Writing Parallel Scientific Applications on PARAM

Dheeraj Bhardwaj
Department of Computer Science & Engineering
Indian Institute of Technology, Delhi –110 016 India
http://www.cse.iitd.ac.in/~dheerajb

Mathematical Models

- **Many Mathematical models which attempt to interpret real problems can be formulated in terms of rate of change of one or more variables as such naturally lead to partial differential equations.**
- **Methods for solution (Analytical / Numerical) of PDEs tend to be very problem dependent, so PDEs are usually solved by custom written analytical methods or/and software to take maximum advantage of particular features of given problem.**

Classification of PDEs

Following terms are often used to describe PDEs even when meaning is not so precise.

- **Hyperbolic PDEs describe time-dependent physical processes such as wave motion, that are not evolving towards steady state.**
- **Parabolic PDEs describe time-dependent physical processes, such as diffusion of heat, that are evolving toward steady state.**
- **Elliptic PDEs describe processes that have already reached steady state, and hence are time independent.**

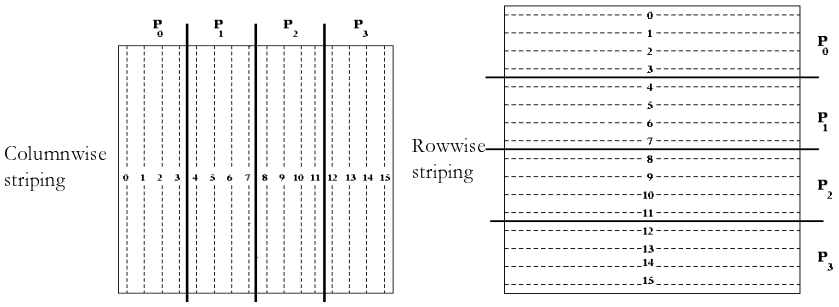
Parallel Matrix Computations

Mapping of Matrices onto Processors

Striped Partitioning

- Division into group of complete rows or columns
- Assign each processor one such group
- If each group contains an equal number of rows or columns - Uniform partitioning

Block-striped partitioning



Mapping of Matrices onto Processors

Block-striped partitioning

- Each processor is assigned contiguous rows or columns
- Processor P_i contains columns with indices

  $(n/p)i, (n/p)i+1, \ldots\ldots, (n/p)(i+1) - 1.$

n x n = size of the matrix; p = No. of processors

Cyclic-striped partitioning

- Distribution of rows or columns among the processors in wraparound manner
- Processor Pi will have rows with indices

  $i, i+p, i+2p, \ldots\ldots, i+n - p.$

Mapping of Matrices onto Processors

Block-Cyclic striped partitioning

- Hybrid between block and Cyclic distribution
- Strip matrix into blocks of q rows (q = n/p)
- Distribution of these blocks among processors in cyclic manner

We can partition an nxn matrix among a maximum of 'n' processors

Checkerboard Partitioning

- Division of matrix into smaller square or rectangular block or submatrices
- Distribution of such blocks or submatrices among processors
- All submatrices of same size - - Uniform partitioning

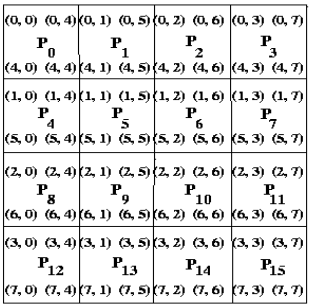
Block-checkerboard partitioning

(0, 0) (0, 1) **P_0** (1, 0) (1, 1)	(0, 2) (0, 3) **P_1** (1, 2) (1, 3)	(0, 4) (0, 5) **P_2** (1, 4) (1, 5)	(0, 6) (0, 7) **P_3** (1, 6) (1, 7)
(2, 0) (2, 1) **P_4** (3, 0) (3, 1)	(2, 2) (2, 3) **P_5** (3, 2) (3, 3)	(2, 4) (2, 5) **P_6** (3, 4) (3, 5)	(2, 6) (2, 7) **P_7** (3, 6) (3, 7)
(4, 0) (4, 1) **P_8** (5, 0) (5, 1)	(4, 2) (4, 3) **P_9** (5, 2) (5, 3)	(4, 4) (4, 5) **P_{10}** (5, 4) (5, 5)	(4, 6) (4, 7) **P_{11}** (5, 6) (5, 7)
(6, 0) (6, 1) **P_{12}** (7, 0) (7, 1)	(6, 2) (6, 3) **P_{13}** (7, 2) (7, 3)	(6, 4) (6, 5) **P_{14}** (7, 4) (7, 5)	(6, 6) (6, 7) **P_{15}** (7, 6) (7, 7)

