Parallel Numerics, WT 2012/2013

2 Elementary Linear Algebra Problems

Contents

1 Introduction
- 1.1 Computer Science Aspects
- 1.2 Numerical Problems
- 1.3 Graphs
- 1.4 Loop Manipulations

2 Elementary Linear Algebra Problems
- 2.1 BLAS: Basic Linear Algebra Subroutines
- 2.2 Matrix-Vector Operations
- 2.3 Matrix-Matrix-Product

3 Linear Systems of Equations with Dense Matrices
- 3.1 Gaussian Elimination
- 3.2 Parallelization
- 3.3 QR-Decomposition with Householder matrices

4 Sparse Matrices
- 4.1 General Properties, Storage
- 4.2 Sparse Matrices and Graphs
- 4.3 Reordering
- 4.4 Gaussian Elimination for Sparse Matrices

5 Iterative Methods for Sparse Matrices
- 5.1 Stationary Methods
- 5.2 Nonstationary Methods
- 5.3 Preconditioning

6 Domain Decomposition

2.1. BLAS: Basic Linear Algebra Subroutines

- First published 1979
- Library of basic linear algebra operations
- Organized in levels according to complexity:

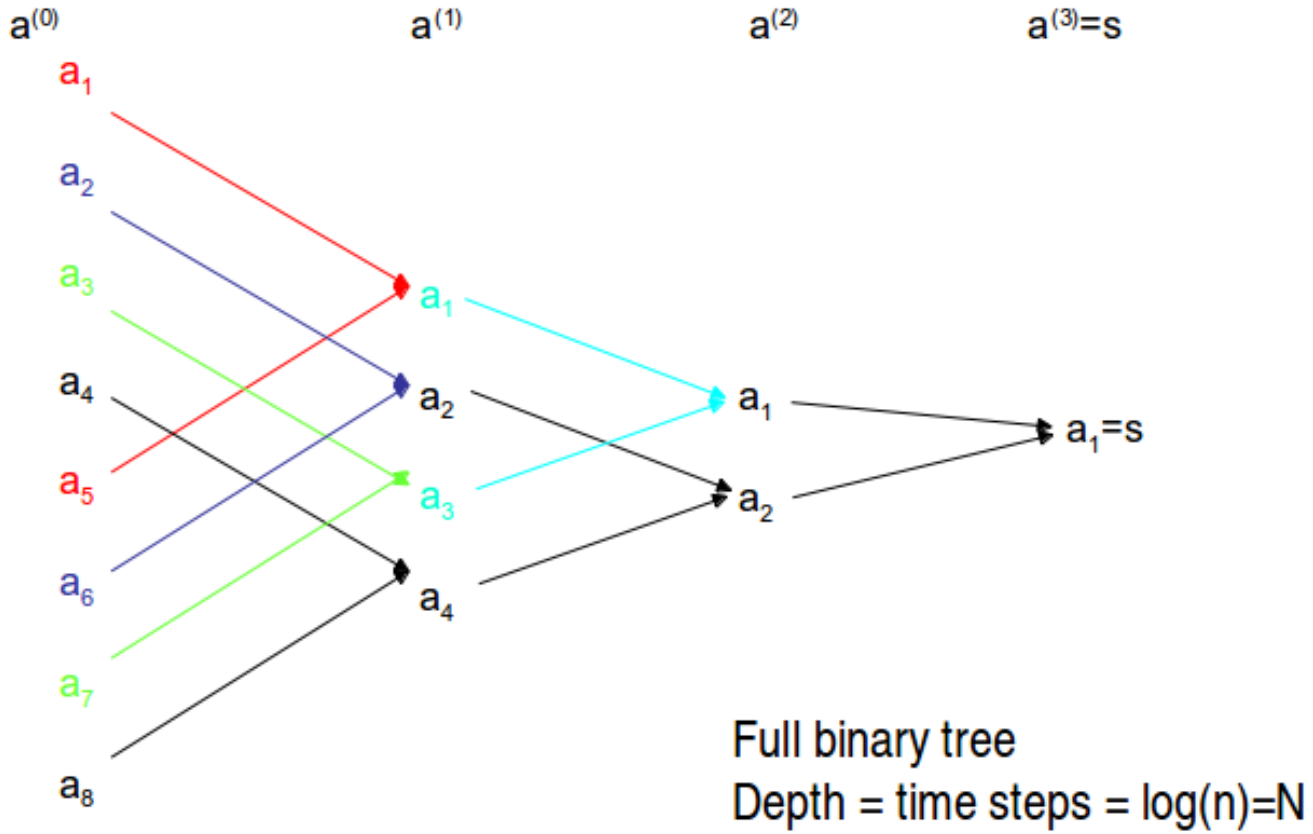
BLAS level	type of operation	complexity
BLAS-1	vector-vector	$\mathcal{O}(n)$
BLAS-2	matrix-vector	$\mathcal{O}(n^2)$
BLAS-3	matrix-matrix	$\mathcal{O}(n^3)$

- There are machine-specific optimized BLAS libraries
- Basis of other libraries, e.g., LAPACK (Linear Algebra Package: for solving linear equations, least squares problems, QR-decomposition, eigenvalues, singular values...)

Computation of Sum in Parallel

Sum of vector components: $s = \sum_{j=1}^{n} a_j$.

