Parallel Implementation of BLAS: General Techniques for Level 3 BLAS

Almadena Chtchelkanova
John Gunnels
Greg Morrow
James Overfelt
Robert A. van de Geijn

Department of Computer Sciences
and
Texas Institute for Computational and Applied Mathematics

November 13, 1995

Acknowledgements

- Recent related projects with the following individuals have had substantial impact on this work: Carter Edwards, Po Geng, Ken Klimkowski, James Overfelt, Abani Patra, and Jaehoon Seol.
- Considerable impact have also come from collaborations with Jack Dongarra (Univ. of TN, ORNL), Satya Gupta (Intel SSD), John Lewis (Boeing), Greg Henry (Intel SSD), David Payne (Intel SSD), Rob Schreiber (RIACS), Lance Shuler (Sandia), David Walker (ORNL), and Jerrell Watts (Caltech).
- This research was performed in part using the Intel Paragon System operated by the California Institute of Technology on behalf of the Concurrent Supercomputing Consortium. Access to this facility was provided by Intel Supercomputer Systems Division and the California Institute of Technology.

Funding was provided in part by

- The NASA High Performance Computing and Communications Program's Earth and Space Sciences Project under NRA Grant NAG5-2497,
- The PRISM project, sponsored by ARPA,
- The Intel Research Council.

CONTENTS

Parallel Implementation of BLAS

Preliminary

We will first show:

- The presented vector manipulations can be used to parallelize the level 2 BLAS.
- The resulting level 2 BLAS can be used to parallelize the level 3 BLAS.
- The introduction of "level 2.5 BLAS" can be used to efficiently implement the level 3 BLAS.
- Performance results for level 3 BLAS using these techniques (in the simplified setting of "traditional" matrix distributions).

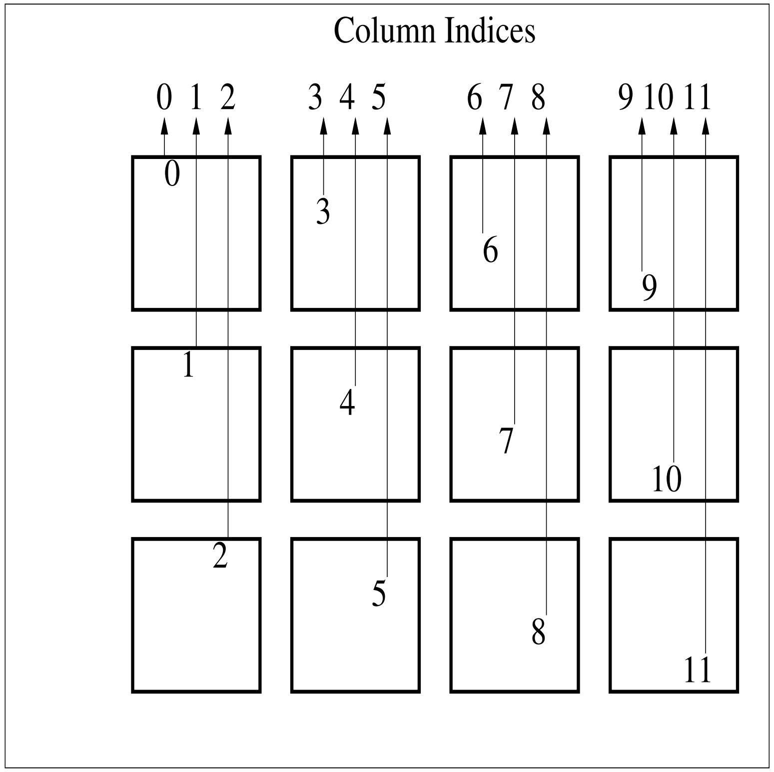
BLAS implementation: Partitioning

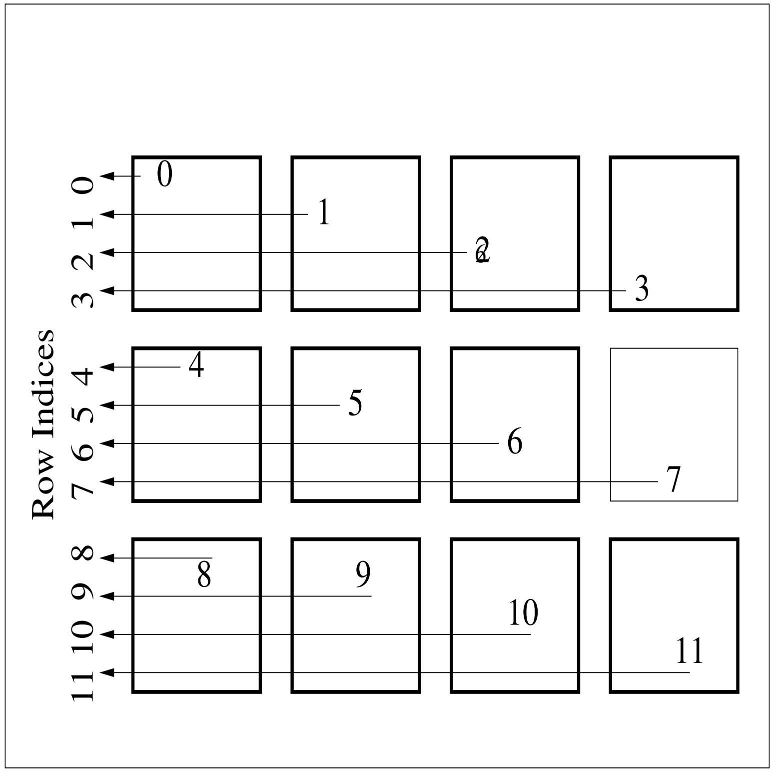
Partition as before:

$$P_x x = \left(\begin{array}{c} x_0 \\ \hline x_1 \\ \hline \vdots \\ \hline x_{p-1} \end{array}\right) \quad \text{and} \quad P_y y = \left(\begin{array}{c} y_0 \\ \hline y_1 \\ \hline \vdots \\ \hline y_{p-1} \end{array}\right)$$

$$P_y A P_x^T = \left(\begin{array}{c|c|c|c} A_{0,0} & A_{0,1} & \cdots & A_{0,p-1} \\ \hline A_{1,0} & A_{1,1} & \cdots & A_{1,p-1} \\ \hline \vdots & \vdots & & \vdots \\ \hline A_{p-1,0} & A_{p-1,1} & \cdots & A_{p-1,p-1} \end{array}\right)$$

