

Parallel Numerics, WT 2015/2016

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
 - 6.1 Overlapping Domain Decomposition
 - 6.2 Non-overlapping Domain Decomposition
 - 6.3 Schur Complements



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

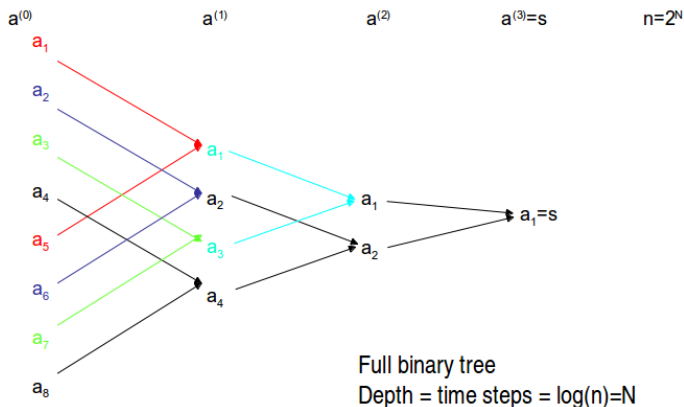
- There are machine-specific optimized BLAS libraries
- Basis of other libraries, e.g., LAPACK (Linear Algebra Package: for solving linear equations, least squares problems, QR-decomposition, eigenvalues, singular values...)



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

$$n = 2^N$$

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

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



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

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

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

```

for(k=1;k<=N;k++)
  for(j=1;j<=2N-k;j++)
    aj = aj + aj+2N-k
  end
end

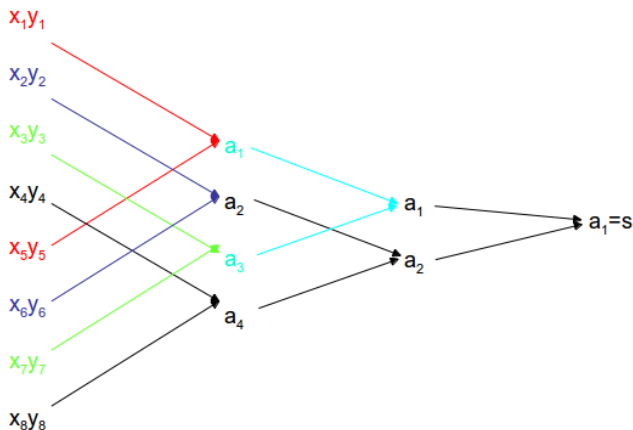
```

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

Level-1 BLAS

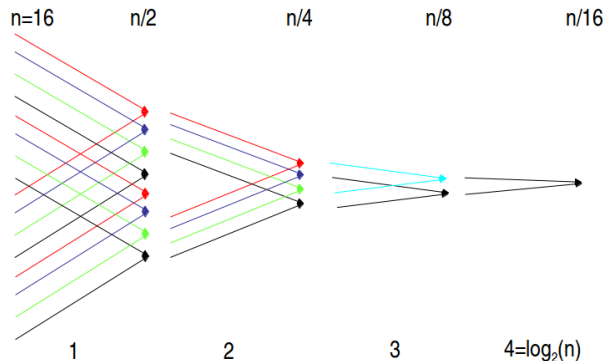
BLAS routines with $\mathcal{O}(n)$ problems (vectors only, $x, y \in \mathbb{R}^n$).

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



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

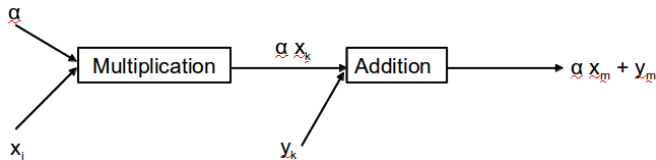
X vector

P plus operation

Y vector

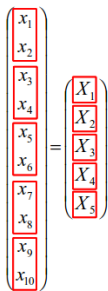
SAXPY: $y = \alpha x + y$

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



SAXPY Parallelization by Partitioning

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

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


- 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 $Y_j = \alpha X_j + Y_j$, $j = 1, 2, \dots, R$.
- 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

V

} general matrix

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 } general matrix

M

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!