Computation by fan-in process:



Vector update in the fan-in process with $n = 2^N$

$$a^{(k)} = \begin{pmatrix} a_1^{(k)} \\ \vdots \\ \vdots \\ a_{2^{N-k}}^{(k)} \end{pmatrix} = \begin{pmatrix} a_1^{(k-1)} \\ \vdots \\ \vdots \\ a_{2^{N-k}}^{(k-1)} \end{pmatrix} + \begin{pmatrix} a_{2^{N-k}+1}^{(k-1)} \\ \vdots \\ \vdots \\ a_{2^{N-k+1}}^{(k-1)} \end{pmatrix}$$

$$a_1 + \ldots + a_8 = [(a_1 + a_5) + (a_3 + a_7)] + [(a_2 + a_6) + (a_4 + a_8)]$$

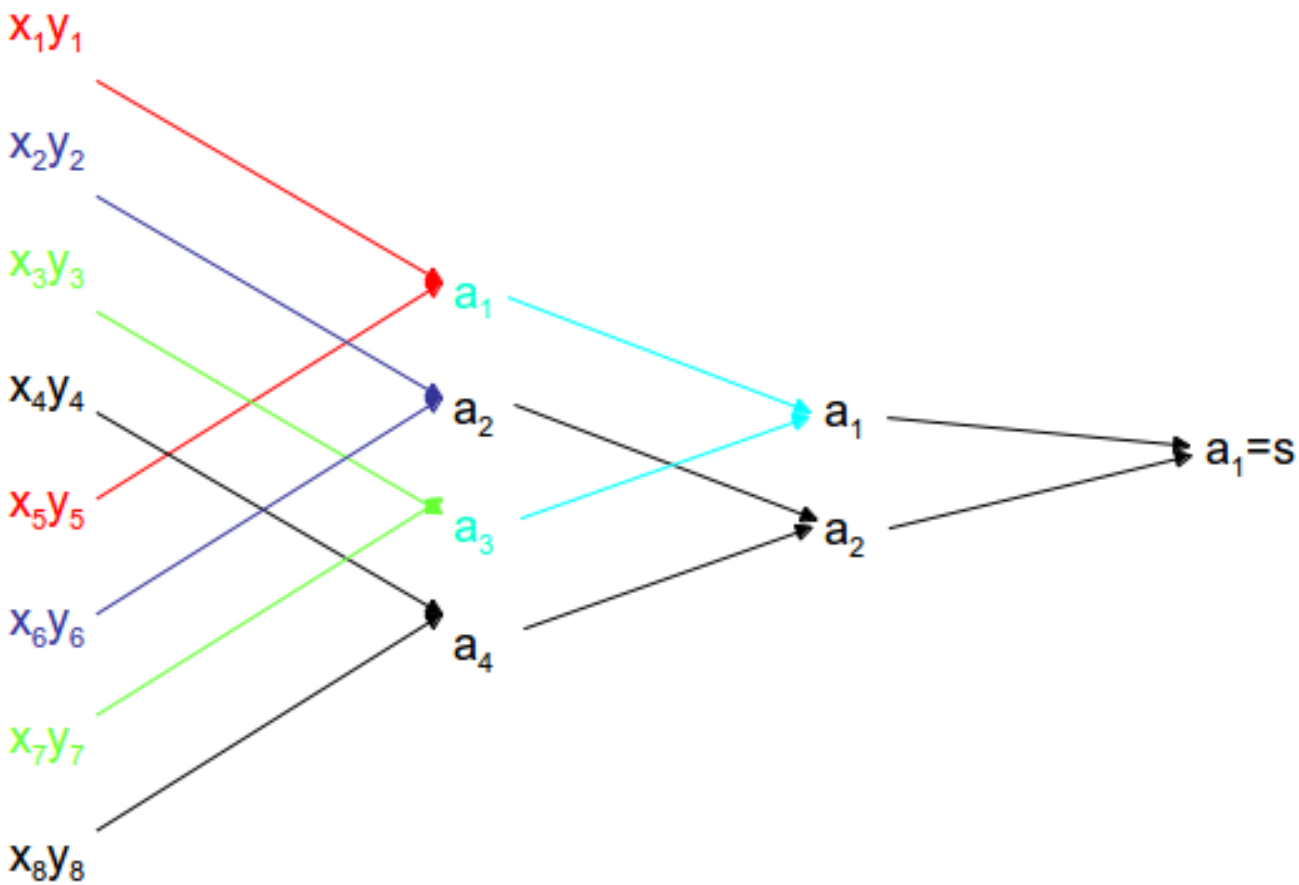
```
for(k=1;k<=N;k++)
   for(j=1;j<=2^(N-k);j++)
      a_j = a_j + a_(j+2^(N-k))
   end
end
```

$\log(n) = N$ time steps in parallel for vector of length n

Level-1 BLAS

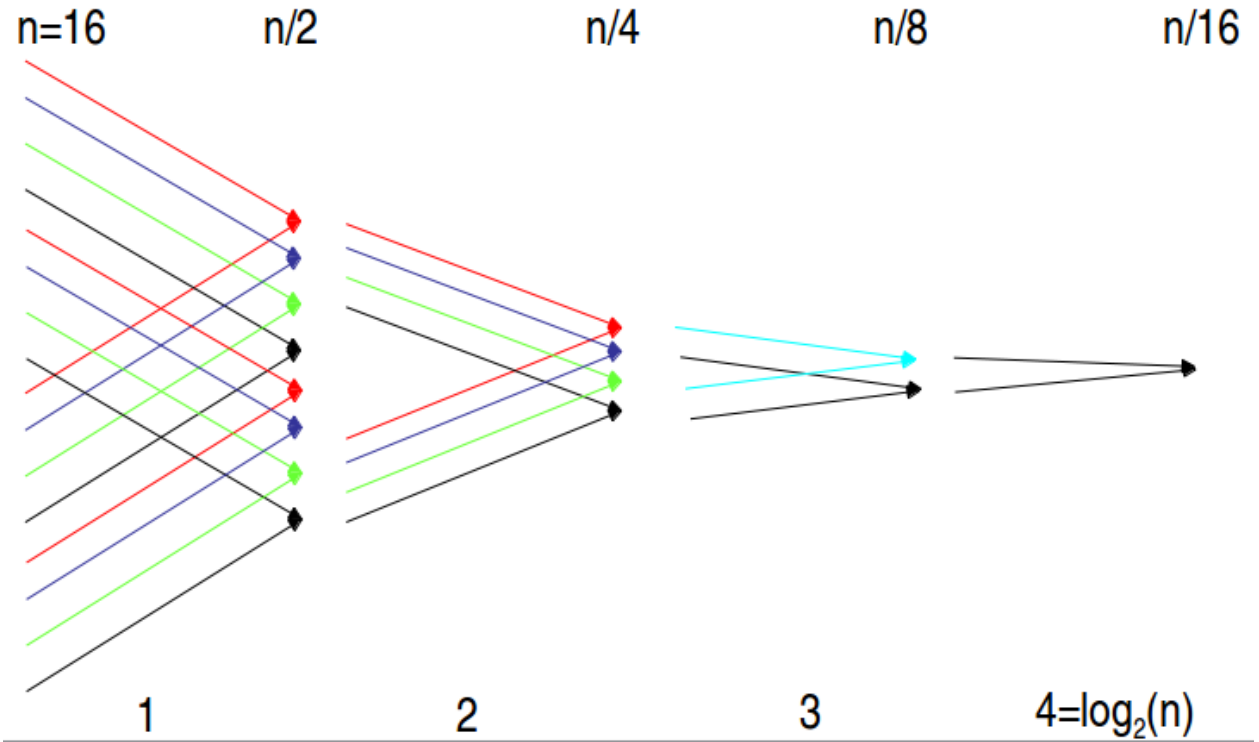
BLAS routines with $\mathcal{O}(n)$ problems (vectors only, $x, y \in \mathbb{R}^n$).