BLAS implementation: Distribution



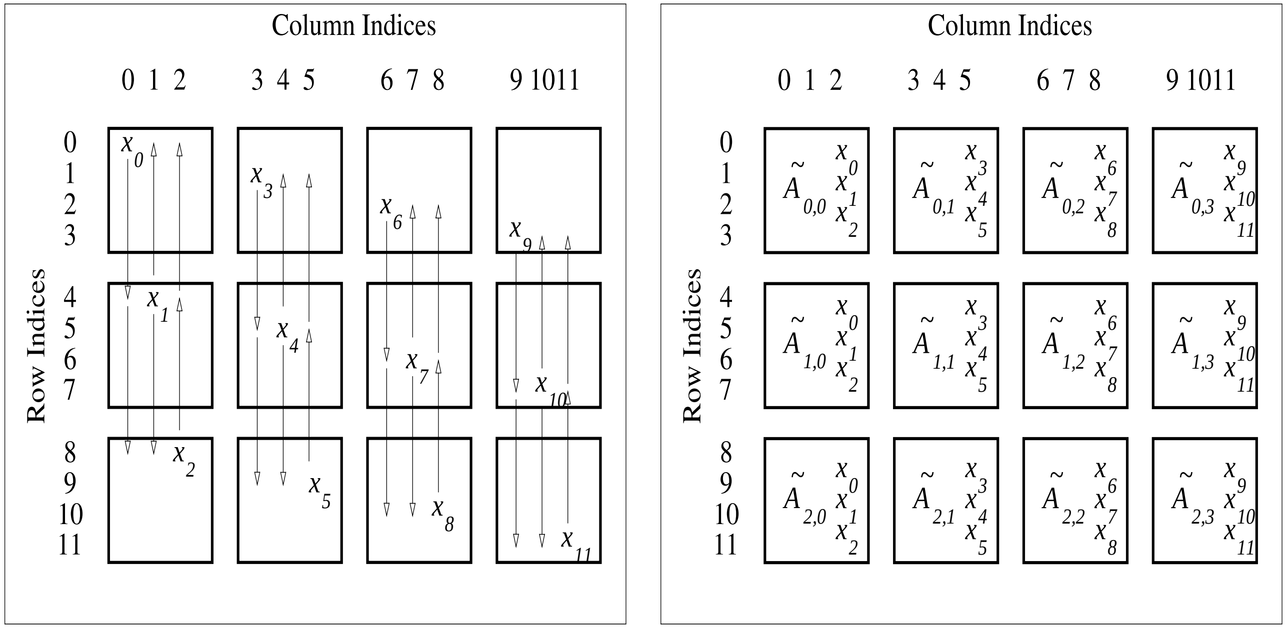


BLAS implementation: Distribution (Continued)

0,0	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	0,10	0,11
1,0	1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8	1,9	1,10	1,11
2,0	2,1	2,2	2,3	2,4	2,5	2,6	2,7	2,8	2,9	2,10	2,11
3,0	3,1	3,2	3,3	3,4	3,5	3,6	3,7	3,8	3,9	3,10	3,11
4,0	4,1	4,2	4,3	4,4	4,5	4,6	4,7	4,8	4,9	4,10	4,11
5,0	5,1	5,2	5,3	5,4	5,5	5,6	5,7	5,8	5,9	5,10	5,11
6,0	6,1	6,2	6,3	6,4	6,5	6,6	6,7	6,8	6,9	6,10	6,11
7,0	7,1	7,2	7,3	7,4	7,5	7,6	7,7	7,8	7,9	7,10	7,11
8,0	8,1	8,2	8,3	8,4	8,5	8,6	8,7	8,8	8,9	8,10	8,11
9,0	9,1	9,2	9,3	9,4	9,5	9,6	9,7	9,8	9,9	9,10	9,11
10,0	10,1	10,2	10,3	10,4	10,5	10,6	10,7	10,8	10,9	10,10	10,11
11,0	11,1	11,2	11,3	11,4	11,5	11,6	11,7	11,8	11,9	11,10	11,11

