Parallel Numerics, WT 2017/2018

2 Elementary Linear Algebra Problems

Contents

2.1. BLAS: Basic Linear Algebra Subroutines

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

2.2. 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	$\mathcal{O}(n^3)$

2.3. BLAS: Basic Linear Algebra Subroutines

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

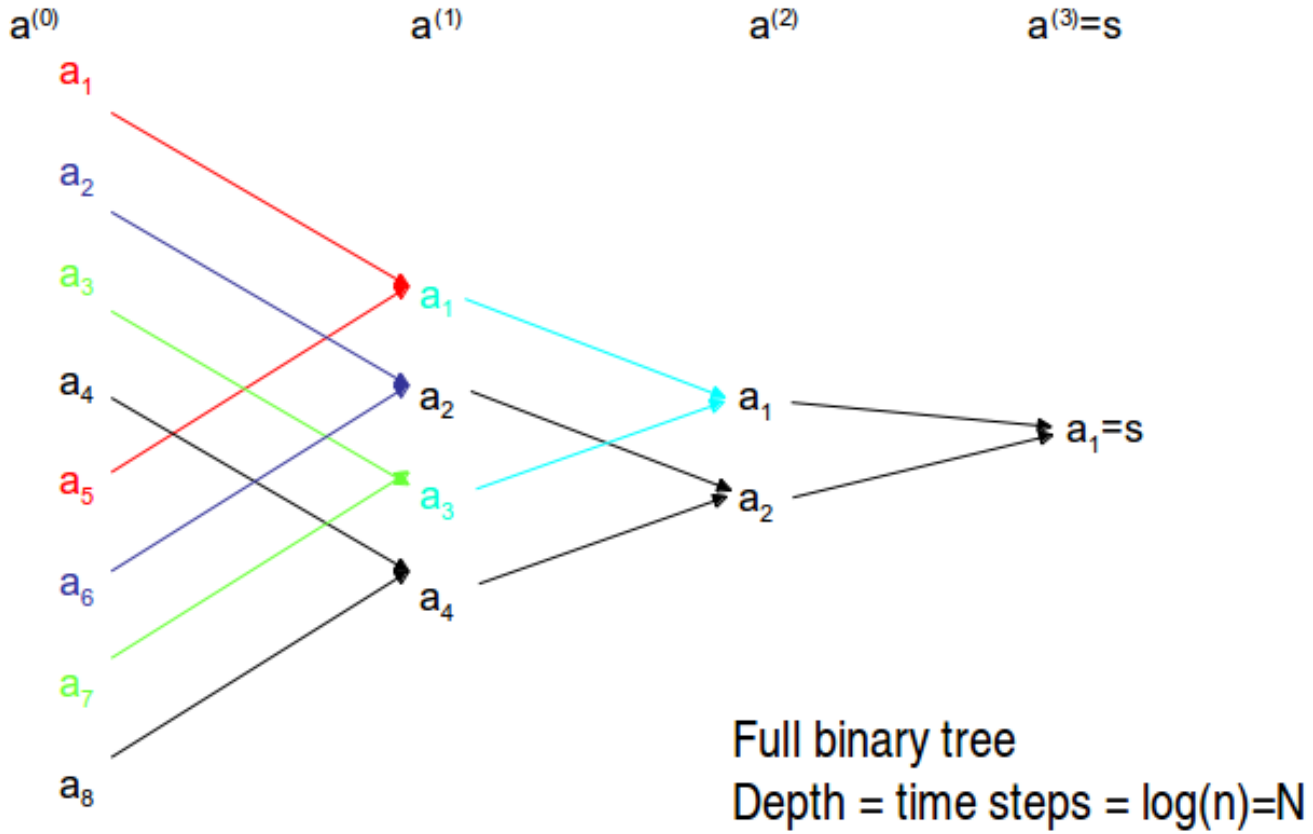
BLAS level	type of operation	complexity
BLAS-1	vector-vector	$\mathcal{O}(n)$
BLAS-2	matrix-vector	$\mathcal{O}(n^2)$
BLAS-3	matrix-matrix	$\mathcal{O}(n^3)$

