TEMPUS: Activity 2

Parallel Algorithms, Chapter 2
Numerical and Matrix Algorithms

I. Vassányi, G. Dózsa, KFKI-MSZKI

20th July 1996

Overview

1 Introduction
- Notations and Terminology

2 Algorithms for General Dense Matrices
- Transposition
- Matrix-vector Multiplication
- Matrix Multiplication
- Matrix Inversion
- Solving a Linear System and the LU Decomposition
- Inversion-based Solution of Linear Systems

3 Algorithms for Special Matrices
- The Finite Element Method
- Multiplication of a Block-tridiagonal Matrix by a Vector
- Jacobi Iteration for Solving Linear Systems
- Matrix-vector Product for Toeplitz Matrices

4 Further Reading

1 Notations and Terminology

Matrices, vectors (colums and rows)

Processing Elements

Cost-optimality of algorithms (simplifying assumptions)

P	**No PEs**
T_P	**Parallel runtime**
T_S	**Serial runtime**
$C=PT_P$	**Cost**

2 Algorithms for General Dense Matrices

Partitioning

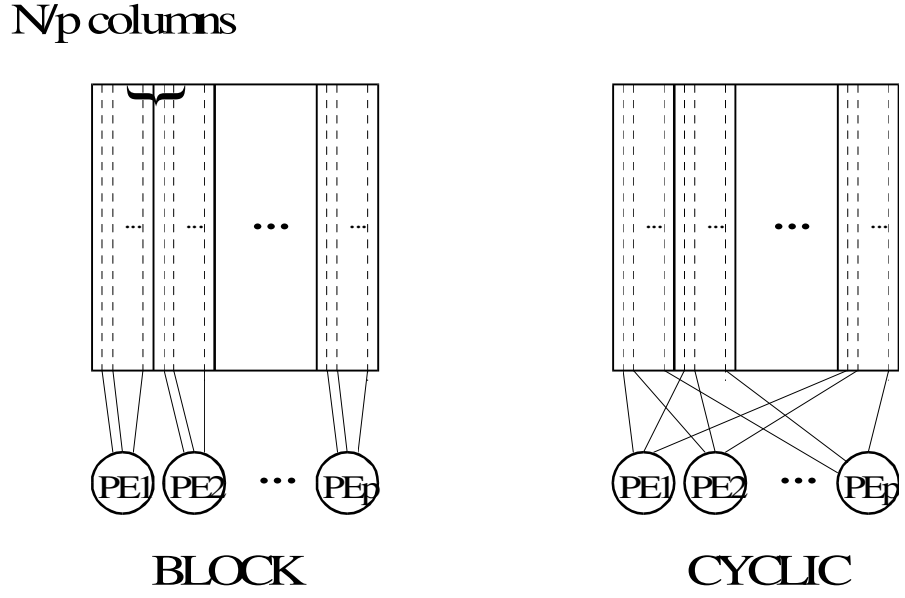
A) <u>Striped</u> —row-wise (block or cyclic)

—column-wise (block or cyclic)

—max N PEs

B) <u>Checkerboard</u> (block or cyclic)

—max N^2 PEs (finer granularity)



Example: Block and cyclic column-wise striping

2 Algorithms for General Dense Matrices
2.1 TRANSPOSITION

Problem $a_{i,j} := a_{j,i}$

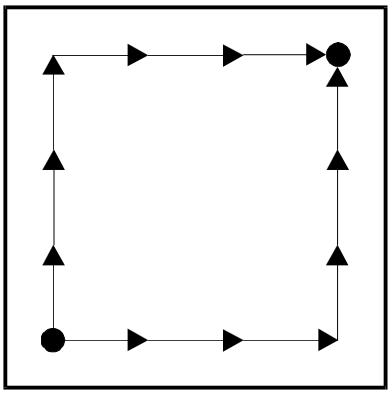
Serial Algorithm **< trivial >; $T_s = O(N^2)$**

Parallel Algorithm

A) Block-checkerboard part. on a mesh

Assumption: **$N^2 = kP$, each PE holds a sub-matrix of size N^2/P**

—**Step 1** **Nearest neighbour exchange of non-diagonal blocks**



—Step 2 Transposition of local blocks in parallel

Analysis:

$T_p = O(N^2/P)$ + <local transpose>
$+ O(N^2/\sqrt{(P)})$ <block transfer>
$C = O(N^2\sqrt{(P)})$, not optimal

B) Block-checkerboard part. on a cube

Assumption: $P=4^n$, $N=2^m$, $m>2n$

—Recursive step Divide all logical subcubes into 4 sub-subcubes and swap upper-right and lower-left blocks in parallel

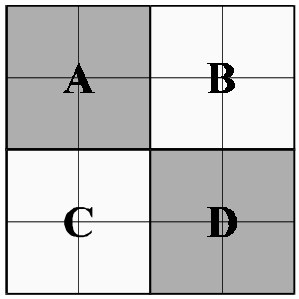
—Exit condition logical subcube is a single PE

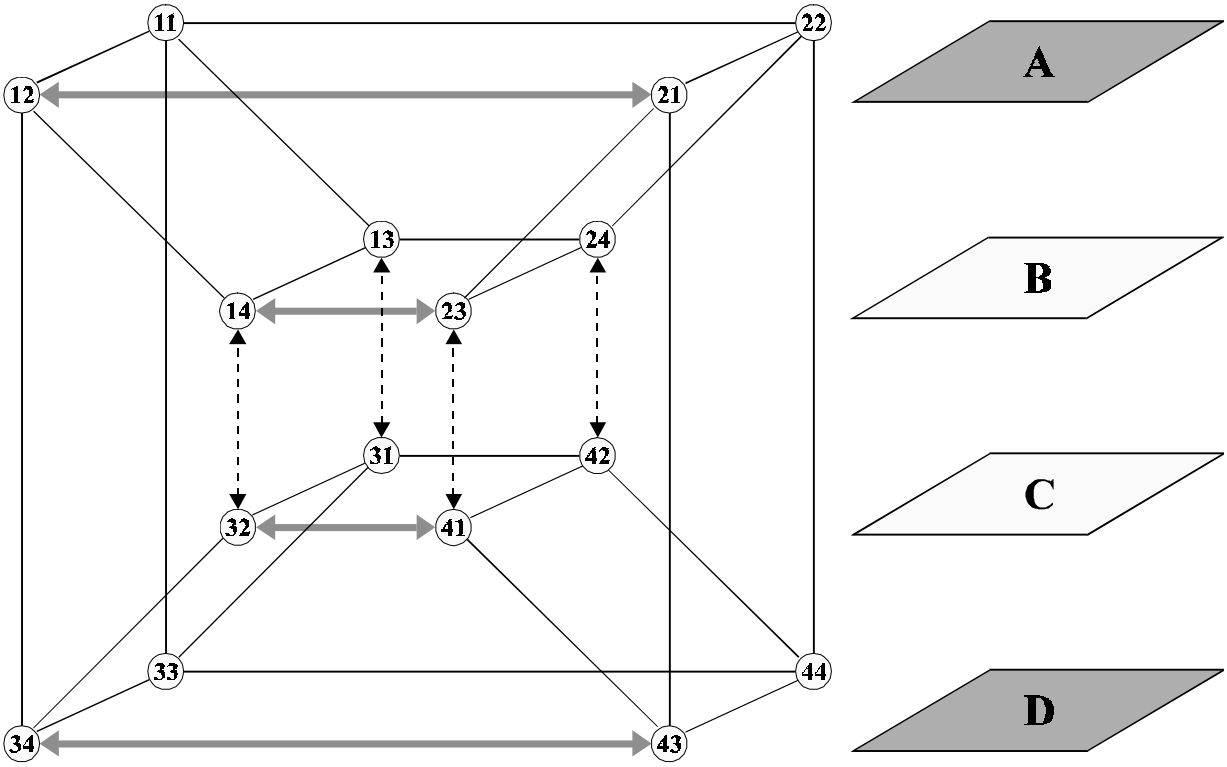
—Final step Transposition of local blocks in parallel