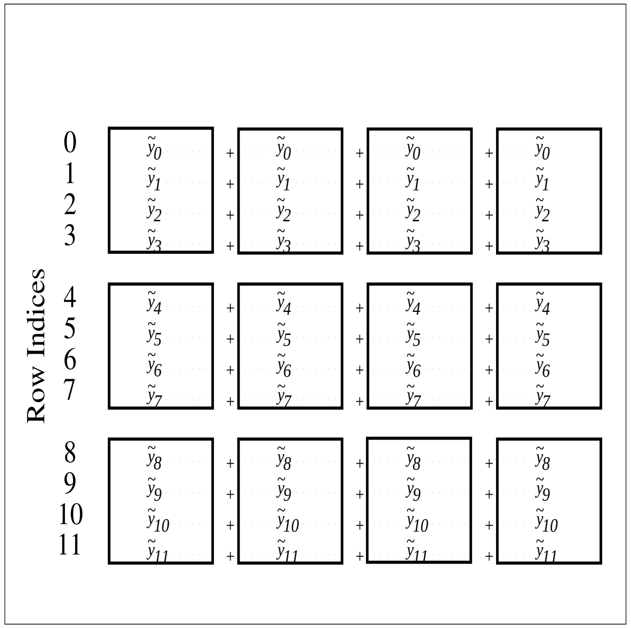
Parallelizing Level 2 BLAS

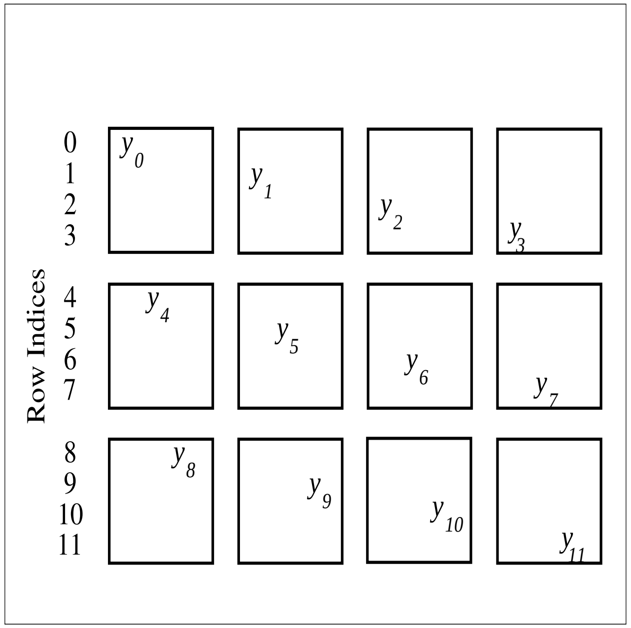
Matrix-Vector Multiply: Algorithm



- Step 1: Spread x.
- Step 2: Local MV multiply.

Matrix-Vector Multiply: Algorithm (continued)





- Step 3: Distributed Sum within rows.

Matrix-Vector Multiply: Other Cases

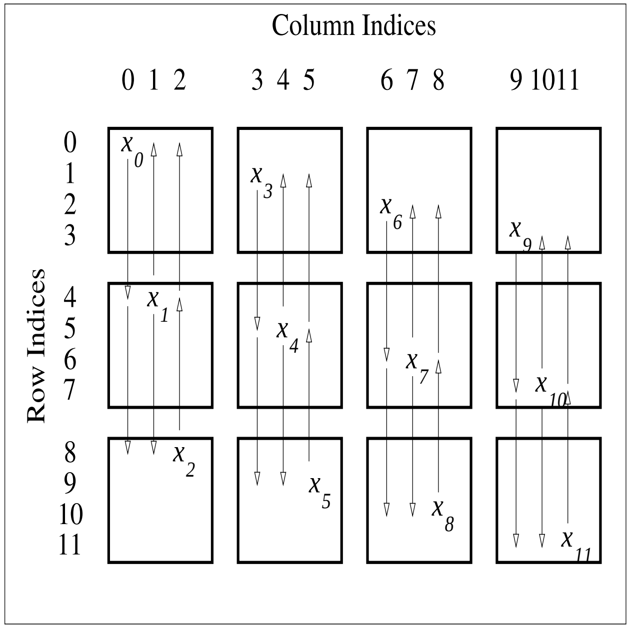
Consider $y = Ax$

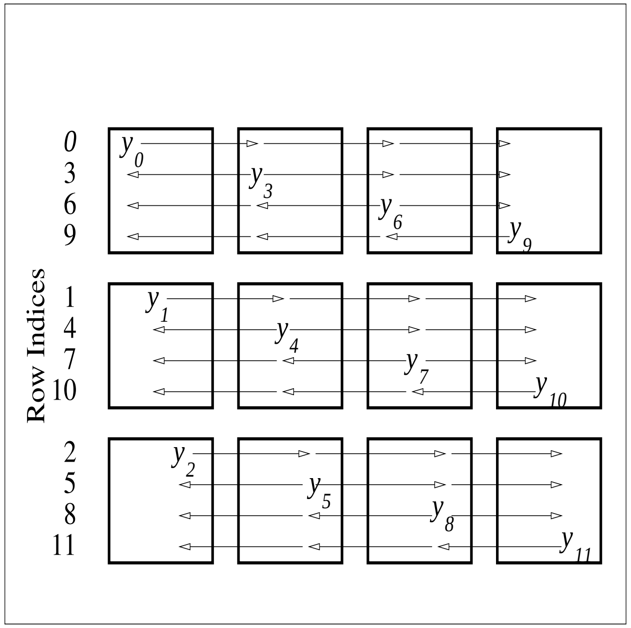
- Case: x is a row, y is a column.
  - $\star$ Step 1: Redistribute row x to cannonical (vector) form.
  - $\star$ Step 2: Perform matrix-vector multiply.
  - $\star$ Step 3: Redistribute y from cannonical form to column.
  - $\star$ **Shortcut: broadcast x within columns, sum-to-one within rows to yield y**

Matrix-Vector Multiply: Other Cases (continued)

- Case: x is a column, y is a row.
  - ⋆ Step 1a: Redistribute row x to cannonical (vector) form.
  - ⋆ Step 1b: Permute if necessary.
  - ⋆ Step 2: Perform matrix-vector multiply.
  - ⋆ Step 3a: Permute if necessary.
  - ⋆ Step 3b: Redistribute y from cannonical form to column.