- 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 $n = 2^N$

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

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

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

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

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

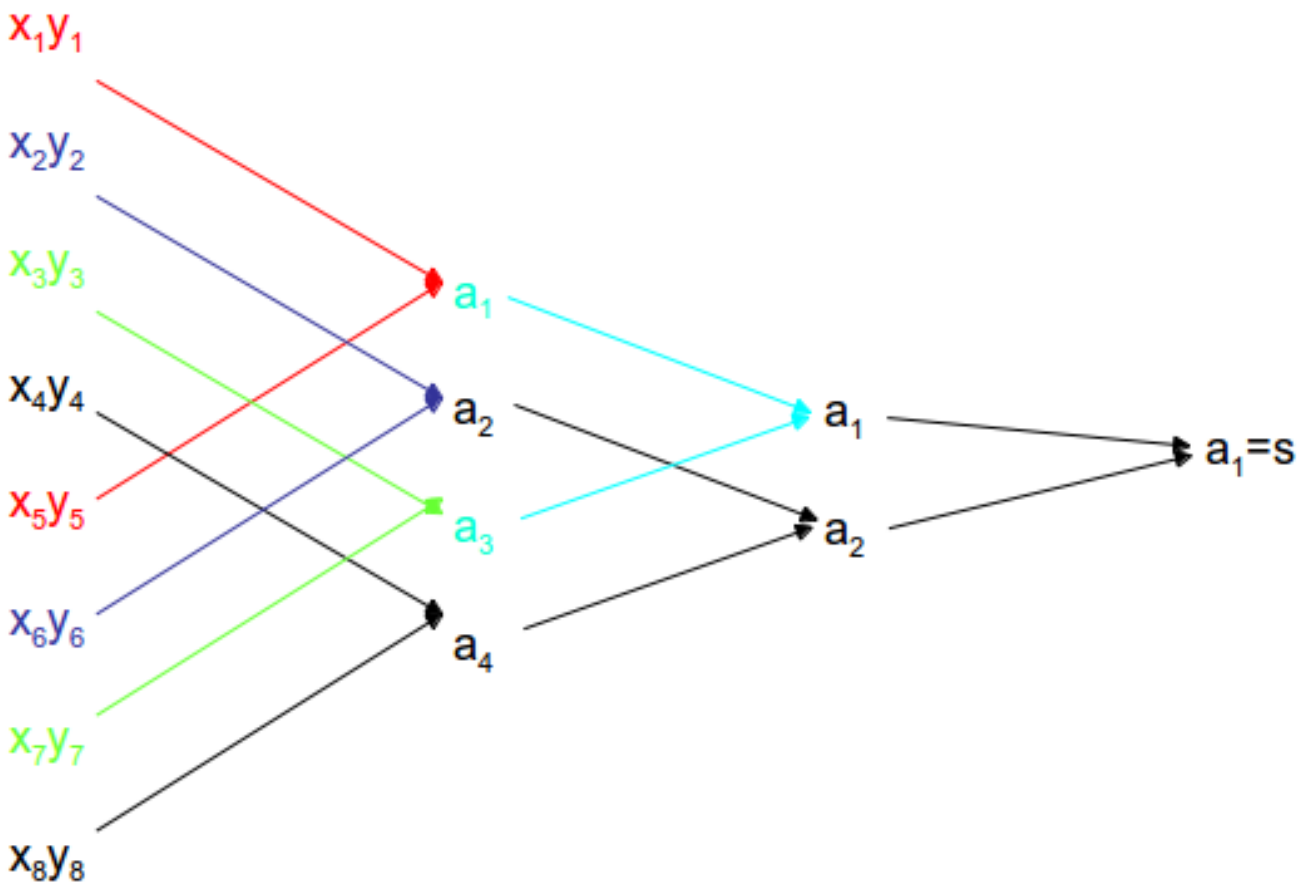
```
for(k=1;k<=N;k++)
   for(j=1;j<=2^{N-k};j++)
      a_j = a_j + a_{j+2^{N-k}}
   end
end
```