2.4. Analysis of the Matrix-Vector-Product

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

2.4.1. Vectorization

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



2.5. Analysis of the Matrix-Vector-Product

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

2.5.1. Vectorization

$$\begin{aligned} \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix} &= \begin{pmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix} = \begin{pmatrix} a_{11}b_1 + \cdots + a_{1m}b_m \\ \vdots \\ a_{n1}b_1 + \cdots + a_{nm}b_m \end{pmatrix} = \\ &= \begin{pmatrix} \sum_{j=1}^m a_{1j}b_j \\ \vdots \\ \sum_{j=1}^m a_{nj}b_j \end{pmatrix} = \sum_{j=1}^m b_j \begin{pmatrix} a_{1j} \\ \vdots \\ a_{nj} \end{pmatrix} \end{aligned}$$

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 *i*th 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

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

$$\boxed{c_i} = \boxed{A_{i\bullet}} \cdot \boxed{b}$$



Pseudocode: *ji*-form

```

c = 0;
for j=1,...,m
  for i=1,...,n
    ci = ci + aijbj
  end
end
end

```

$$c = c + b_j \cdot A_{\bullet j}$$

- 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 .
- $\text{length}(\text{GAXPY}) = \text{length}(y)$
- Advantage: less data transfer!



2.5.2. Parallelization by Building Blocks

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

$$\{1, 2, \dots, n\} = \langle 1, n \rangle = I_1 \cup I_2 \cup \dots \cup I_R \text{ disjoint: } I_j \cap I_k = \emptyset \text{ for } j \neq k$$

$$\{1, 2, \dots, m\} = \langle 1, m \rangle = J_1 \cup J_2 \cup \dots \cup J_S \text{ disjoint: } J_j \cap J_k = \emptyset \text{ for } j \neq k$$



Pseudocode

```

for  $r = 1, \dots, R$ 
  for  $s = 1, \dots, S$ 
     $c_r^{(s)} = A_{rs} b_s$ ;
  end
end

```

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

```

for  $r = 1, \dots, R$ 
   $c_r = 0$ 
  for  $s = 1, \dots, S$ 
     $c_r = c_r + c_r^{(s)}$ ;
  end
end

```

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



Blocking: Special Cases

$S = 1$: The computation of $A_i \cdot b$ is vectorizable by GAXPYs.

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

No communication necessary between processor P_1, \dots, P_R



Blocking: Special Cases

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

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

No communication necessary between processor P_1, \dots, P_R

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

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

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



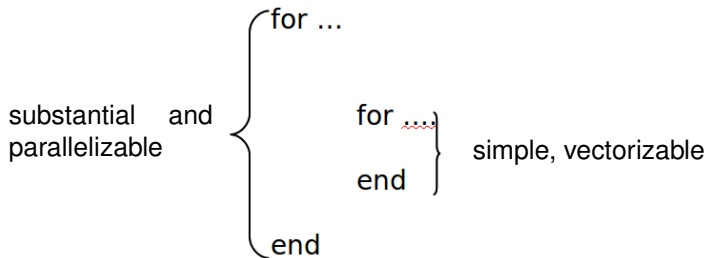
Rules

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



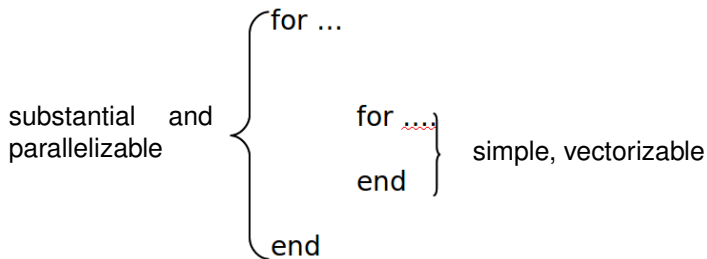
Rules

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



Rules

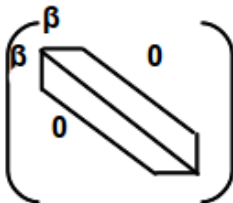
1. Inner loops of a program should be simple, vectorizable
2. Outer loop of a program should be substantial, independent, parallelizable.