$\mathbf{X} =$

11	12	13	14
21	22	23	24
31	32	33	34
41	42	43	44

$=$





Swap in Step 1

Swap in Step 2

Analysis:

$T_p = O(N^2/P) +$ <local transpose>

$+ O(N^2 \log(P) / P)$ <block transfer>

$C = O(N^2 \log(P))$, not optimal

2 Algorithms for General Dense Matrices
2.2 MATRIX-VECTOR MULTIPLICATION

Problem $Ax=y,\ y_i=\Sigma\, a_{i,j}x_j$

Serial Algorithm < trivial >; $T_s=O(N^2)$

Parallel Algorithm

A) Striping (row-wise)

Assumption: P=N, each PE holds a row

—**Step 1** x is loaded into all PEs

—**Step 2** each PE_i computes y_i in parallel

Analysis:

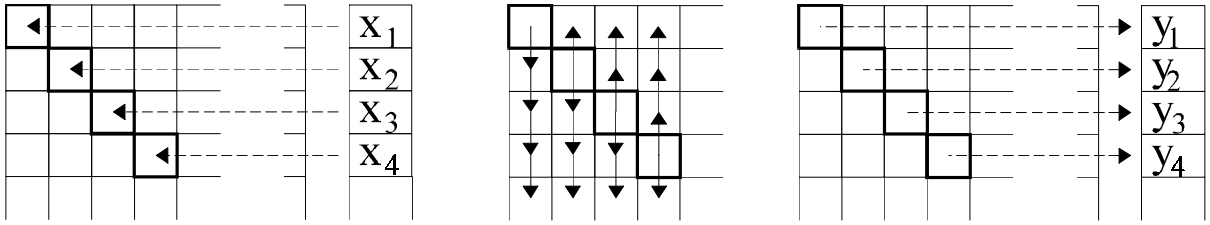
$T_p=O(N)$ + <one-to-all broadcast of x>
+ $O(N)$ <local product>
$C=O(N^2)$, optimal.

Note: if P<N (block stripes) on a cube or tree, then

$T_p=O(N+\log(N))$ + <one-to-all broadcast of x>
+ $O(N^2/P)$ <local product>
$C=O(N^2)$, optimal.

B) Checkerboard

Assumption: $p=N^2$



Step 1 **Step2** **Step3**

Note:

Block-checkerboard is faster than block-striping for a fixed *P*.

2 Algorithms for General Dense Matrices
2.3 MATRIX MULTIPLICATION

Problem AB=C, $c_{i,j}=\Sigma\, a_{i,k}b_{k,j}$
Serial Algorithm < trivial >; $T_s=O(N^3)$
Note: best serial algorithm $T_s=O(N^{2.37})$

Parallel Algorithm

A) Block-checkerboard

Assumption: $P<N$, each PE holds an $N/\sqrt{(P)} \times N/\sqrt{(P)}$ submatrix

—**Step 1** all-to-all broadcast of submatrices

—<u>Step 2</u> each $PE_{i,j}$ computes the (i,j) block of C

Analysis:

$T_P = O(N^2/\sqrt{(P)} + \log(P))$ + <block transfer on a cube>

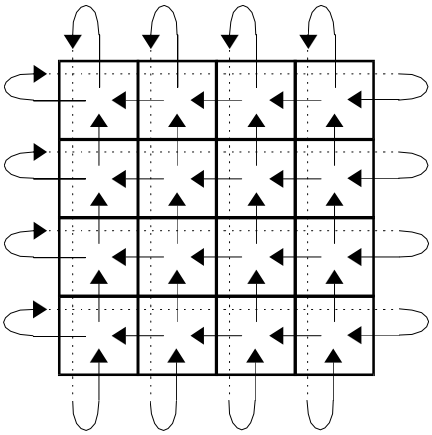
$+ O(N^3/P)$ <local product>

$C = O(N^3)$, optimal

B) Cannon´s algorithm on a mesh

<u>Advantage</u>: much less memory needed in PEs

<u>Idea</u>: Same algorithm, but wrapped-around shifts instead of global communication