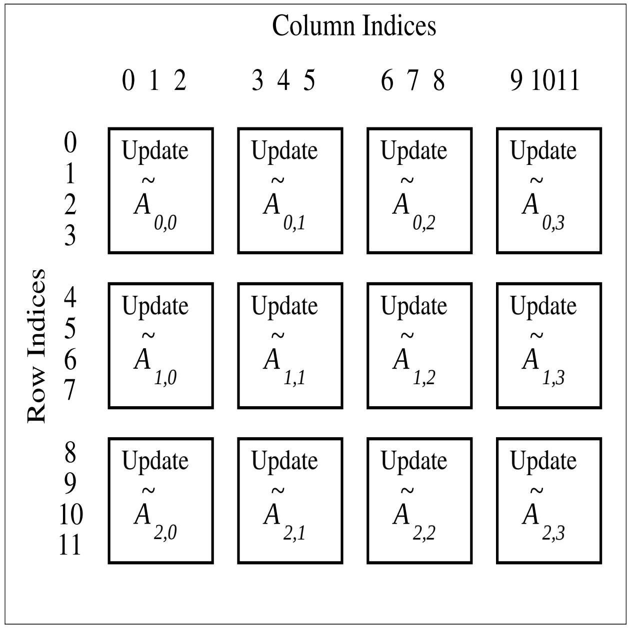
Rank-1 Update: Algorithm





- Step 1: Spread x.
- Step 2: Spread y.

Rank-1 Update: Algorithm (continued)



- Step 3: Local rank-1 update.

Parallelizing Level 3 BLAS

Building Level 3 BLAS on Level 2 BLAS

- All level 3 BLAS can be effectively parallelized using the Matrix-Vector multiply and Rank-1 update.
- Efficiency can be attained by introducing “Level 2.5” BLAS: Matrix-Panel and Rank-k update.
- Reference: Almadena Chtchelkanova, John Gunnels, Greg Morrow, James Overfelt, Robert A. van de Geijn, "Parallel Implementation of BLAS: General Techniques for Level 3 BLAS," submitted to *Concurrency: Practice and Experience*.

Partitioning the Matrices

$$A = \left(\begin{array}{c|c|c|c} A_{0,0} & A_{0,1} & \cdots & A_{0,N-1} \\ \hline A_{1,0} & A_{1,1} & \cdots & A_{1,N-1} \\ \hline \vdots & \vdots & & \vdots \\ \hline A_{N-1,0} & A_{N-1,1} & \cdots & A_{N-1,N-1} \end{array}\right)$$

- Same for B and C.

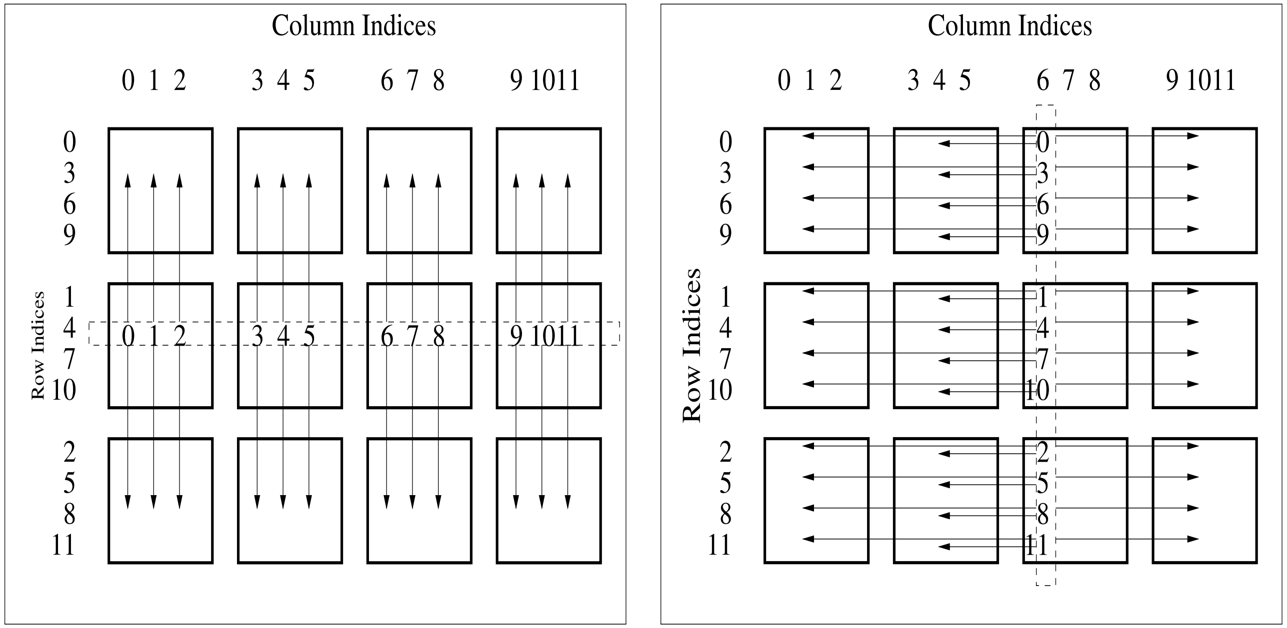
Matrix-Matrix Multiply: $C = AB$

$$C = AB = \left(\begin{array}{c} A_{00} \\ \hline A_{10} \\ \hline \vdots \\ \hline A_{(N-1)0} \end{array}\right) \left(B_{00} \,\middle|\, B_{01} \,\middle|\, \cdots \,\middle|\, B_{0(N-1)} \right) \tag{1}$$

$$+ \left(\begin{array}{c} A_{01} \\ \hline A_{11} \\ \hline \vdots \\ \hline A_{(N-1)1} \end{array}\right) \left(B_{10} \,\middle|\, B_{11} \,\middle|\, \cdots \,\middle|\, B_{1(N-1)} \right) + \cdots \tag{2}$$

