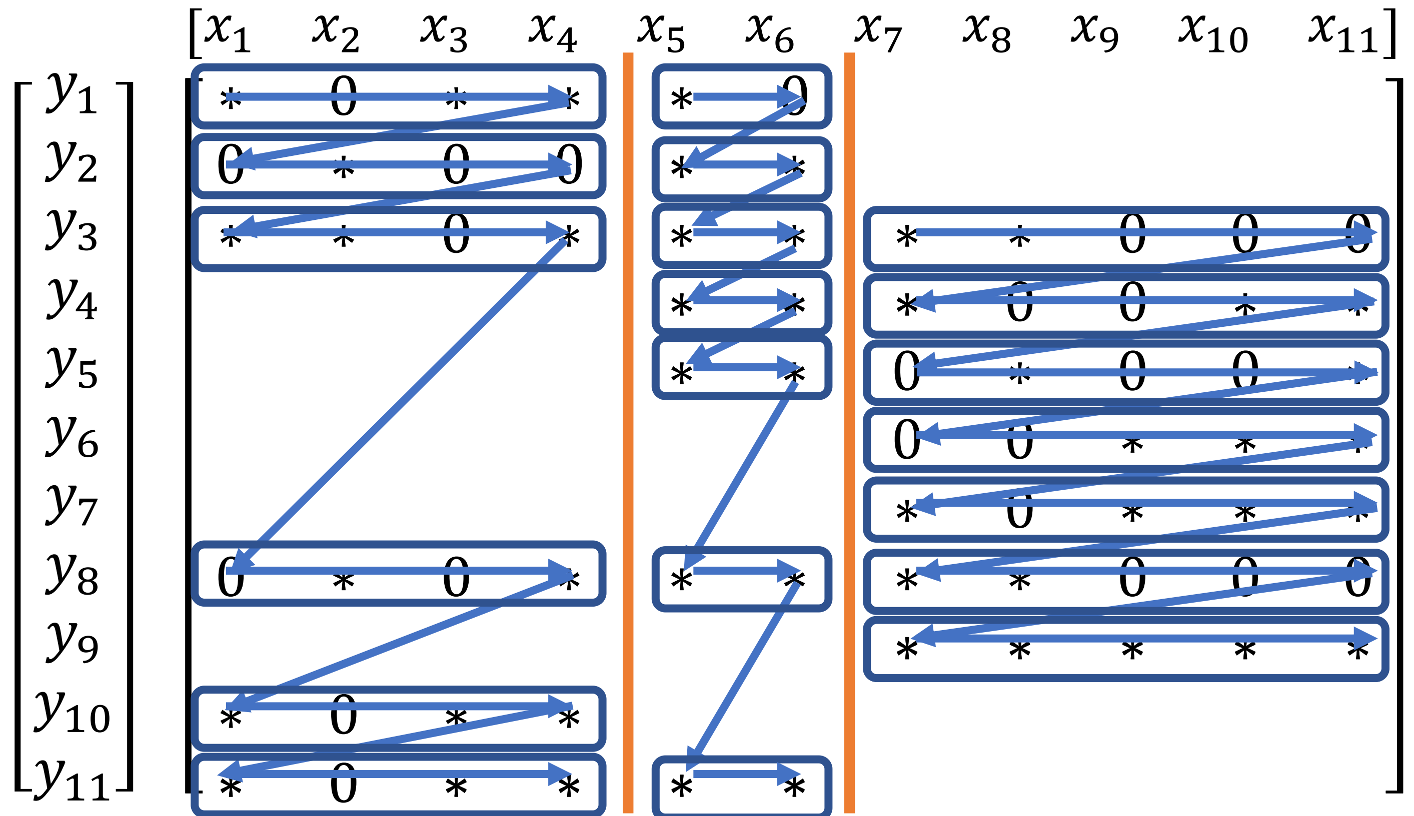
$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \\ y_8 \\ y_9 \\ y_{10} \\ y_{11} \end{bmatrix} \begin{bmatrix} * & & * & * & * & & & & & & \\ & * & & & * & * & & & & & \\ * & * & & * & * & * & * & * & & & \\ & & & & * & * & * & & & * & * \\ & & & & * & * & & * & & & * \\ & & & & & & & & * & * & * \\ & & & & & & * & & * & * & * \\ & * & & * & * & * & * & * & & & \\ & & & & & & * & * & * & * & * \\ * & & * & * & & & & & & & \\ * & & * & * & * & * & & & & & \end{bmatrix}$$