Analysis: $N/\sqrt{(P)}$ shift+add steps needed

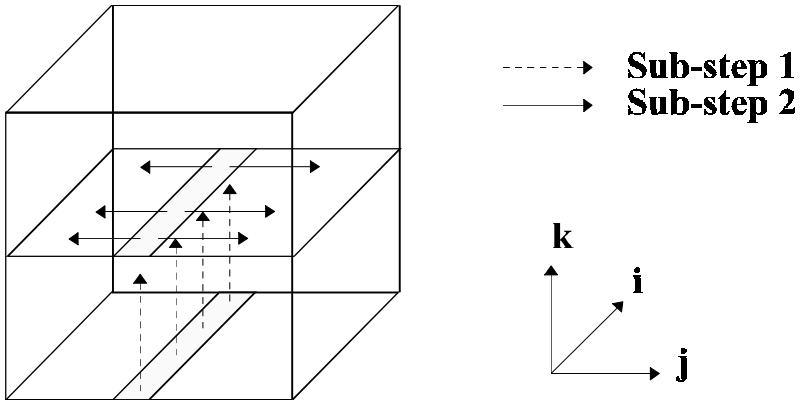
T_p , C : same as above (optimal)

C) DNS algorithm on a 3-D mesh

Note: **Fastest if enough processors available**

Assumption: $P=N^3$,

—Step 1 **Distribution of matrix A**



—Step 2 **Distribution of matrix B**

—Step 3 **Computation of product**

—Step 4 **Parallel addition**

Analysis:

$T_p=O(\log(N))$ <on a hypercube>

$C=O(N^3\log(N))$, not optimal.

2 Algorithms for General Dense Matrices
2.4 MATRIX INVERSION

Problem $AA^{-1}=I$, $A^{-1}=?$

Idea:

$$A^{-1}=\begin{bmatrix} I & -A_{11}^{-1}A_{12} \\ 0 & I \end{bmatrix}\begin{bmatrix} -A_{11}^{-1} & 0 \\ 0 & X^{-1} \end{bmatrix}\begin{bmatrix} I & 0 \\ -A_{21}A_{11}^{-1} & I \end{bmatrix},$$

where

$$X = A_{22} - A_{21}A_{11}^{-1}A_{12}$$

Parallel Algorithm

Using matr. mult., compute recursively A_{11}^{-1} **and** X^{-1}**.**

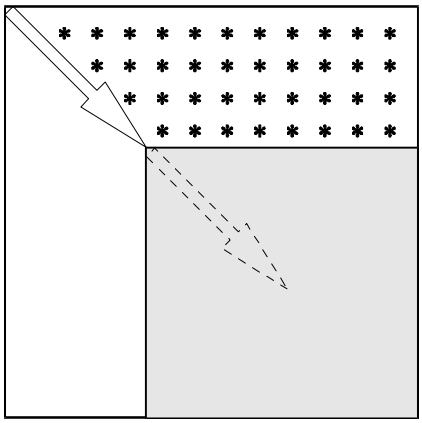
Analysis:

$T_p(N)=2T_p(N/2)+O(\text{matr.mult})=\log^2(N)$, **at least.**

2 Algorithms for General Dense Matrices
2.5 SOLVING A LINEAR SYSTEM AND THE LU DECOMPOSITION

Problem $Ax=b$, $x=?$

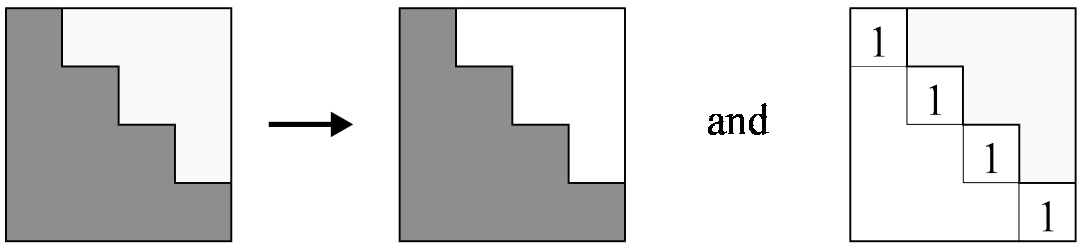
Serial Algorithm: Gaussian elimination



Substract appropriate multiples of equations to eliminate unknowns —> Upper-triangular coeff. matrix.