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

Level-1 BLAS

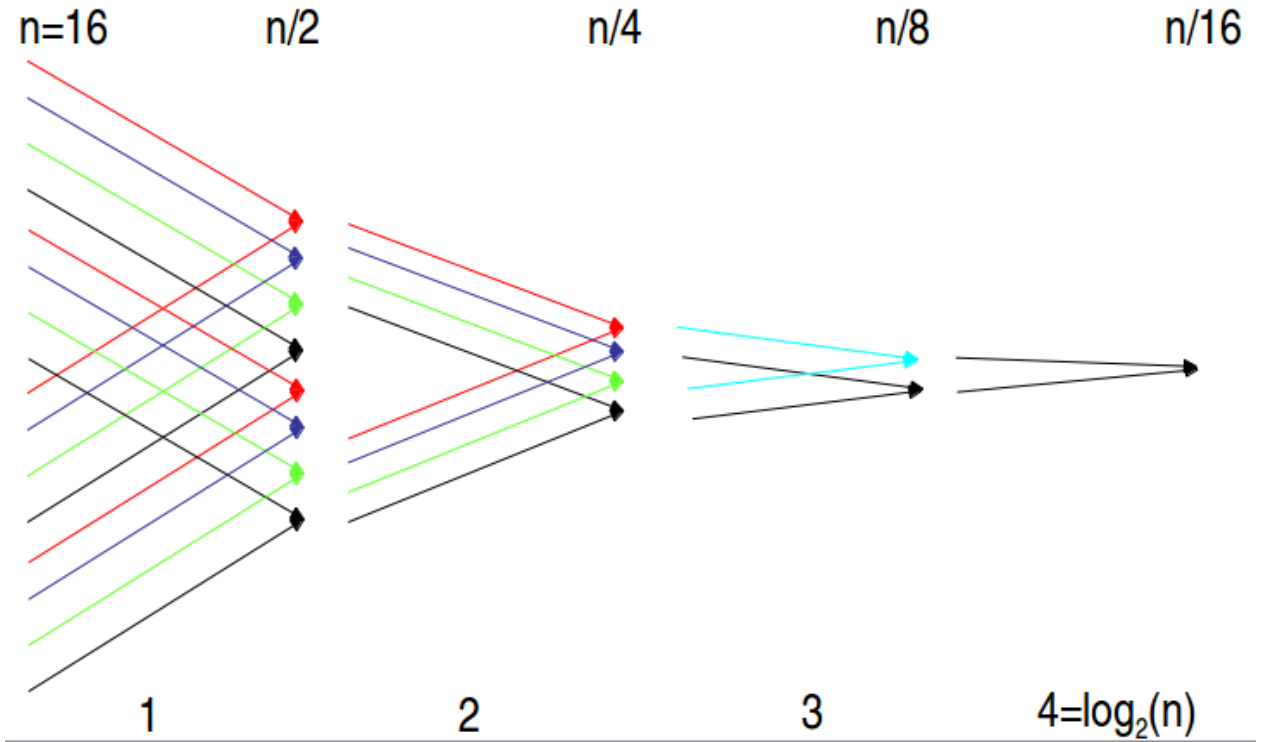
BLAS routines with $\mathcal{O}(n)$ problems (vectors only, $x, y \in \mathbb{R}^n$).

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: $y = \alpha x + y$

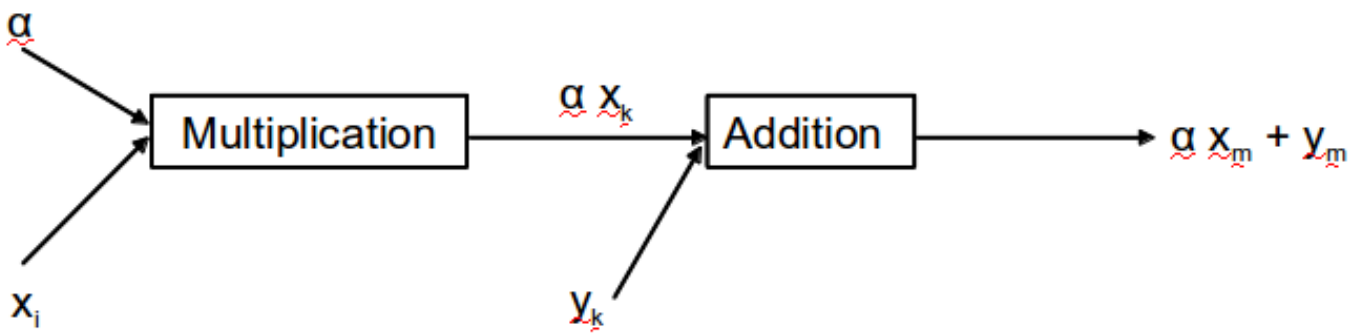
Level-1 BLAS: SAXPY

BLAS-Notation:

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

SAXPY: $y = \alpha x + y$

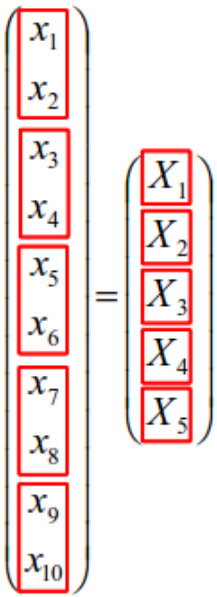
Vectorization of SAXPY ($\alpha x + y$) by pipelining:



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
BLAS-2	GEMV: $2n^2$	$\alpha Ax + \beta y$	n^2	2
BLAS-3	GEMM: $2n^3$	$\alpha AB + \beta C$	$4n^2$	$\frac{n}{2}$

Note that the numbers in the table are $O(n)$ and memory means load+store.
BLAS-3 has best operations to memory ratio!