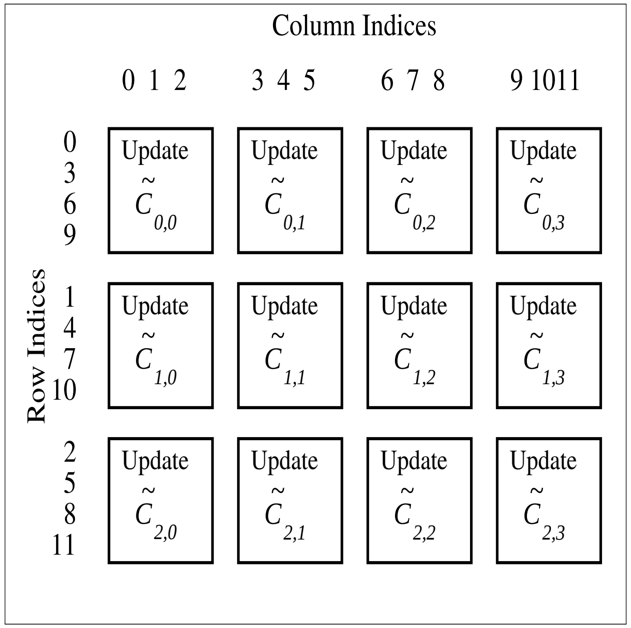
$$+ \left(\begin{array}{c} A_{0(N-1)} \\ \hline A_{1(N-1)} \\ \hline \vdots \\ \hline A_{(N-1)(N-1)} \end{array}\right) \left(B_{(N-1)0} \,\middle|\, B_{(N-1)1} \,\middle|\, \cdots \,\middle|\, B_{(N-1)(N-1)} \right) \tag{3}$$

Matrix-Matrix Multiply: $C = AB$ (continued)



- Step 1: Spread current row of B
- Step 2: Spread current column of A.

Matrix-Matrix Multiply: $C = AB$ (continued)



- Step 3: Local rank-1 update.

Matrix-Matrix Multiply: $C = AB^T$

$$\left(\begin{array}{c} C_{0i} \\ \hline C_{1i} \\ \hline \vdots \\ \hline C_{(N-1)i} \end{array} \right) = A \left(\; B_{i0} \,\middle|\, B_{i1} \,\middle|\, \cdots \,\middle|\, B_{i(N-1)} \; \right)$$

- C is computed one column at a time.

Matrix-Matrix Multiply: $C = A^T B^T$

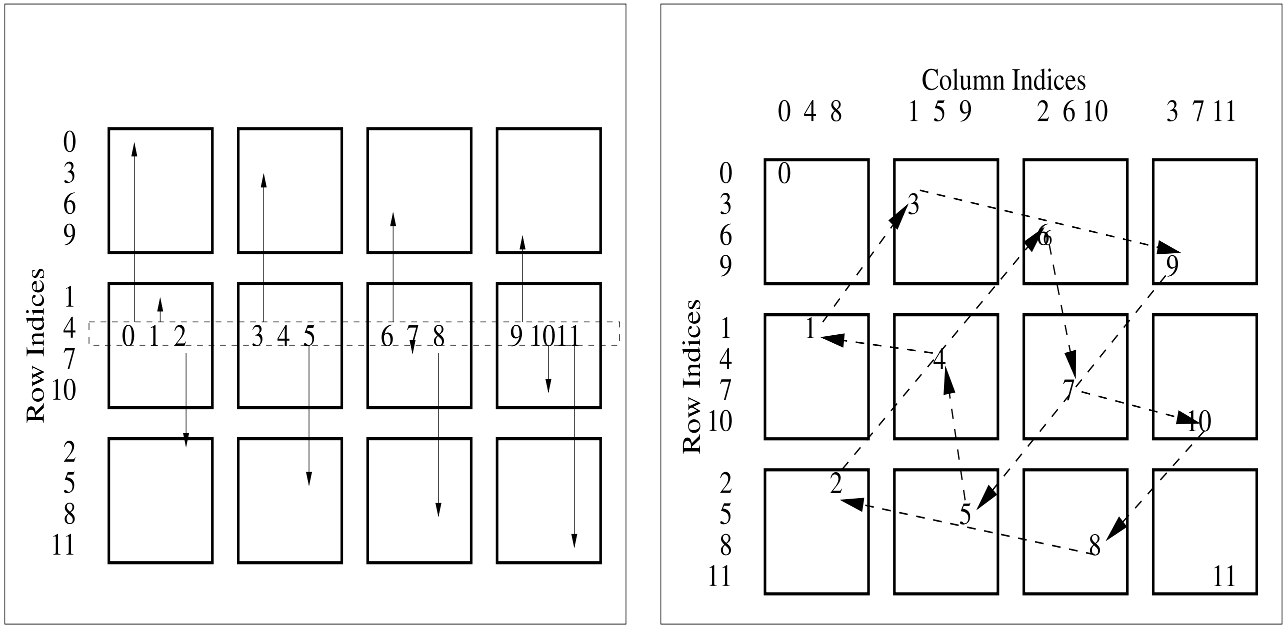
$$C \;=\; A^T B^T = \left(A_{00} \middle| A_{01} \middle| \cdots \middle| A_{0(N-1)} \right)^T \left(\begin{array}{c} B_{00} \\ \hline B_{10} \\ \hline \vdots \\ \hline B_{(N-1)0} \end{array} \right)^T \tag{4}$$

$$+ \quad \left(A_{10} \middle| A_{11} \middle| \cdots \middle| A_{1(N-1)} \right)^T \left(\begin{array}{c} B_{01} \\ \hline B_{11} \\ \hline \vdots \\ \hline B_{(N-1)1} \end{array} \right)^T + \cdots \tag{5}$$

Matrix-Matrix Multiply: $C = A^T B^T$ (continued)