Note: This also solves the LU-decomp:

I. $A \xrightarrow{\text{Modified Elimination}} A'$



II. $A' \longrightarrow L$ and U

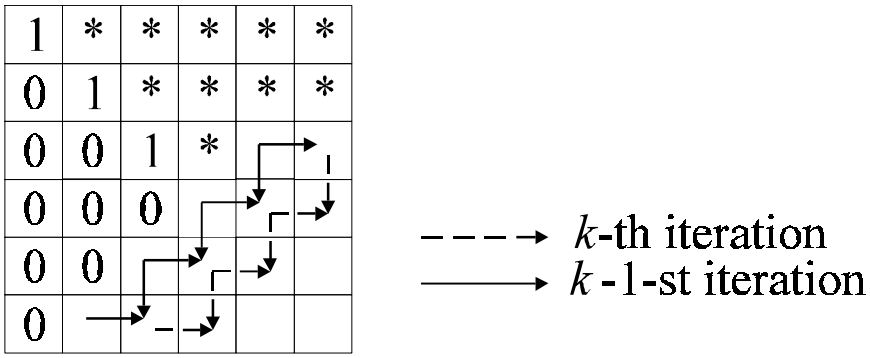
III. Then $A=LU$

Parallel Algorithm

Block-checkerboard partitioning

Assumption: $P<N$, each PE holds an $N/\sqrt{(P)} \times N/\sqrt{(P)}$ submatrix

Serial alg. with both row-wise and column-wise pipelining:



Analysis:
C=$O(N^3)$, can be improved by cyclic partitioning

Note #1:

Better numerical stability, if a *pivot element* is chosen; implementation with row-wise cyclic striping.

Note #2:

Back-substitution (Ux=y, x=?) is a simple recursive formula

$$x_k = y_k - u_{k,k+1}x_{k+1} - \ldots - u_{k,N}x_N$$

Implementation with row-wise block striping of U.

2 Algorithms for General Dense Matrices
2.6 INVERSION-BASED SOLUTION OF LINEAR SYSTEMS

Idea: **(LU-decomposition)**

We want that

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix}$$

#1: compute A_{11}^{-1} **...an N/2×N/2 matrix**

#2: LU-decompose A_{11} **...an N/2×N/2 matrix**

#3: LU-decompose $Y = A_{22} - A_{21} A_{11}^{-1} A_{12}$ **...an N/2×N/2 matrix**

Analysis:

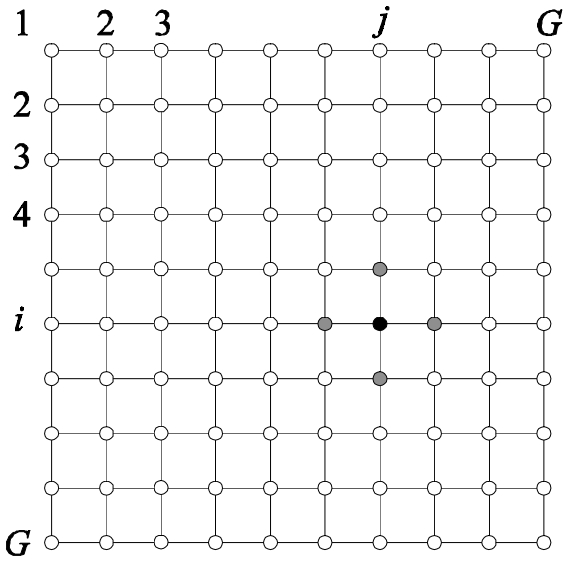
$T_p(N)=T_p(N/2)+O$(matr.mult+inv.)=$\log^3(N)$, at least.