2.4. Analysis of the Matrix-Vector-Product

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

2.4.1. Vectorization

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

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

2.5. Analysis of the Matrix-Vector-Product

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

2.5.1. 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

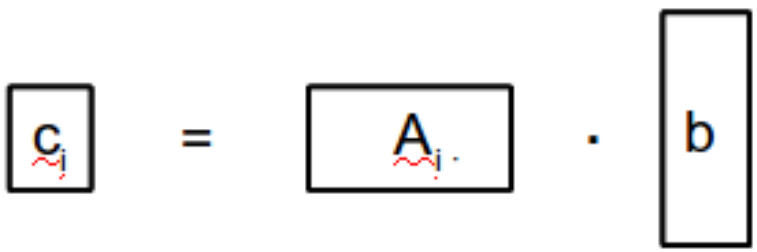
$c_i = A_{i\bullet} b$, DOT-product of ith row of A with vector b

Pseudocode: *ij*-form

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

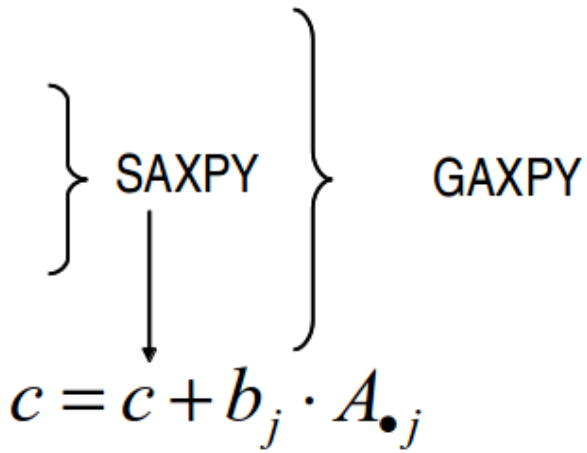
} DOT-product

$c_i = A_{i\bullet} b$, 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` $i = 1 : n$

  $y := y + \alpha_i x_i$

  `end`

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

2.5.2. 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 $j \neq k$

2.5.3. Parallelization by Building Blocks

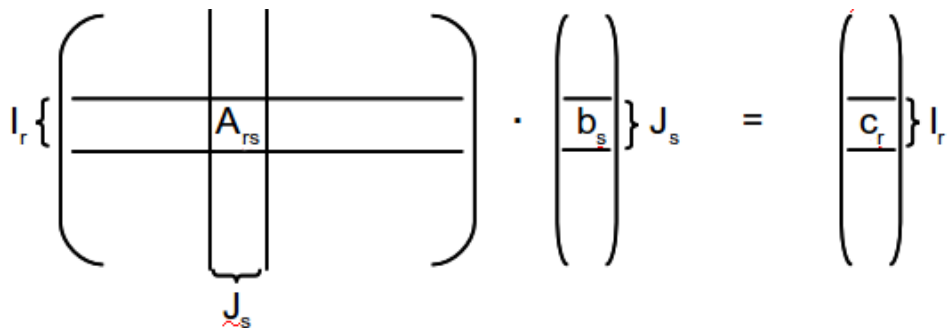
Reduce matrix-vector product on smaller matrix-vector products.

$\{1, 2, \ldots, n\} = \langle 1, n \rangle = I_1 \cup I_2 \cup \ldots \cup I_R$ 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 $P_1, \ldots, P_R$

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.