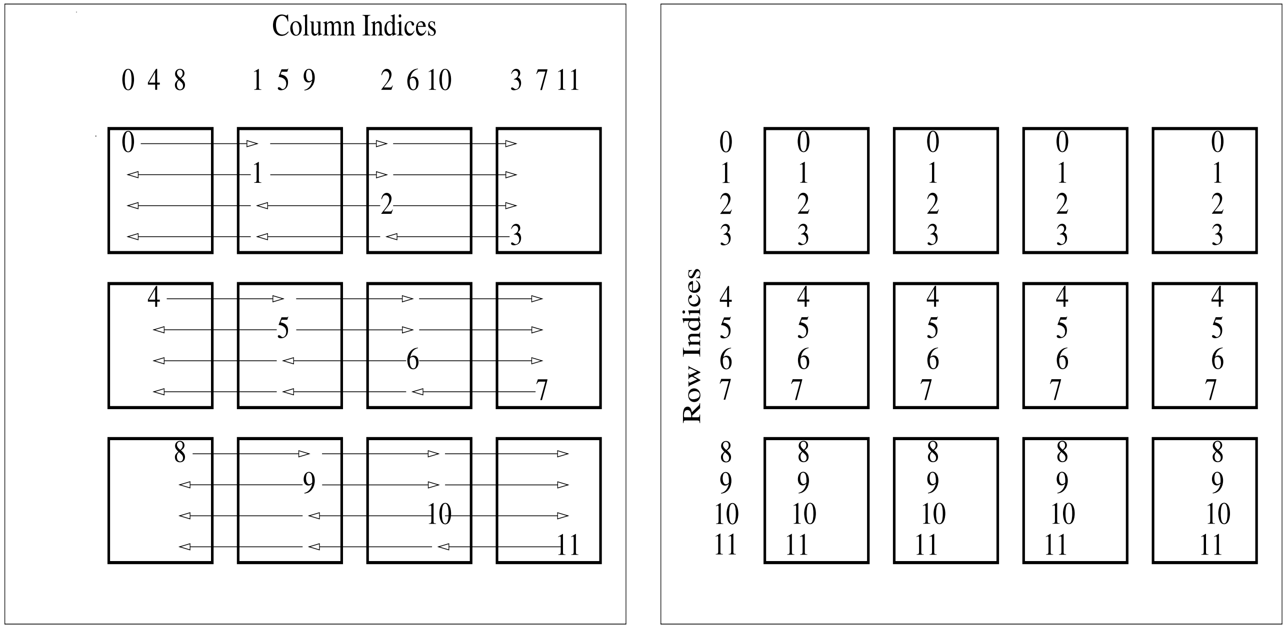
- Step 1: Row of A to cannonical form. Distributes like vector that induces column distribution of A.
- Step 2: Redistributed (if necessary) like vector that induces row distribution of C.
- Step 3: Spread within rows.
- Step 4: Column of B to cannonical form. Distributes like vector that induces row distribution of B.
- Step 5: Redistributed (if necessary) like vector that induces column distribution of C.
- Step 6: Spread within columnss.
- Step 7: Peform local rank-1 update.

Matrix-Matrix Multiply: $C = A^T B^T$ (continued)



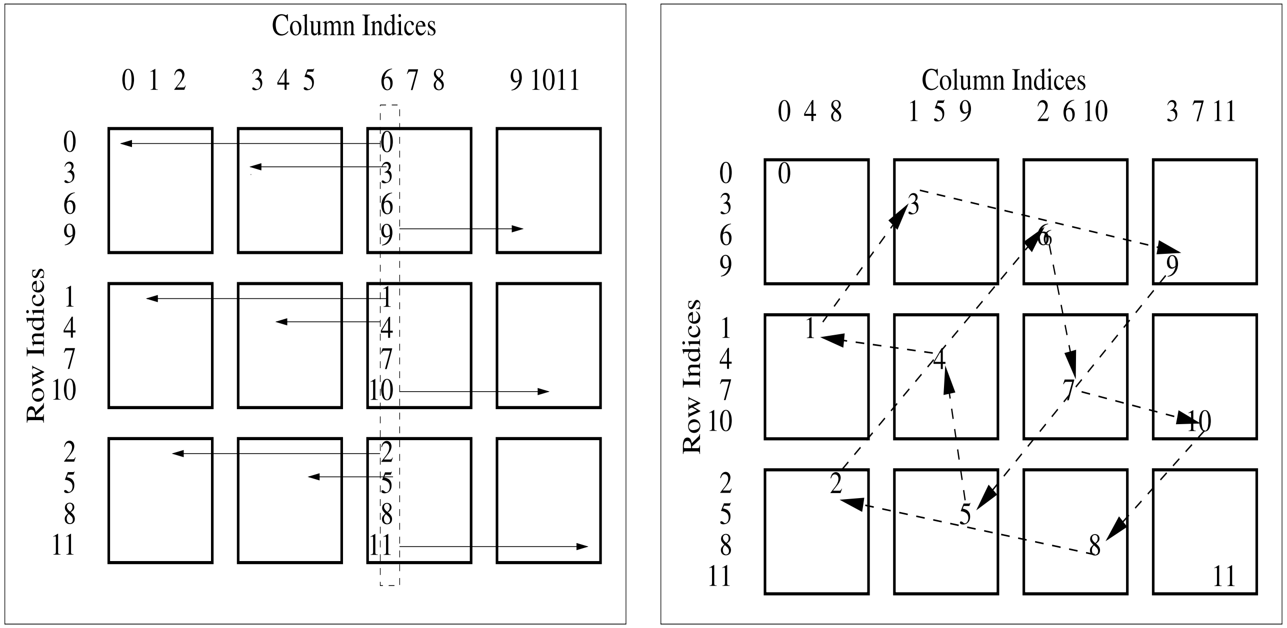
- Step 1: Spread current row of A within rows.