Partitioning

Split the columns into K disjoint parts π_k

$$\Pi = [\{1,2,3,4\},\{5,6\},\{7,8,9,10,11\}]$$

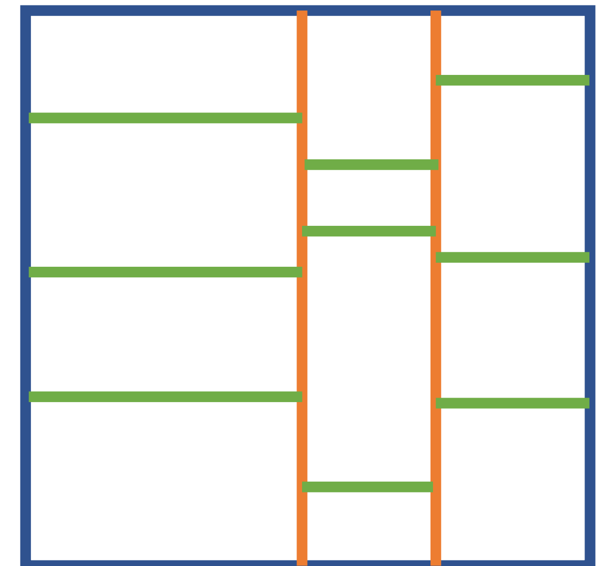
- Register Blocking
  - Fill in zeros, store the location of each block once
- Cache Blocking
  - Don't fill, only load rows once per part
  - Column values stay in cache because part sizes bounded
- Communication Minimization
  - One part per processor
  - Only communicate rows once per part



Contiguity (No Reordering!)

- Natural ordering might be good
- Ordering might be constrained
  - Numerical concerns (direct solvers)
  - DAG scheduling (data dependence constraints, topological ordering)
- We can make the ordering good
  - Natural embedding
    - Physical simulation space
  - Induced embedding
    - Spectral / Cuthill McKee order
  - Multidimensional -> One dimensional
    - Space-filling Curves (Hilbert or Travelling Salesman)
    - Geometric partitioners (Multi-Jagged or Recursive Coordinate)
- Noncontiguous partitioning is expensive (NP Hard)
  - Optimizers compete with quality of their optimizations

Multi-Jagged Decomposition



Register Blocking: Variable Size

- Variable Block Row Format
- Added flexibility
  - Supernodal Factorizations, Circuit Simulations, Linear Programming
- Alignment is useful
  - Recursive linear algebra
    - Reuse
    - Coiteration

Φ

$$\Pi \begin{bmatrix} & & x & x & x & 0 & & & x \\ x & x & & x & x & x & & x & \\ x & 0 & & 0 & 0 & x & & x & \\ x & x & & x & x & x & & x & \\ & & x & & & & & x & \\ & & x & & & & & x & \\ 0 & x & & 0 & x & x & x & & x \\ x & x & & 0 & x & x & x & & x \end{bmatrix}$$