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



2.5.4. $c = Ab$ for Banded Matrix



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

Notation: Banded Matrices A and \tilde{A}

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



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 .

$$\tilde{a}_{i,s} = a_{i,i+s} \quad \text{for row } i = 1, \dots, n$$
$$1 \leq i + s \leq n \quad \text{and} \quad -\beta \leq s \leq \beta \quad \text{and} \quad 1 \leq i \leq n$$



$c = Ab$ for Banded Matrix

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

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

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

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

↓ in row i

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



$c = Ab$ for Banded Matrix

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

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

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

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

↓ *in row i*

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

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

↓ *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

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

- General TRIAD, no SAXPY:

```
for s = -β : β
```

```
    for i = max{1 - s, 1} : min{n - s, n}
```

```
        c_i = c_i + \tilde{a}_{i,s} b_{i+s}
```

```
    end
```

```
end
```



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

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

- General TRIAD, no SAXPY:

```
for s = -β : β
```

```
  for i = max{1 - s, 1} : min{n - s, n}
```

```
    c_i = c_i + \tilde{a}_{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 + \tilde{a}_{i,s} b_{i+s}
```

```
  end
```

```
end
```

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



Band Ab in Parallel

- Partitioning:

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

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 l_r : $b_j = b_{i+s}$:

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

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

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

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

$$\left(\begin{array}{c} \text{---} \\ \text{---} \\ \text{---} \\ \text{A} \end{array} \right) \left(\begin{array}{c} | \\ | \\ | \\ \text{b} \end{array} \right)$$



2.6. 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

$$\begin{pmatrix} * & * & * \\ a_{i1} & \cdots & a_{im} \\ * & * & * \end{pmatrix} \cdot \begin{pmatrix} * & \boxed{b_{1j}} & * \\ * & \vdots & * \\ * & \boxed{b_{mj}} & * \end{pmatrix} = \begin{pmatrix} * & * & * \\ * & \boxed{c_{ij}} & * \\ * & * & * \end{pmatrix}$$



2.6.1. Vectorization

- Algorithm 1: (ijk)-Form:

```
for  $i = 1 : n$ 
```

```
  for  $j = 1 : q$ 
```

```
    for  $k = 1 : m$ 
```

```
       $c_{ij} = c_{ij} + a_{ik} b_{kj}$ 
```

```
    end
```

```
  end
```

```
end
```

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

} DOT-product of length m

- 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**

$$\begin{aligned}
 A &= A_1 e_1^T + \dots + A_m e_m^T = (A_1 \ 0 \ \dots) + (0 \ A_2 \ 0 \ \dots) + \dots + (\dots \ 0 \ A_m) \\
 &= e_1 a_1 + \dots + e_n a_n = \begin{pmatrix} a_1 \\ 0 \\ \vdots \end{pmatrix} + \dots + \begin{pmatrix} \vdots \\ 0 \\ a_n \end{pmatrix}
 \end{aligned}$$

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

as a sum of full matrices $A_k b_k$ by outer product of the k th column of A and the k th 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
end

```

$\left. \begin{array}{l} \text{SAXPY} \\ \text{GAXPY} \end{array} \right\}$

- 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
       $c_{ij} = c_{ij} + a_{ik}b_{kj}$ 
    end
  end
end
end

```

} SAXPY

- 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
Computat ion of c_{ij}	direct	delayed	delayed	direct	delayed	delayed
Vector ope- ration	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)



2.6.2. 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} :

$$I_r \left(\begin{array}{c|c|c} & & \\ \hline & & \\ \hline & A_{rs} & \\ \hline & & \end{array} \right)_{K_s} \cdot \left(\begin{array}{c|c|c} & & \\ \hline & B_{st} & \\ \hline & & \end{array} \right)_{J_t} = \left(\begin{array}{c|c|c} & & \\ \hline & C_{rt}^{(s)} & \\ \hline & & \end{array} \right)_{J_t} I_r$$

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 :

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



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 $C = AB$

A, B_1

A, B_2

.....

A, B_{np}

- 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 $j = 1 : n$

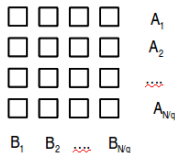
 for $k = 1 : n$

$$C_{j,i} = C_{j,i} + A_{j,k} B_{k,i}$$



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:

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

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

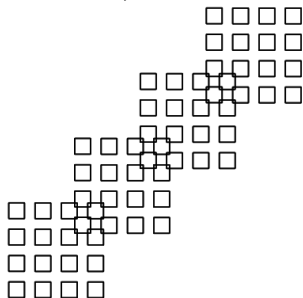
- Blocking in i, j , and k !