Matrix-Matrix Multiply: $C = A^T B^T$ (continued)

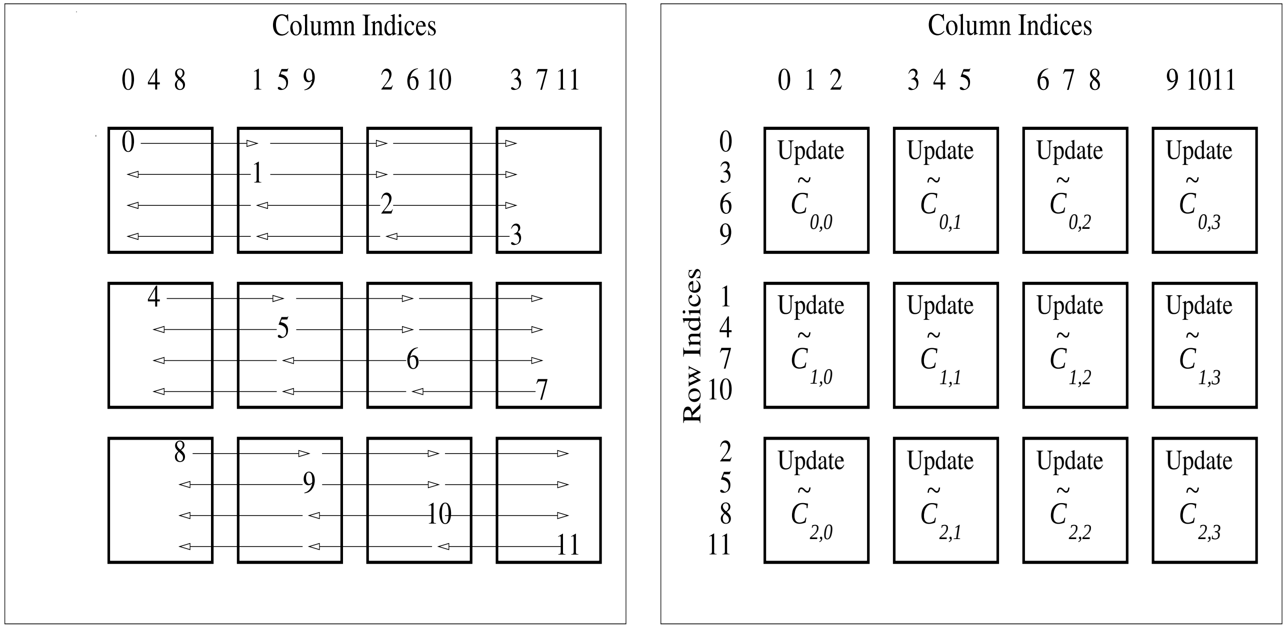


- Step 1: Spread current row of A within rows (continued).

Matrix-Matrix Multiply: $C = A^T B^T$ (continued)



- Step 2: Spread current column of B within columns.



- Step 2: Spread current column of B within columns (continued).
- Step 3: Local rank-1 updates.

Triangular Solve with Multiple RHS: $B = A^{-T}B$

- Assume: A lower triangular.
- $AX = B$ where X overwrites B.

$$\left(\begin{array}{c||c|c|c} A_{00} & 0 & \cdots & 0 \\ \hline\hline A_{10} & A_{11} & \cdots & 0 \\ \hline \vdots & \vdots & & \vdots \\ \hline A_{(N-1)0} & A_{(N-1)1} & \cdots & A_{(N-1)(N-1)} \end{array}\right) \left(\begin{array}{c} X_{0*} \\ \hline\hline X_{1*} \\ \hline \vdots \\ \hline X_{(N-1)*} \end{array}\right) = \left(\begin{array}{c} B_{0*} \\ \hline\hline B_{1*} \\ \hline \vdots \\ \hline B_{(N-1)*} \end{array}\right)$$

Implies

$$A_{00} \left(X_{00} \middle| X_{01} \middle| \cdots \middle| X_{0(N-1)} \right) = \left(B_{00} \middle| B_{01} \middle| \cdots \middle| B_{0(N-1)} \right) \tag{6}$$

and

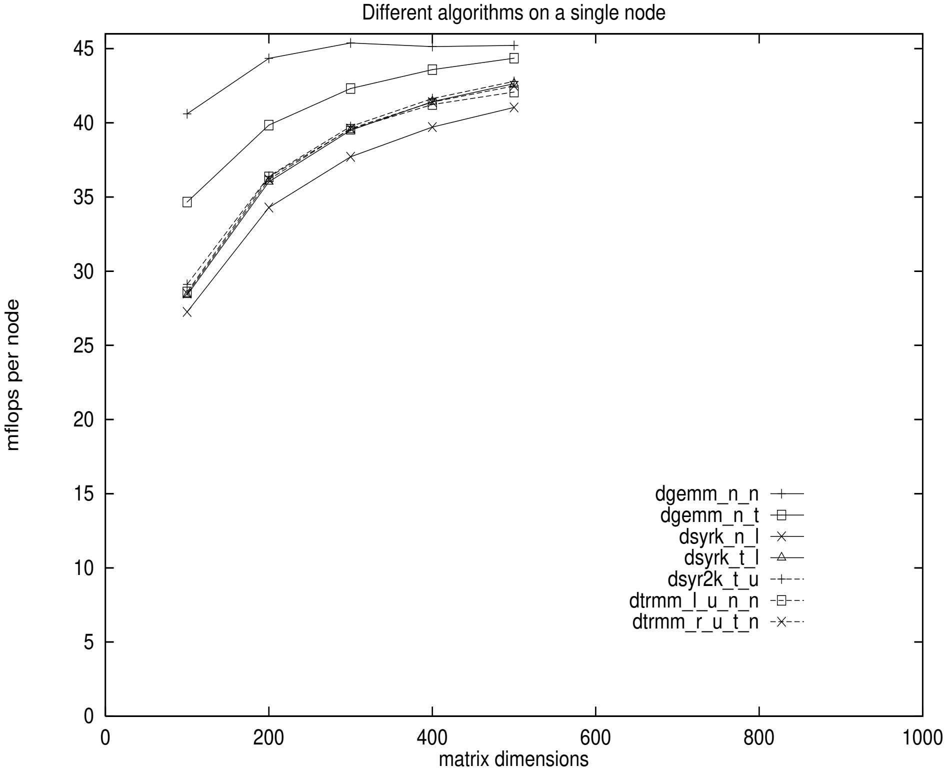
$$\left(\begin{array}{c|c|c} A_{11} & \cdots & 0 \\ \hline \vdots & & \vdots \\ \hline A_{(N-1)1} & \cdots & A_{(N-1)(N-1)} \end{array}\right) \left(\begin{array}{c} X_{1*} \\ \hline \vdots \\ \hline X_{(N-1)*} \end{array}\right) = \left(\begin{array}{c} B_{1*} \\ \hline \vdots \\ \hline B_{(N-1)*} \end{array}\right) - \left(\begin{array}{c} A_{10} \\ \hline \vdots \\ \hline A_{(N-1)0} \end{array}\right) \left(X_{0*} \right) \tag{7}$$

- Step 1: Compute X_{0*}.
- Step 2: Update B with a rank-1 update.
- Step 3: Recursively continue with smaller system.