First example: DOT-product by fan-in: $s = x^T y = \sum_{i=1}^{n} x_j y_j$



DOT-Product in parallel

- Time steps in parallel of the DOT-product cannot be better than $\log(n)$.
- Every computation involving fan-in will take $\log(n)$ time steps in parallel.



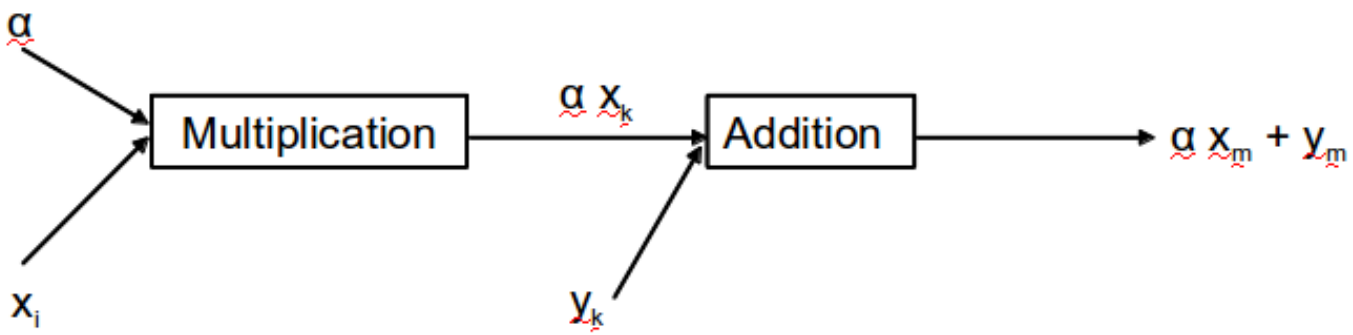
Level-1 BLAS: SAXPY

BLAS-Notation:

- S single precision (D for double, C for complex)
- A α scalar
- X vector
- P plus operation
- Y vector

SAXPY: $y = \alpha x + y$

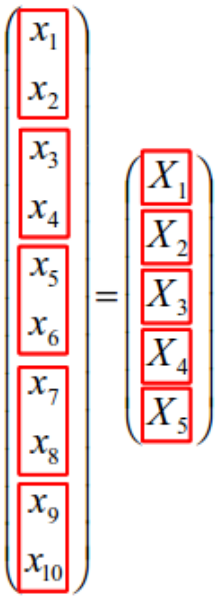
Vectorization of SAXPY ($\alpha x + y$) by pipelining:



SAXPY Parallelization by Partitioning

$$\{1, 2, \ldots, n\} = \langle 1, n \rangle = I_1 \cup I_2 \cup \ldots \cup I_R$$

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} X_1 \\ \vdots \\ X_R \end{pmatrix}, \qquad y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} Y_1 \\ \vdots \\ Y_R \end{pmatrix}$$



- Using short vectors X_j and Y_j of length $\frac{n}{R}$.
- Each processor P_j gets partial vector X_j and Y_j and computes $Y_j = \alpha X_j + Y_j,\ j = 1, 2, \ldots, R$.
- Result: SAXPY very good vectorizable and parallelizable.

Further Level-1 BLAS Routines

- SCOPY: $y = x$ or $y \leftarrow x$ (compare SAXPY)
- DOT-product $x^T y$: $\sum_i x_i y_i$
- norm: $\|x\|_2 = \sqrt{\sum_{j=1}^{n} x_j^2} = \sqrt{x^T x}$ (compare DOT-product)

Level-2 BLAS

- Matrix-Vector operations with $\mathcal{O}(n^2)$ operations (sequentially)
- BLAS-Notation:

S	single precision
G E M	general matrix
V	vector

- defines SGEMV, matrix-vector product: $y = \alpha Ax + \beta y$
- Other Level-2 BLAS: solving triangular system $Lx = b$ with triangular matrix L.

Level-3 BLAS

- Matrix-Matrix operations with $\mathcal{O}(n^3)$ operations (sequentially)
- BLAS-Notation:

S	single precision
G E M	general matrix
M	matrix

- defines SGEMM, matrix-matrix product: $C = \alpha AB + \beta C$

Granularity for BLAS

BLAS level	operation	formula	memory	granularity
BLAS-1	AXPY: $2n$	$\alpha x + y$	$2n + 1$	< 1
BLAS-2	GEMV: $2n^2$	$\alpha Ax + \beta y$	$n^2 + 2n$	2
BLAS-3	GEMM: $2n^3$	$\alpha AB + \beta C$	$4n^2$	$\frac{n}{2}$

BLAS-3 has best operations to memory ratio!