for $i = 1 : n$

 for $j = 1 : n$

 for $k = 1 : n$

$$C_{j,i} = C_{j,i} + A_{j,k} B_{k,i}$$



Parallel Numerics, WT 2015/2016

3 Linear Systems of Equations with Dense Matrices



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
 - 6.1 Overlapping Domain Decomposition
 - 6.2 Non-overlapping Domain Decomposition
 - 6.3 Schur Complements



3.1. Linear Systems of Equations with Dense Matrices

3.1.1. Gaussian Elimination: Basic Properties

- Linear system of equations:

$$\begin{aligned} a_{11}x_1 + \dots + a_{1n}x_n &= b_1 \\ &\vdots \\ a_{n1}x_1 + \dots + a_{nn}x_n &= b_n \end{aligned}$$

- Solve $Ax = b$

$$\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}$$

- Generate simpler linear equations (matrices). Transform A in triangular form: $A = A^{(1)} \rightarrow A^{(2)} \rightarrow \dots \rightarrow A^{(n)} = U$.



Transformation to Upper Triangular Form

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}$$

row transformations: $(2) \rightarrow (2) - \frac{a_{21}}{a_{11}} \cdot (1), \dots, (n) \rightarrow (n) - \frac{a_{n1}}{a_{11}} \cdot (1)$
leads to

$$A^{(2)} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \cdots & a_{2n}^{(2)} \\ 0 & a_{32}^{(2)} & a_{33}^{(2)} & \cdots & a_{3n}^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & a_{n2}^{(2)} & a_{n3}^{(2)} & \cdots & a_{nn}^{(2)} \end{pmatrix}$$

next transformations: $(3) \rightarrow (3) - \frac{a_{32}^{(2)}}{a_{22}^{(2)}} \cdot (2), \dots, (n) \rightarrow (n) - \frac{a_{n2}^{(2)}}{a_{22}^{(2)}} \cdot (2)$



Transformation to Triangular Form (cont.)

$$A^{(3)} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \cdots & a_{2n}^{(2)} \\ 0 & 0 & a_{33}^{(3)} & \cdots & a_{3n}^{(3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & a_{n3}^{(3)} & \cdots & a_{nn}^{(3)} \end{pmatrix}$$

next transformations: $(4) \rightarrow (4) - \frac{a_{43}^{(3)}}{a_{33}^{(3)}} \cdot (3)$, \dots , $(n) \rightarrow (n) - \frac{a_{n3}^{(3)}}{a_{33}^{(3)}} \cdot (3)$

$$A^{(n)} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \cdots & a_{2n}^{(2)} \\ 0 & 0 & a_{33}^{(3)} & \cdots & a_{3n}^{(3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_{nn}^{(n)} \end{pmatrix} = U$$



Pseudocode Gaussian Elimination (GE)

Simplification: assume that no pivoting is necessary.

$$a_{kk}^{(k)} \neq 0 \quad \text{or} \quad |a_{kk}^{(k)}| \geq \rho > 0 \quad \text{for} \quad k = 1, 2, \dots, n$$

```

for  $k = 1 : n - 1$ 
  for  $i = k + 1 : n$ 
     $l_{i,k} = \frac{a_{i,k}}{a_{k,k}}$ 
  end
  for  $i = k + 1 : n$ 
    for  $j = k + 1 : n$ 
       $a_{i,j} = a_{i,j} - l_{i,k} \cdot a_{k,j}$ 
    end
  end
end
end
  
```

In practice:

- Include pivoting and include right hand side b .
- There is still to solve a triangular system in U !



Intermediate Systems

$$A^{(k)}, k = 1, 2, \dots, n \quad \text{with} \quad A = A^{(1)} \quad \text{and} \quad U = A^{(n)}$$

$$\begin{pmatrix} a_{11}^{(1)} & \cdots & a_{1,k-1}^{(1)} & a_{1,k}^{(1)} & \cdots & a_{1,n}^{(1)} \\ 0 & \ddots & \vdots & \vdots & \ddots & \vdots \\ \vdots & \ddots & a_{k-1,k-1}^{(k-1)} & a_{k-1,k}^{(k-1)} & \cdots & a_{k-1,n}^{(k-1)} \\ 0 & \cdots & 0 & a_{k,k}^{(k)} & \cdots & a_{k,n}^{(k)} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & a_{n,k}^{(k)} & \cdots & a_{n,n}^{(k)} \end{pmatrix}$$

Main part of $A^{(k)}$ that will be used and changed in the following computations.