Performance

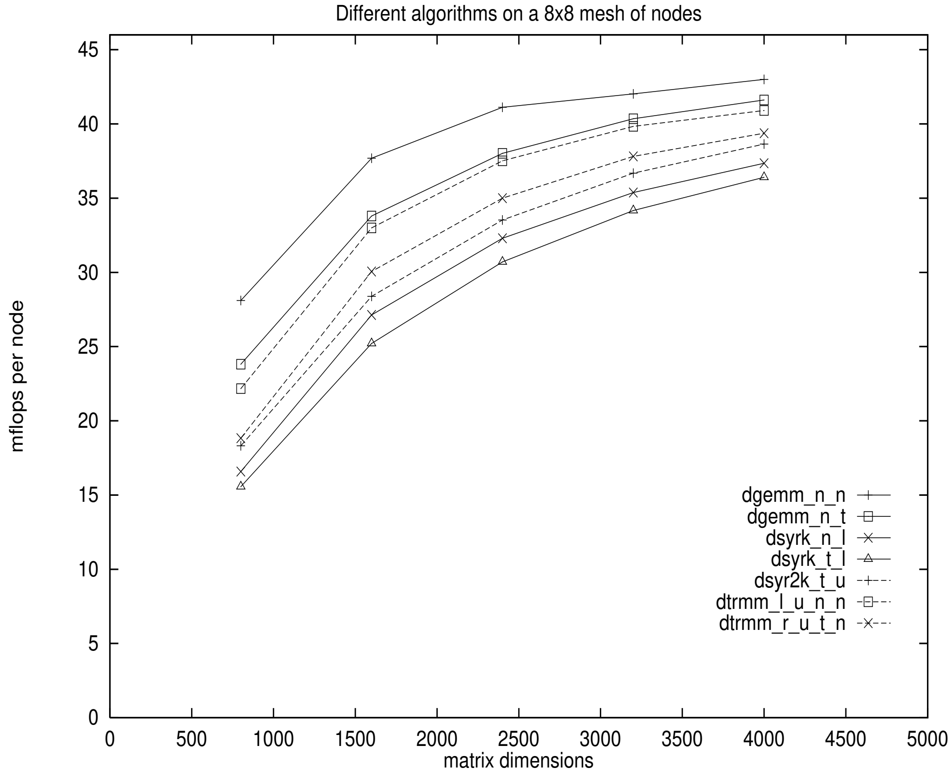
- Platform: 512 node Intel Paragon System at Caltech.
- OS: OSF R1.3.
- Communication library: MPI.
- Compute kernels: single node BLAS.

Performance (continued)



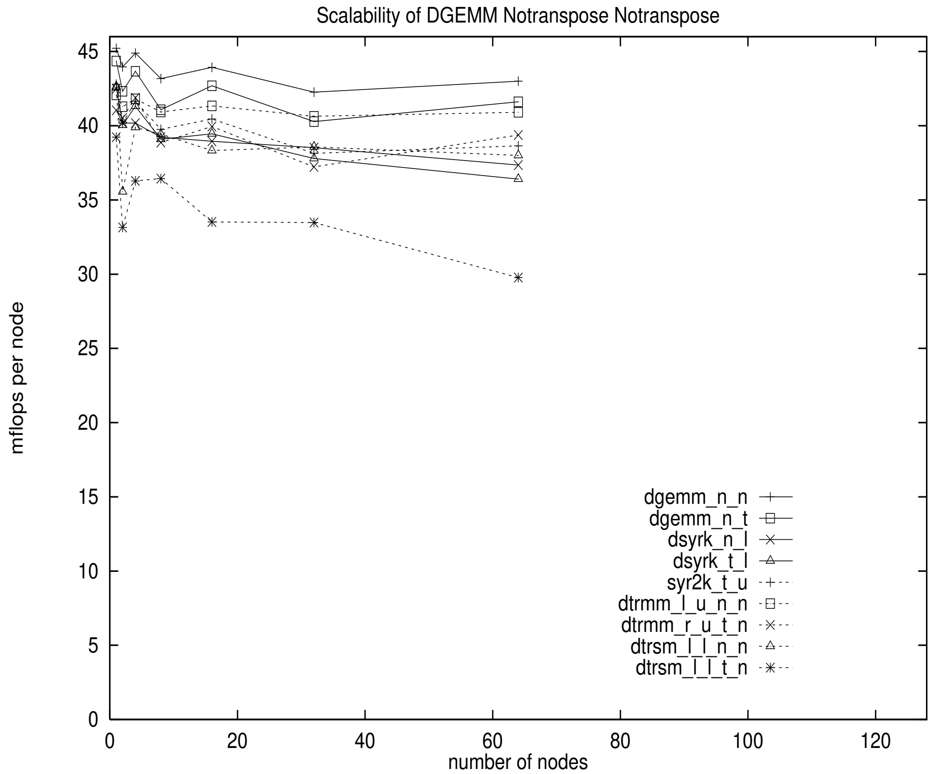
- Performance of various algorithms on a single node.

Performance (continued)



- Performance on 8×8 mesh.

Scalability



- Scalability memory utilization per node is equivalent to a 500×500 matrix.