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

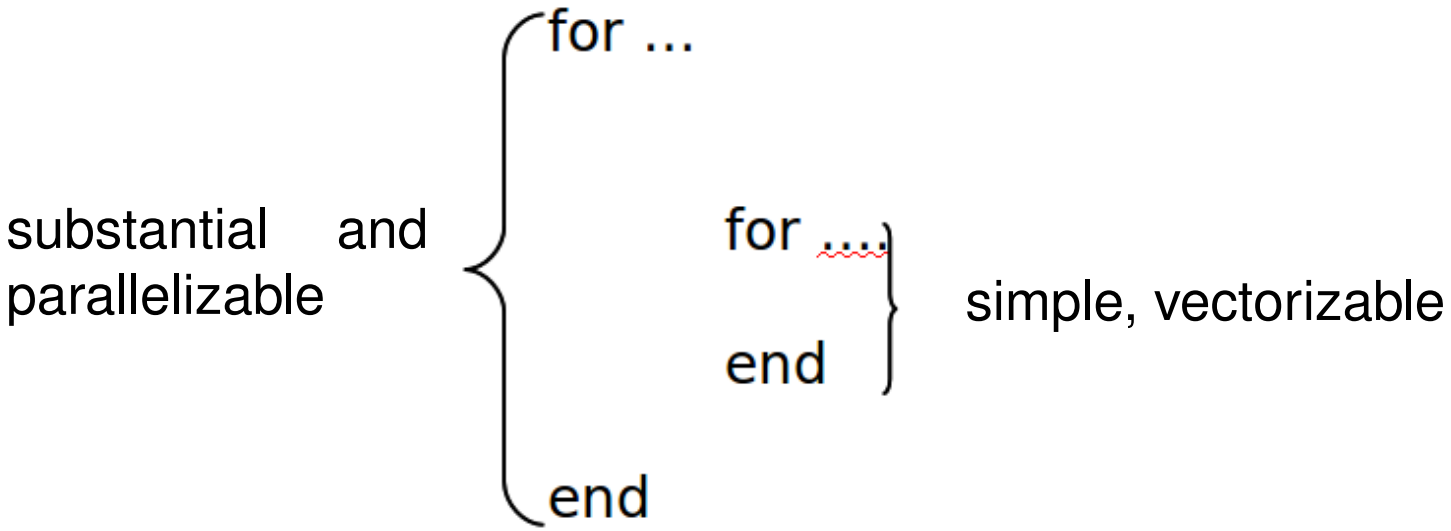
Then collection of partial results from processor $P_1, \ldots, P_S$. Fan-in.
Final sum in one processor: vectorizable by GAXPYs.

Rules

1. Inner loops of a program should be simple, vectorizable

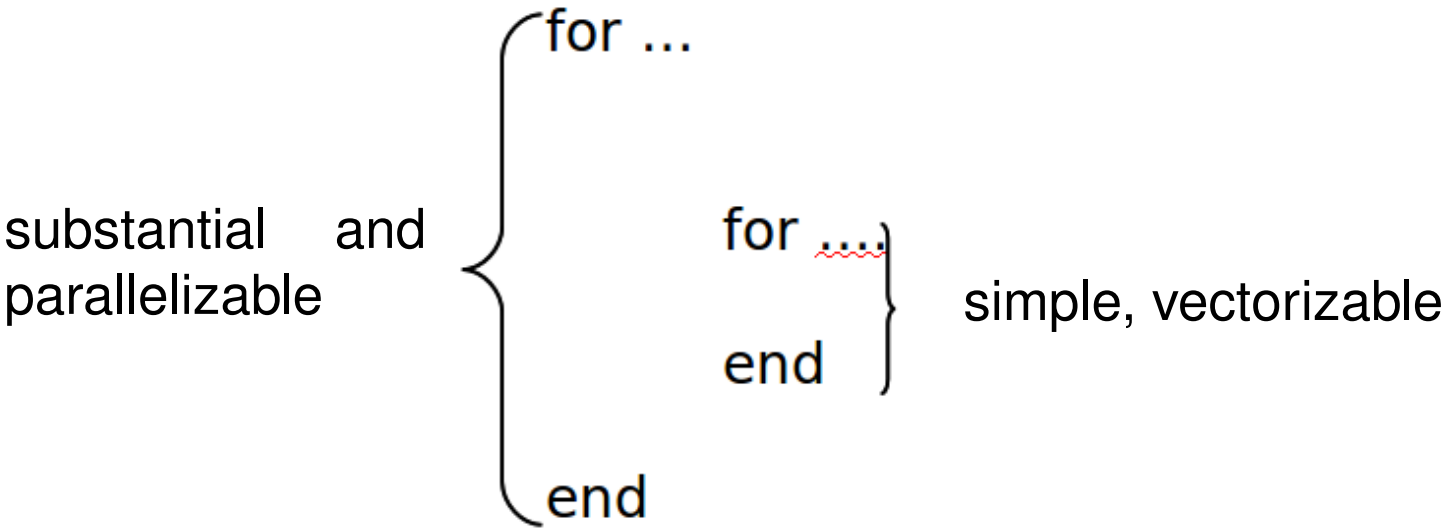
Rules

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



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)