2.2. Analysis of the Matrix-Vector-Product

$$A = (a_{ij})_{\substack{i=1,\dots,n \\ j=1,\dots,m}} \in \mathbb{R}^{n\times m}, \qquad b \in \mathbb{R}^m, \qquad c \in \mathbb{R}^n$$

2.2.1. Vectorization

$$\begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix} = \begin{pmatrix} a_{11}b_1 + \cdots + a_{1m}b_m \\ \vdots \\ a_{n1}b_1 + \cdots + a_{nm}b_m \end{pmatrix} =$$

$$= \begin{pmatrix} \sum_{j=1}^{m} a_{1j}b_j \\ \vdots \\ \sum_{j=1}^{m} a_{nj}b_j \end{pmatrix} \qquad = \sum_{j=1}^{m} b_j \begin{pmatrix} a_{1j} \\ \vdots \\ a_{nj} \end{pmatrix}$$

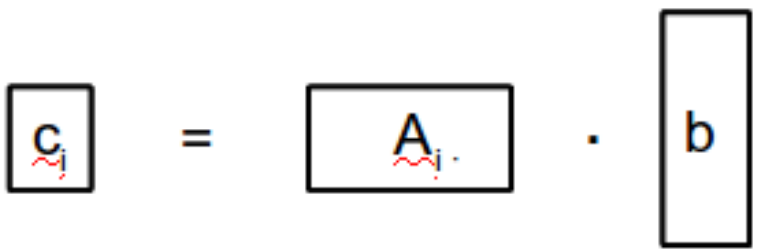
n DOT-products of length m — m SAXPYs of length n (GAXPY)

Pseudocode: *ij*-form

```
c = 0;
for i=1,...,n
    for j=1,...,m
        c_i = c_i + a_ij b_j
    end
end
```

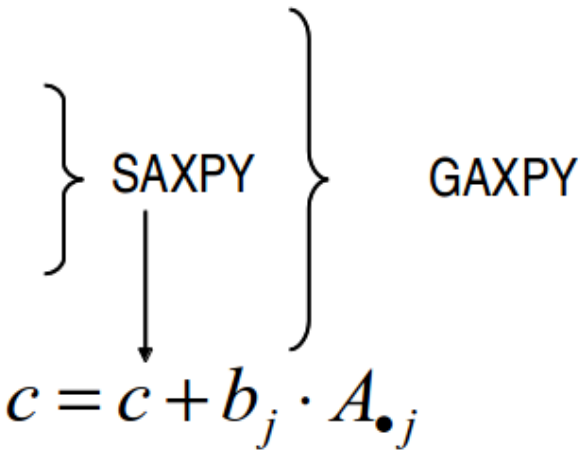
} DOT-product

$c_i = A_{i\bullet} b$, DOT-product of ith row of A with vector b



Pseudocode: *ji*-form

```
c = 0;
for j=1,...,m
    for i=1,...,n
        c_i = c_i + a_ij b_j
    end
end
```



- SAXPY updating vector *c* with *j*th column of *A*
- GAXPY:
  - Sequence of SAXPYs related to the same vector
  - Advantage: vector *c*, that is updated, can be kept in fast memory
- No additional data transfer

GAXPY (repetition)

- SAXPY:

  $y := y + \alpha x$

- GAXPY:

  $y = y_0$

  `for` $i = 1 : n$

  $\quad y := y + \alpha_i x_i$

  `end`

- Series of SAXPYs regarding the same vector y.
- length(GAXPY) $=$ length(y)
- Advantage: less data transfer!

2.2.2. Parallelization by Building Blocks

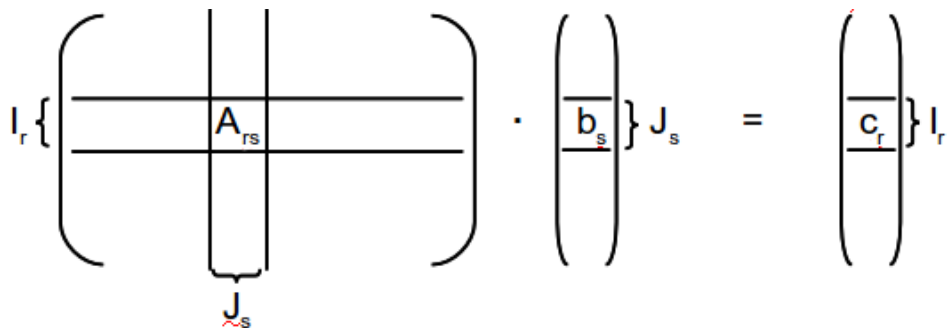
Reduce matrix-vector product on smaller matrix-vector products.

$\{1, 2, \ldots, n\} = \langle 1, n \rangle = I_1 \cup I_2 \cup \ldots \cup I_R$ disjunct: $I_j \cap I_k = 0$ for $j \neq k$

$\{1, 2, \ldots, m\} = \langle 1, m \rangle = J_1 \cup J_2 \cup \ldots \cup J_S$ disjunct: $J_j \cap J_k = 0$ for $j \neq k$

Use 2-dimensional array of processors P_{rs}.

P_{rs} gets matrix block $A_{rs} := A(I_r, J_s)$, $b_s := b(J_s)$, $c_r := c(I_r)$.



$$c_r = \sum_{s=1}^{S} A_{rs} b_s =: \sum_{s=1}^{S} c_r^{(s)}$$

Pseudocode

```
for r = 1,...,R
    for s = 1,...,S
        c_r^(s) = A_rs b_s;
    end
end
```

Small, independent matrix-vector products. No communication necessary during computations!

```
for r = 1,...,R
    c_r = 0
    for s = 1,...,S
        c_r = c_r + c_r^(s);
    end
end
```

Blockwise collection and addition of vectors. Rowwise communication! Fan-in.