Define Auxiliary Matrices

$$L = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ l_{2,1} & 1 & \ddots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ l_{n,1} & \cdots & l_{n,n-1} & 1 \end{pmatrix} \quad \text{and} \quad U = A^{(n)}$$

$$L_k := \begin{pmatrix} 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & l_{k+1,k} & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & l_{n,k} & 0 & \cdots & 0 \end{pmatrix}, \quad L = I + \sum_k L_k$$



Elimination Step in Terms of Auxiliary Matrices

$$A^{(k+1)} = (I - L_k) \cdot A^{(k)} = A^{(k)} - L_k \cdot A^{(k)}$$

$$U = A^{(n)} = (I - L_{n-1}) \cdot A^{(n-1)} = \dots = (I - L_{n-1}) \cdots (I - L_1) A^{(1)} = \tilde{L} \cdot A$$

$$\tilde{L} := (I - L_{n-1}) \cdots (I - L_1)$$

$$A = \tilde{L}^{-1} \cdot U \quad \text{with } U \text{ upper triangular and } \tilde{L} \text{ lower triangular}$$



Elimination Step in Terms of Auxiliary Matrices

$$A^{(k+1)} = (I - L_k) \cdot A^{(k)} = A^{(k)} - L_k \cdot A^{(k)}$$

$$U = A^{(n)} = (I - L_{n-1}) \cdot A^{(n-1)} = \dots = (I - L_{n-1}) \cdots (I - L_1) A^{(1)} = \tilde{L} \cdot A$$

$$\tilde{L} := (I - L_{n-1}) \cdots (I - L_1)$$

$$A = \tilde{L}^{-1} \cdot U \quad \text{with } U \text{ upper triangular and } \tilde{L} \text{ lower triangular}$$

- **Theorem 2:** $\tilde{L}^{-1} = L$ and therefore $A = LU$.
- Advantage: Every further problem $Ax = b_j$ can be reduced to $(LU)x = b_j$ for arbitrary j .
- Solve two triangular problems $(LU)x = Ly = b$ and $Ux = y$.



3.2. GE in Parallel: Blockwise

Main idea: Blocking of GE to avoid data transfer between processors.

Basic Concepts:

Replace GE or large LU -decomposition of full matrix by small intermediate steps (by sequence of small block operations):

- Solving collection of small triangular systems $LU_k = B_k$ (parallelism in columns of U)
- $A \rightarrow A - LU$ updating matrices (also easy to parallelize)
- small $B = LU$ -decompositions (parallelism in rows of B)

How to Choose Blocks in L , resp. U Satisfying $LU = A$

$$\begin{pmatrix} L_{11} & 0 & 0 \\ L_{21} & L_{22} & 0 \\ L_{31} & L_{32} & L_{33} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} & U_{13} \\ 0 & U_{22} & U_{23} \\ 0 & 0 & U_{33} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} =$$

$$= \begin{pmatrix} L_{11}U_{11} & L_{11}U_{12} & L_{11}U_{13} \\ L_{21}U_{11} & L_{21}U_{12} + L_{22}U_{22} & L_{21}U_{13} + L_{22}U_{23} \\ L_{31}U_{11} & L_{31}U_{12} + L_{32}U_{22} & * \end{pmatrix}$$



How to Choose Blocks in L , resp. U Satisfying $LU = A$

$$\begin{pmatrix} L_{11} & 0 & 0 \\ L_{21} & L_{22} & 0 \\ L_{31} & L_{32} & L_{33} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} & U_{13} \\ 0 & U_{22} & U_{23} \\ 0 & 0 & U_{33} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} =$$

$$= \begin{pmatrix} L_{11}U_{11} & L_{11}U_{12} & L_{11}U_{13} \\ L_{21}U_{11} & L_{21}U_{12} + L_{22}U_{22} & L_{21}U_{13} + L_{22}U_{23} \\ L_{31}U_{11} & L_{31}U_{12} + L_{32}U_{22} & * \end{pmatrix}$$

Different ways of computing L and U depending on

- start (assume first entry/row/column of L/U as given)
- how to compute new entry/row/column of L/U
- update of block structure of L/U by grouping in
 - known blocks
 - blocks newly to compute
 - blocks to be computed later