Note :

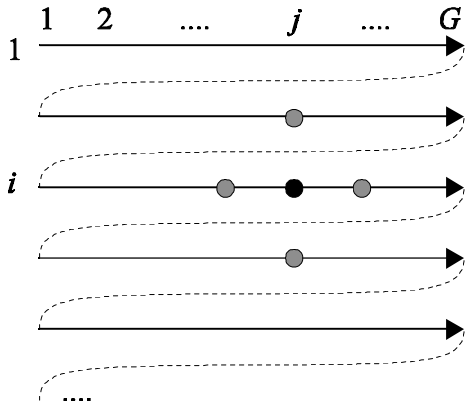
Similar recursive scheme for the back-substitution as well.

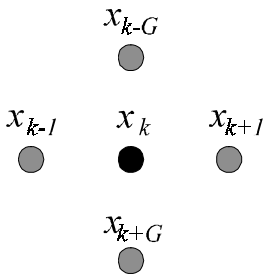
3 Algorithms for Special Matrices
3.1 THE FINITE ELEMENT METHOD

- **Simulation, Modelling**
- **Approximate solution of diff. equations**
- **Discretization of the problem space**
- **in 1-, 2-, 3-D**

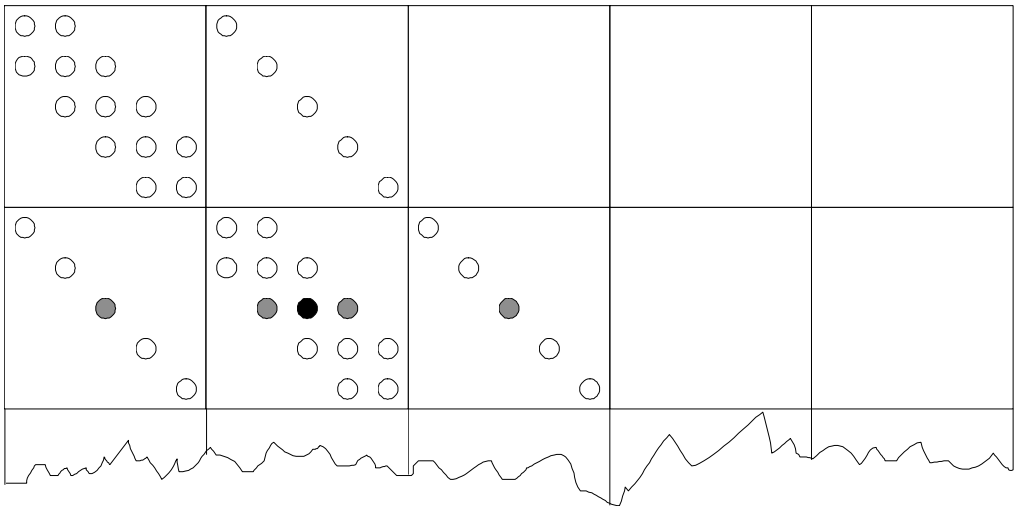


Lexicographical ordering of equations:





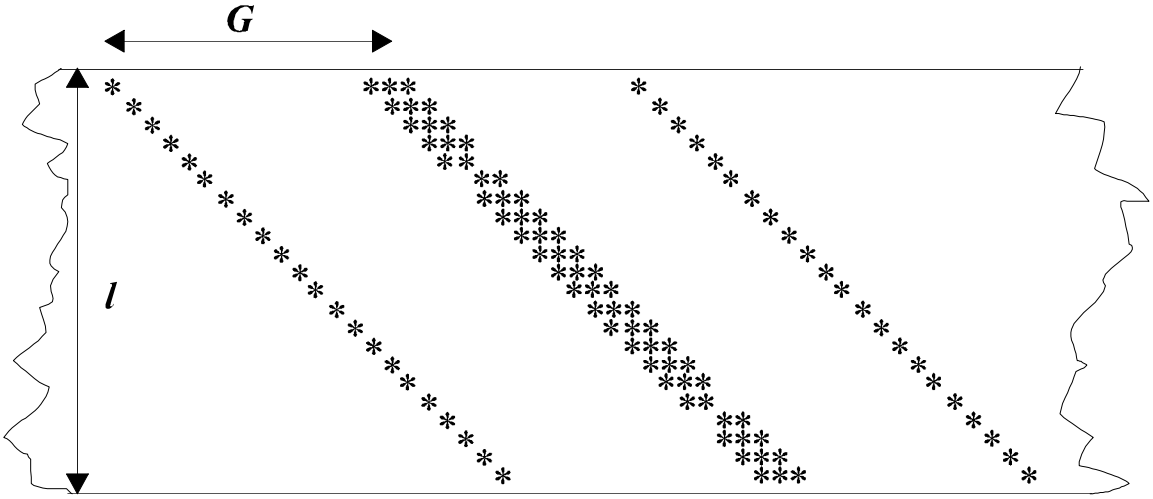
Problem formulation with a tridiagonal matrix:



3 Algorithms for Special Matrices

3.2 MULTIPLICATION OF A BLOCK-TRID. MATRIX BY A VECTOR

Rowwise block-striped partitioning:



Parallel algorithm: (for $l > G$)

— Step 1: Not wrapped-around shift UP in *G* steps

— Step 2: Not wrapped-around shift DOWN in *G* steps

3 Algorithms for Special Matrices
3.3 JACOBI-ITERATION FOR LINEAR SYSTEMS

Iterative methods:

- **improve an initial approximate solution**
- **evaluate convergence criteria regularly**
- **faster than direct methods, but not so general-purpose**

<u>Serial algorithm:</u> Ax=b, x=?

- **random initial solution x[0]**
- **Compute** $x_i[k] = \frac{r_i[k-1]}{a_{i,i}} + x_i[k-1]; \ldots . a_{i,i} \neq 0$
- **Compute residual vector** $r[k] = b - Ax[k]$
- **Convergence check:** $\|r[k]\| \leq \varepsilon$

<u>Parallel algorithm:</u> (rowwise block striping)

Step 1: No inter-PE comm.

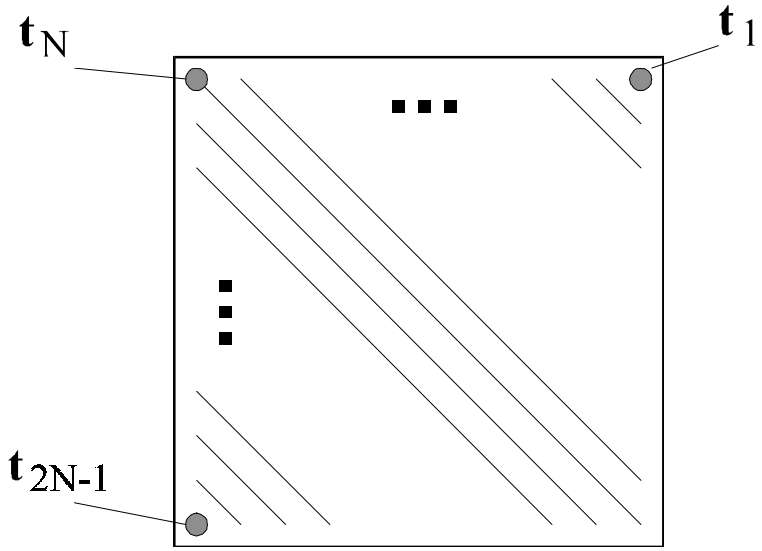
Step 2: Matrix-vector mult. + parallel addition of subsums

(log(N)) <hypercube>

3 Algorithms for Special Matrices
3.4 TOEPLITZ MATRIX-VECTOR PRODUCT

Toeplitz matrices:

Grid method for 1-D constant coeff. diff. equations with less localized dependences.



The product is part of the convolution

$$Ta = d = a \otimes t,$$

where

$$d_k = \sum_{j=1}^{k} a_j t_{k-j}$$

Convolution: O(`log` (a+b)) on a hypercube

Further reading

- **Rózsa Pál, *Lineáris algebra és alkalmazásai,* Tankönyvkiadó, Budapest 1991.**
- **J.Jájá, *An Introduction to Parallel Algorithms*, Addison-Wesley 1992.**
- **S.Y.Kung, *VLSI Array Processors*, Prentice Hall 1992.**
- **V.Kumar et al, *Introduction to parallel computing*, Benjamin/Cummings 1994.**
- **S.G.Akl, *The Design and Analysis of Parallel Algorithms*, Prentice Hall 1989.**