Blocking: Special Cases

$S = 1$: The computation of $A_{i\bullet} b$ is vectorizable by GAXPYs.

$$c = \begin{pmatrix} A_{1\bullet} \\ \hline A_{2\bullet} \\ \hline \vdots \end{pmatrix} \cdot b = \begin{pmatrix} A_{1\bullet} b \\ \hline A_{2\bullet} b \\ \hline \vdots \end{pmatrix}$$

No communication necessary between processor $P_1, \ldots, P_R$

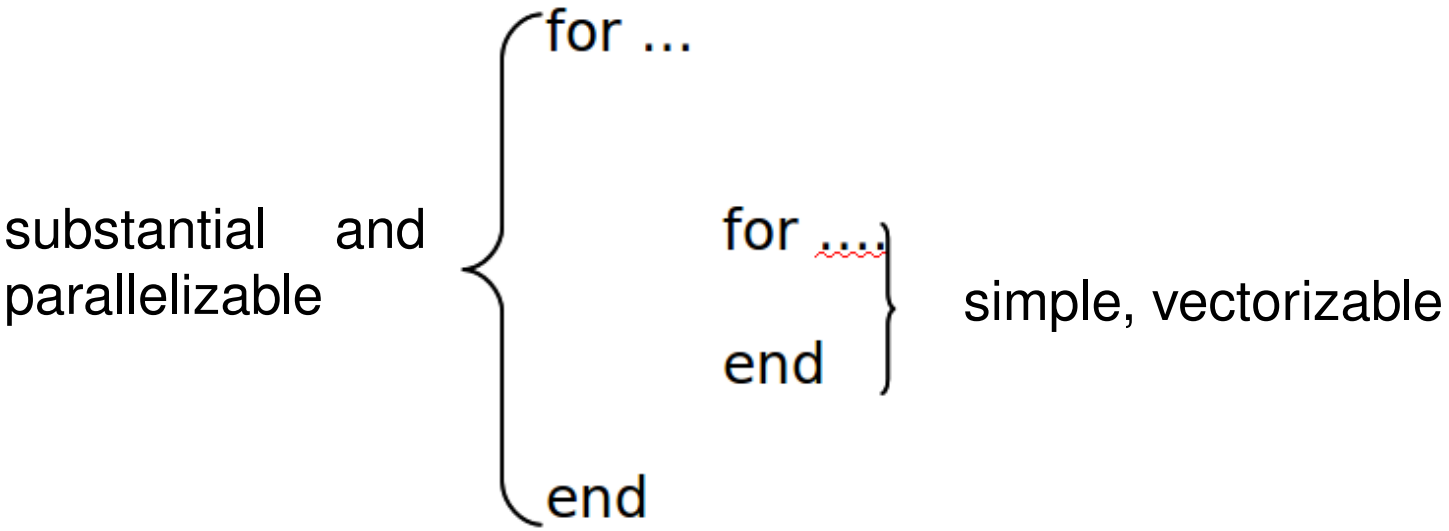
$R = 1$: $A_{\bullet j} b_j$ are independent.

$$c = (A_{\bullet 1} | A_{\bullet 2} | \cdots) \cdot \begin{pmatrix} b_1 \\ \hline b_2 \\ \hline \vdots \end{pmatrix} = A_{\bullet 1} b_1 + A_{\bullet 2} b_2 + \ldots$$

Then collection of partial results from processor $P_1, \ldots, P_S$. Fan-in.
Final sum in one processor: vectorizable by GAXPYs.

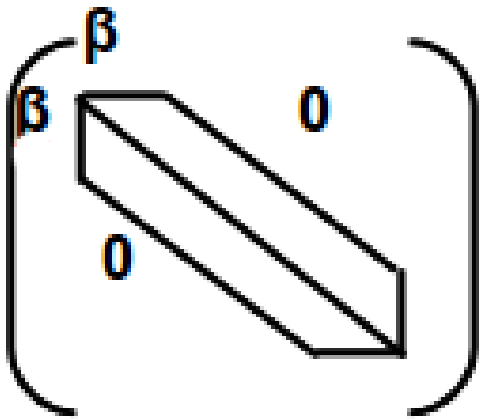
Rules

1. Inner loops of a program should be simple, vectorizable
2. Outer loop of a programm should be substantial, independent, parallelizable.



3. Reuse of data (cache, minimal data transfer, blocking)

2.2.3. $c = Ab$ for Banded Matrix



- Bandwidth β (symmetric)
- 2β+1 diagonals: main diag. + β subdiag. + β superdiag.
- $\beta = 1$: tridiagonal

Notation: Banded Matrices A and $\tilde{A}$

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1,\beta+1} & 0 & \cdots & 0 \\ \vdots & a_{22} & \ddots & \ddots & \cdots & \vdots \\ a_{\beta+1,1} & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & a_{n-\beta,n} \\ \vdots & \vdots & \ddots & \ddots & a_{n-1,n-1} & \vdots \\ 0 & \cdots & 0 & a_{n,n-\beta} & \cdots & a_{nn} \end{pmatrix} \rightarrow$$

$$\tilde{A} = \begin{pmatrix} \tilde{a}_{10} & \cdots & \tilde{a}_{1,\beta} & 0 & \cdots & 0 \\ \vdots & \tilde{a}_{20} & \ddots & \ddots & \cdots & \vdots \\ \tilde{a}_{\beta+1,-\beta} & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \tilde{a}_{n-\beta,\beta} \\ \vdots & \vdots & \ddots & \ddots & \tilde{a}_{n-1,0} & \vdots \\ 0 & \cdots & 0 & \tilde{a}_{n,-\beta} & \cdots & \tilde{a}_{n,0} \end{pmatrix}$$