Crout Form

$$\begin{pmatrix} L_{11} & & \\ L_{21} & L_{22} & \\ L_{31} & L_{32} & * \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} & U_{13} \\ & U_{22} & U_{23} \\ & & * \end{pmatrix} = A$$

Already computed To compute in this step

$$\begin{pmatrix} L_{22}U_{22} & L_{22}U_{23} \\ L_{32}U_{22} & * \end{pmatrix} = \begin{pmatrix} A_{22} - L_{21}U_{12} & A_{23} - L_{21}U_{13} \\ A_{32} - L_{31}U_{12} & * \end{pmatrix} = \begin{pmatrix} \hat{A}_{22} & \hat{A}_{23} \\ \hat{A}_{32} & * \end{pmatrix}$$

Leads to two subproblems in and in



Crout Form (cont.)

1. Solve

$$\begin{pmatrix} L_{22} \\ L_{32} \end{pmatrix} \cdot U_{22} = \begin{pmatrix} \hat{A}_{22} \\ \hat{A}_{32} \end{pmatrix}$$

by small LU -decomposition of the modified part of $A \rightarrow L_{22}, L_{32}$, and U_{22} .

2. Solve

$$L_{22} \cdot U_{23} = \hat{A}_{23}$$

by solving small triangular systems of equations in $L_{22} \rightarrow U_{23}$.



Crout Form (cont.)

1. Solve

$$\begin{pmatrix} L_{22} \\ L_{32} \end{pmatrix} \cdot U_{22} = \begin{pmatrix} \hat{A}_{22} \\ \hat{A}_{32} \end{pmatrix}$$

by small LU -decomposition of the modified part of $A \rightarrow L_{22}, L_{32}$, and U_{22} .

2. Solve

$$L_{22} \cdot U_{23} = \hat{A}_{23}$$

by solving small triangular systems of equations in $L_{22} \rightarrow U_{23}$.

Initial steps:

$$L_{11} U_{11} = A_{11}, \quad \begin{pmatrix} L_{21} \\ L_{31} \end{pmatrix} U_{11} = \begin{pmatrix} A_{21} \\ A_{31} \end{pmatrix}, \quad L_{11}(U_{12} \ U_{13}) = (A_{12} \ A_{13})$$



New Partitioning

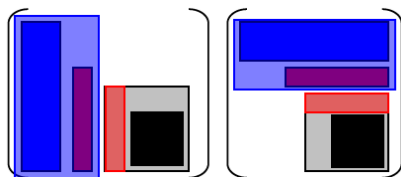
$$A = \left(\begin{array}{ccc|c} \boxed{\begin{matrix} L_{11} & & \\ L_{21} & L_{11} & L_{22} \end{matrix}} & & & \\ \hline \boxed{\begin{matrix} L_{31,1} & L_{21} & L_{32,1} \\ L_{31,2} & L_{32,2} \end{matrix}} & \boxed{\begin{matrix} L_{33,2} \\ L_{33,1} \end{matrix}} & & \\ \hline & & & L_{33,new} \end{array} \right) \cdot \left(\begin{array}{cc|cc} \boxed{\begin{matrix} U_{11} & U_{12} \\ U_{22} \end{matrix}} & & \boxed{\begin{matrix} U_{13,1} \\ U_{23,1} \end{matrix}} & \boxed{\begin{matrix} U_{13,2} \\ U_{23,2} \end{matrix}} \\ \hline & & \boxed{\begin{matrix} U_{33,1} \\ U_{33,2} \end{matrix}} & \boxed{\begin{matrix} U_{33,2} \\ U_{33,1} \end{matrix}} \\ \hline & & & U_{33,new} \end{array} \right)$$

- Combine already computed parts from second column of L and second row of U into first column of L and first row of U .
- Split the until now ignored parts L_{33} and U_{33} into new columns/rows.
- Repeat this overall procedure until L and U are fully computed.



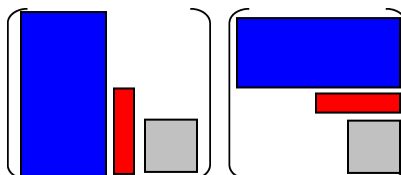
Block Structure

Intermediate block structure:



Solve for red blocks.

Reconfigure the block structure:



Repeat until done.