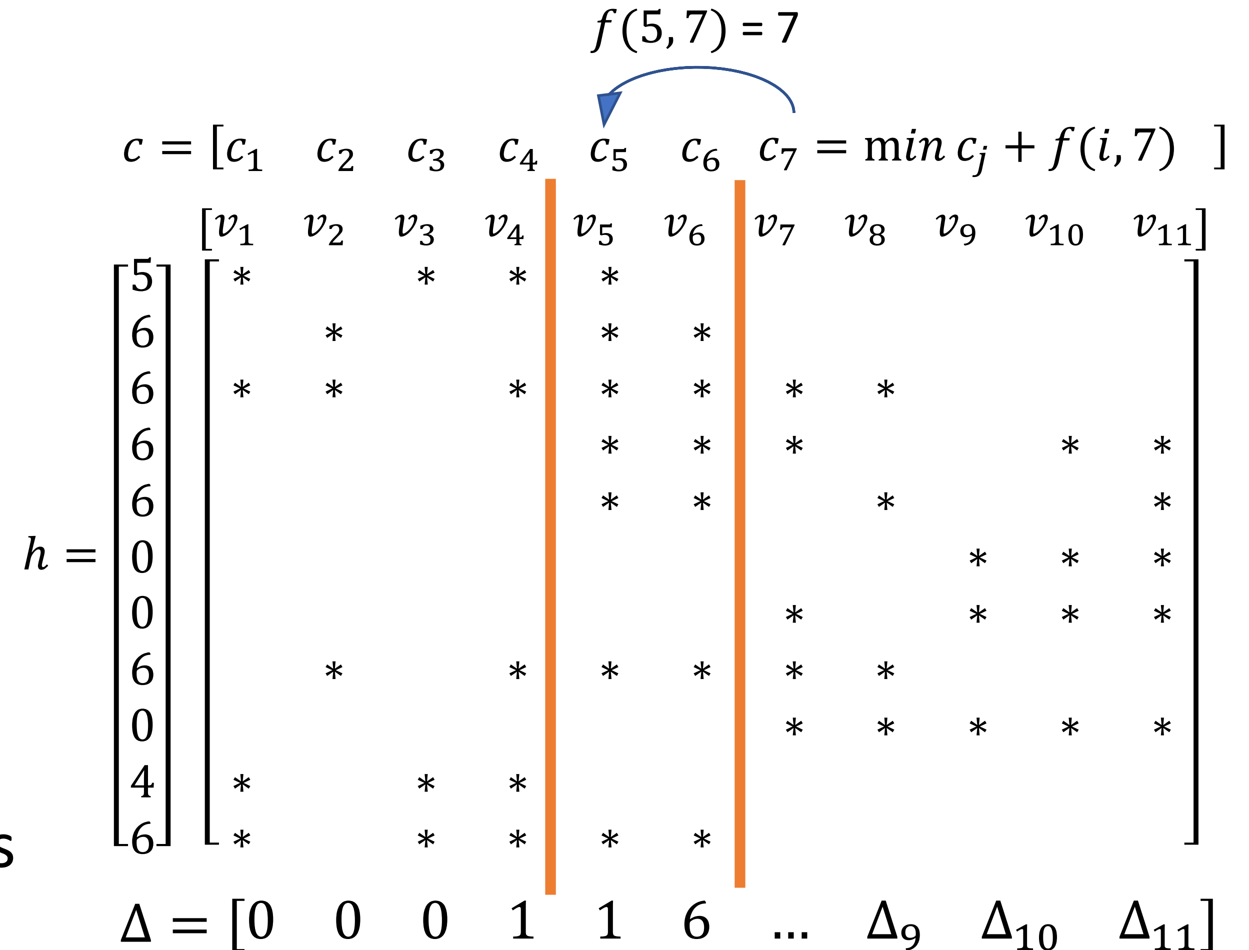
Register Blocking: Variable Size

- # of blocks * cost of block
- Cost per block measured experimentally for all sizes
- # of blocks depends on pattern and partition
- If cost is # of blocks or total memory, problem is NP-Hard
  - Apply heuristic to rows only, then columns only, then rows only, etc…
- Prior heuristics greedily merge similar adjacent columns with at least θ pattern similarity
- We propose optimal dynamic programming to calculate best splits

P. Ahrens and E. Boman, 2020

Optimal Dynamic Programming Algorithm

- $c_{j'} = \min_j c_j + f(j, j')$
  - where $j' - w_{\max} \leq j < j'$
- $f(j, j') = |v_j \cup \cdots \cup v_{j'}|$
- Use hashing tricks to calculate f
- $O(w_{\max} \cdot n + N)$
  - Linear time
  - Single pass
  - $w_{\max}$ is small, so it's okay
- 2.2x mean speedup over unblocked
- Justified optimization within 16 SpMVs



P. Ahrens and E. Boman, 2020

What about caches and networks?

- Blocks get big
  - Dynamic programming becomes $O(n^2)$ quadratic time!
- Use convexity in the cost function!
  - $O(n \log(n))$ queries for Least-Weight-Subsequence algorithms
  - Dominance counting to support random cost queries
  - Modify LWS algorithms for weight constraints
- Applies to
  - Classical graph partitioning (simple edge cut)
  - Hyperedge cut
  - Hypergraph connectivity

Total Partitioning

Blocking Problem (Variable K)

$$c_{j'} = \min_{j} c_j + f(j, j')$$

Prior Approach

$$O(w_{\max} \cdot n + N)$$

- $w_{\max}$ is big

Our Approach

$$O(n \log(n) \log(N) + N \log N)$$

- Crossover at $w_{max} > 1000$

Partitioning Problem (Fixed K)

$$c_{j',k+1} = \min_{j} c_{j,k} + f(j, j')$$

Prior Approach

$$O(K\ ((1+\epsilon)n^2/K + N))$$

- w_{max} is $(1+\epsilon)n/K$

Our Approach

$$O(Kn \log(n) \log(N) + N \log N)$$

- Speedup of $53 \times$ (Average of $15 \times$)

B. W. Kernighan, 1971

A. Grandjean, J. Langguth, and B. Uçar, 2012

Bottleneck partitioning

- When running in parallel, we care about the longest processor
- Cost is work + communication
- Decision problem: Is there a partition with max cost at most c?
  - Binary search over space of costs
- Linear time
  - Average of 5 SpMVs
- Often achieves 2x quality improvements