$c = Ab$ for Banded Matrix

Storing entries diagonalwise: $n(2\beta + 1)$ matrix instead of n^2.

$$\tilde{a}_{i,s} = a_{i,i+s} \quad \text{for row} \quad i = 1, \ldots, n$$

$$1 \leq i + s \leq n \quad \text{and} \quad -\beta \leq s \leq \beta \quad \text{and} \quad 1 \leq i \leq n$$

$$1 - i \leq s \leq n - i \quad \text{and} \quad -\beta \leq s \leq \beta$$

$$\downarrow \quad \textit{in row } i$$

$$s \in [l_i, r_i] = [\max\{-\beta, 1 - i\}, \min\{\beta, n - i\}]$$

$$1 - s \leq i \leq n - s \quad \text{and } 1 \leq i \leq n$$

$$\downarrow \quad \textit{in diag. } s$$

$$i \in [\tilde{l}_s, \tilde{r}_s] = [\max\{1, 1 - s\}, \min\{n, n - s\}]$$

Computation of the mtx-vec product based on storage scheme on vector CPUs

$$\text{For } i = 1, \ldots, n: \; c_i = A_{i\bullet} b = \sum_j a_{ij} b_j = \sum_{s=l_i}^{r_i} a_{i,i+s} b_{i+s} = \sum_{s=l_i}^{r_i} \tilde{a}_{i,s} b_{i+s}$$

- General TRIAD, no SAXPY:

```
for s = -β : β
    for i = max{1 - s, 1} : min{n - s, n}
        c_i = c_i + ã_{i,s} b_{i+s}
    end
end
```

- or, partial DOT-product:

```
for i = 1 : n
    for s = max{-β, 1 - i} : max{β, n - i}
        c_i = c_i + ã_{i,s} b_{i+s}
    end
end
```

- Sparsity $\Rightarrow$ less operations, but also loss of efficiency.

Band Ab in Parallel

- Partitioning:

$$\langle 1, n \rangle = \bigcup_{r=1}^{R} I_r, \text{ disjunct}$$

`for` $i \in I_r$

$$c_i = \sum_{s=l_i}^{r_i} \tilde{a}_{is} b_{i+s}$$

`end`

- Processor P_r gets rows to index set $I_r := [m_r, M_r]$ in order to compute its part of the final vector c.

- What part of vector b does processor P_r need in order to compute its part of c?

Band *Ab* in Parallel

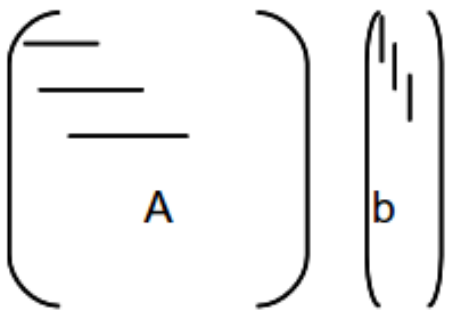
- Necessary for I_r: $b_j = b_{i+s}$:

$$j = i + s \geq m_r + l_{m_r} + \max\{-\beta, 1 - m_r\} = \max\{m_r - \beta, 1\}$$

$$j = i + s \leq M_r + r_{M_r} = M_r + \min\{\beta, n - M_r\} = \min\{M_r + \beta, n\}$$

- Processor P_r with index set I_r needs from b the indices

$$j \in [\max\{1, m_r - \beta\}, \min\{n, M_r + \beta\}]$$

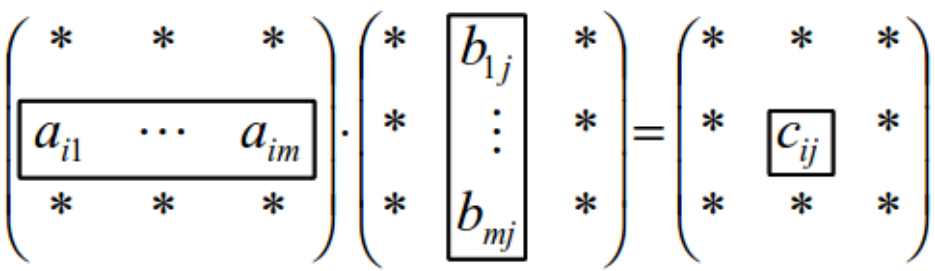


2.3. Analysis of Matrix-Matrix Product

$$A = (a_{ij})_{\substack{i=1,\dots,n \\ j=1,\dots,m}} \in \mathbb{R}^{n\times m}, \quad B(b_{ij})_{\substack{i=1,\dots,m \\ j=1,\dots,q}} \in \mathbb{R}^{m\times q},$$

$$C = AB = (c_{ij})_{\substack{i=1,\dots,n \\ j=1,\dots,q}} \in \mathbb{R}^{n\times q}$$

```
for i = 1 : n
    for j = 1 : q
        c_ij = sum_{k=1}^{m} a_ik b_kj
    end
end
```



2.3.1. Vectorization

- Algorithm 1: (ijk)-Form:

```
for i = 1 : n
    for j = 1 : q
        for k = 1 : m
            c_ij = c_ij + a_ik b_kj   } DOT-product of length m
        end
    end
end
```

$c_{ij} = A_{i\bullet} B_{\bullet j}$ for all i, j

- All entries c_{ij} are fully computed, one after another.
- Access to A and C is rowwise, to B columnwise (depends on inner most loops!)

Other View on the Matrix-Matrix Product

Matrix A considered as combination of columns or rows

$$A = A_1 e_1^T + \ldots + A_m e_m^T = (A_1 \;\; 0 \;\; \cdots) + (0 \;\; A_2 \;\; 0 \;\; \ldots) + \ldots + (\ldots \;\; 0 \;\; A_m)$$

$$= e_1 a_1 + \ldots + e_n a_n = \begin{pmatrix} a_1 \\ 0 \\ \vdots \end{pmatrix} + \ldots + \begin{pmatrix} \vdots \\ 0 \\ a_n \end{pmatrix}$$

$$AB = \sum_{j=1}^{n} A_j e_j^T \sum_{k=1}^{m} e_k b_k = \sum_{k,j} A_j (e_j^T e_k) b_k = \sum_{k=1}^{m} \underbrace{A_k b_k}_{\text{full } n \times q \text{ matrices}}$$

as a sum of full matrices $A_k b_k$ by outer product of the kth column of A and the kth row of B.

Algorithm 2: (jki)-Form

```
for j=1,...,q
    for k=1,...,m
        for i=1,...,n
            c_ij = c_ij + a_ik b_kj
        end
    end
end
```

SAXPY

GAXPY

- Vector update: $c_{\bullet j} = c_{\bullet j} + a_{\bullet k} b_{kj}$
- Sequence of SAXPYs for the same vector: $c_{\bullet j} = \sum_k b_{kj} a_{\bullet k}$
- C computed columnwise; access to A columnwise. Access to B columnwise, but delayed.

Algorithm 3: (kji)-Form

```
for k=1,...,m
    for j=1,...,q
        for i=1,...,n
            c_ij = c_ij + a_ik b_kj        } SAXPY
        end
    end
end
```

- Vector update: $c_{\bullet j} = c_{\bullet j} + a_{\bullet k} b_{kj}$
- Sequence of SAXPYs for different vectors $c_{\bullet j}$ (no GAXPY)
- Access to A columnwise. Access to B rowwise + delayed. C computed with intermediate values $c_{ij}^{(k)}$ which are computed columnwise.

Overview of Different Forms

	ijk Alg. 1	ikj	kij	jik	jki Alg. 2	kji Alg. 3
Access to A by	row	—	—	row	column	column
Access to B by	column	row	row	column	—	—
Comput. of C	row	row	row	column	column	column
Computation of c_{ij}	direct	delayed	delayed	direct	delayed	delayed
Vector operation	DOT	GAXPY	SAXPY	DOT	GAXPY	SAXPY
Vector length	m	q	q	m	n	n

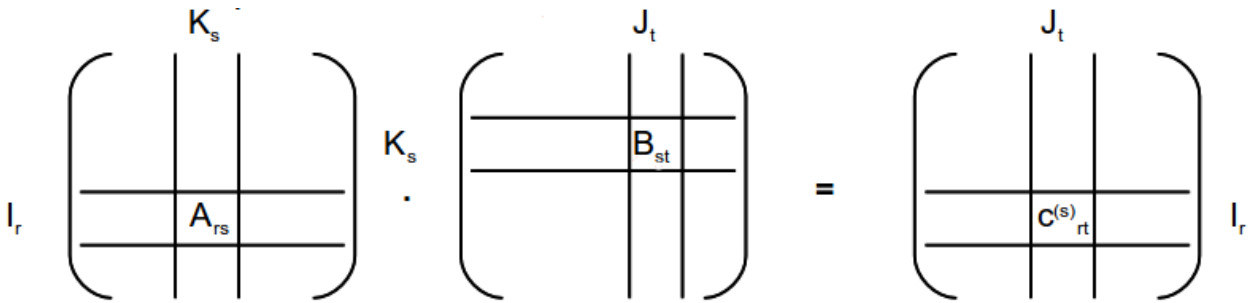
Better: GAXPY (longer vector length).

Access to matrices according to storage scheme (rowwise or columnwise)

2.3.2. Matrix-Matrix Product in Parallel

$$\langle 1, n \rangle = \bigcup_{r=1}^{R} I_r, \quad \langle 1, m \rangle = \bigcup_{s=1}^{S} K_s, \quad \langle 1, q \rangle = \bigcup_{t=1}^{T} J_t$$

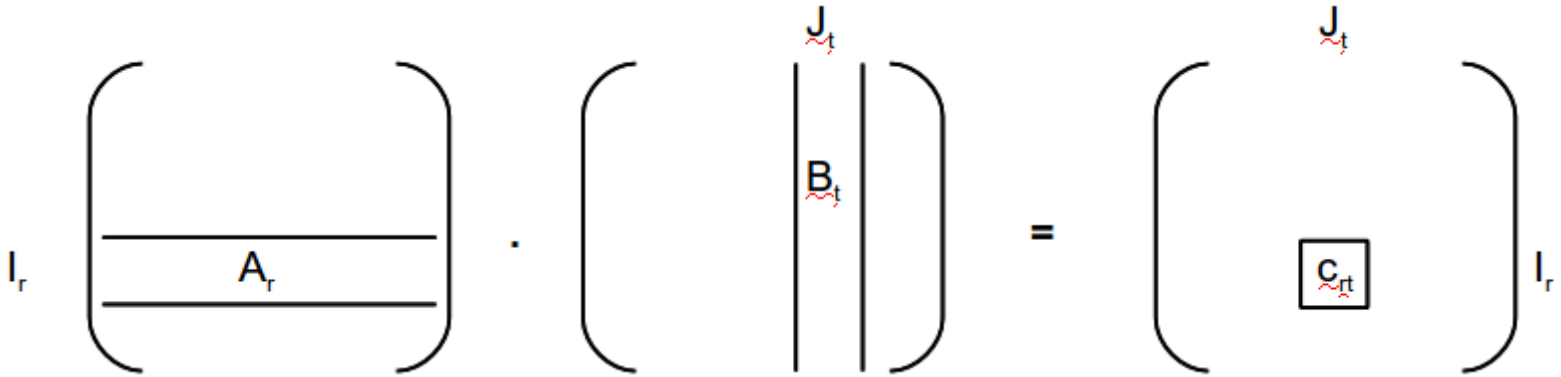
Distribute the blocks relative to index sets I_r, K_s, and J_t to processor array P_{rst}:



1. Processor P_{rst} computes small matrix-matrix product. All processors in parallel: $c_{rt}^{(s)} = A_{rs} B_{st}$
2. Compute sum by fan-in in s:

$$c_{rt} = \sum_{s=1}^{S} c_{rt}^{(s)}$$

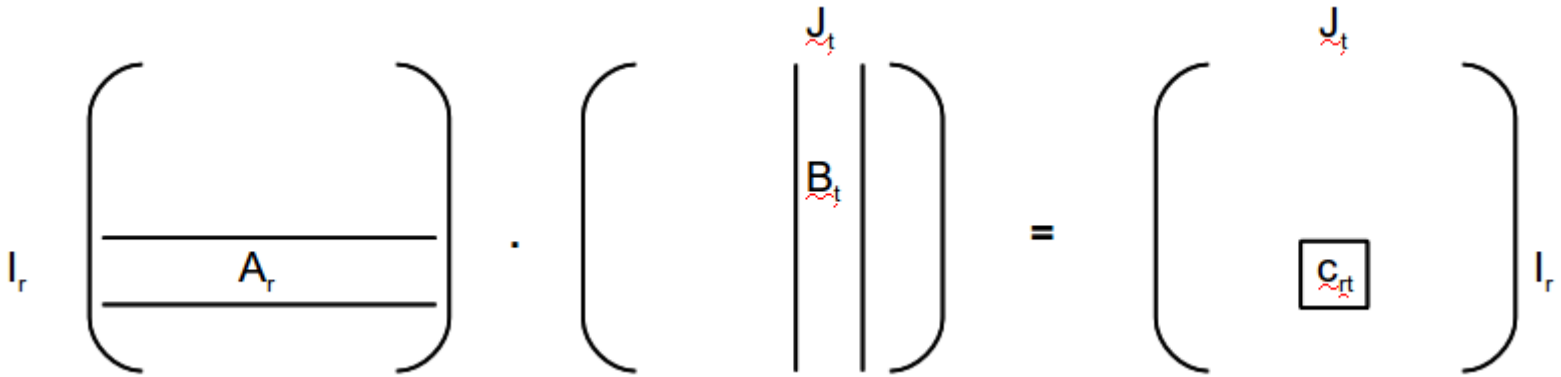
Mtx-Mtx in Parallel: Special Case $S = 1$



- Each processor P_{rt} can compute its part of c, c_{rt}, independently without communication.
- Each processor needs
  - full block of rows of A, relative to index set I_r, and
  - full block of columns of B, relative to index set J_t,

  to compute c_{rt} relative to rows I_k and columns J_t.

Mtx-Mtx in Parallel: Special Case $S = 1$



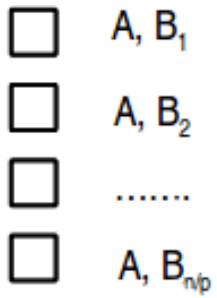
- With $n \cdot q$ processors each processor has to compute one DOT-product with $\mathcal{O}(m)$ parallel time steps.

$$c_{rt} = \sum_{k=1}^{m} a_{rk} b_{kt}$$

- Fan-in by $m \cdot nq$ additional processors for all DOT-products reduces number of parallel time steps to $\mathcal{O}(\log(m))$.

1D-Parallelization of $A \cdot B$

- 1D: p processors linear, each processor gets full A and column slice of B, computing the related column slice of $C = AB$

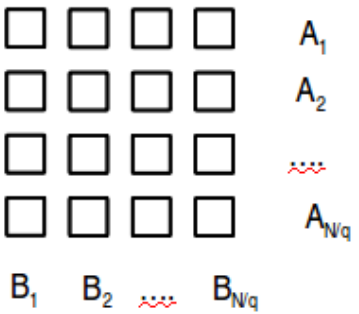


- Communication: $N^2 p$ for A and $(N \cdot \frac{N}{p}) \cdot p = N^2$ for B
- Granularity: $$\frac{N^3}{N^2(1+p)} = \frac{N}{1+p}$$
- Blocking only in i, the columns of B!

```
for i = 1 : n
    for j = 1 : n
        for k = 1 : n
            C_{j,i} = C_{j,i} + A_{j,k} B_{k,i}
```

2D-Parallelization of $A \cdot B$

- 2D: p processors square, $q := \sqrt{p}$, each proc. gets row slice of A and column slice of B computing full subblock of $C = AB$



- Communication: $N^2\sqrt{p}$ for A and $N^2\sqrt{p}$ for B
- Granularity: $$\frac{N^3}{2N^2\sqrt{p}} = \frac{N}{2\sqrt{p}}$$
- Blocking in i and j, the columns of B and the rows of A!

```
for i = 1 : n
    for j = 1 : n
        for k = 1 : n
            C_{j,i} = C_{j,i} + A_{j,k} B_{k,i}
```

3D-Parallelization $A \cdot B$

- 3D: p processors cubic, each processor gets subblock of A and subblock of B, computing part of subblock of $C = AB$.

  Additional fan-in to collect parts to full subblock of C. ($q = p^{\frac{1}{3}}$).

- Communication:

  $$N^2 p^{\frac{1}{3}} \text{ for } A \quad \text{and for } B \left(= p \cdot \frac{N^2}{p^{\frac{2}{3}}} = p \cdot \text{blocksize} \right), \text{ fan-in: } N^2 p^{\frac{1}{3}}$$

- Granularity: $\dfrac{N^3}{3N^2 p^{\frac{1}{3}}} = \dfrac{N}{3p^{\frac{1}{3}}}$

- Blocking in i, j, and k!

```
for i = 1 : n
    for j = 1 : n
        for k = 1 : n
            C_{j,i} = C_{j,i} + A_{j,k} B_{k,i}
```