Checkerboard Partitioning

Cyclic-checkerboard partitioning

Mapping the rows onto processors in a cyclic manner followed by the columns or vice-versa



Hybrid-checkerboard partitioning

- **Block-cyclic checkerboard partitioning**
- **Divide matrix into mxn blocks and map these blocks of size pxq in a cyclic manner**

Checkerboard Partitioning

- **Partitioned square matrix maps naturally onto a 2D square mesh of processors**
- **It is often convenient to visualize the ensemble of processors as a logical 2D mesh**
- **Unlike striping, the lowest level of granulate in checker- -boarding is one matrix per element**
- **Can exploit more concurrency then striping**

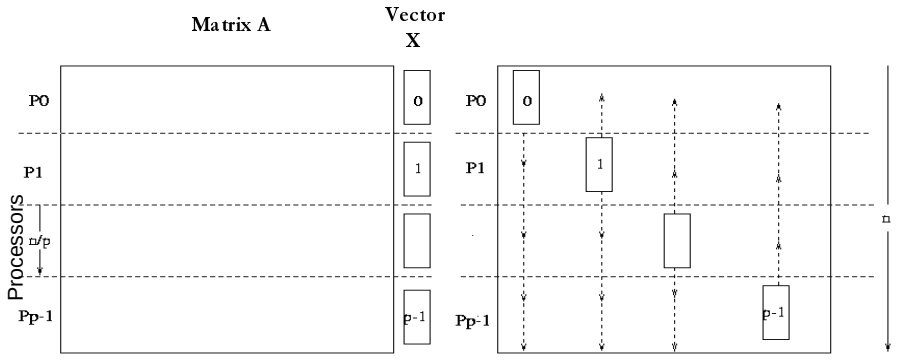
Mapping of Matrices onto Processors

- **Block partitioning leads to high granularity, which is recommended for high latency networks (EtherNet)**
- **Cyclic distribution provides an opportunity for load balancing on network of workstations. If load balancing is not performed then overall efficiency of the code will be decided by the slowest processor OR it might lead to idling of the processors.**
- **Most of the commercial Linear Algebra libraries make use of block-cyclic distribution (e.g. ScaLAPACK & PetSc)**

Matrix - Vector Multiplication

Parallel Formulation

Rowwise Striping



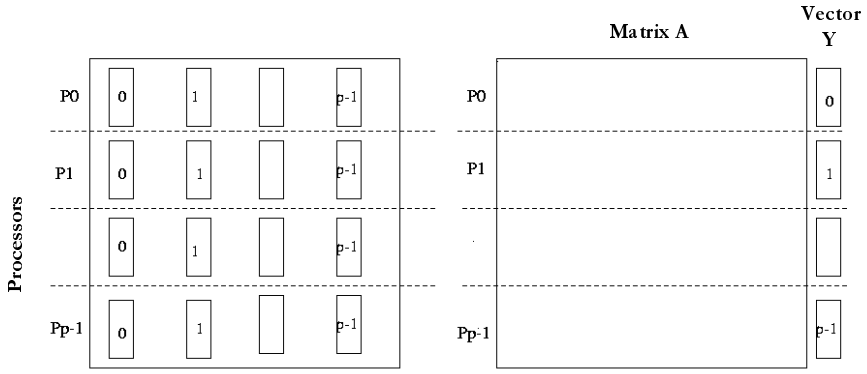
Initial partitioning

Distribution of the full vector among all the processors by all-to-all broadcast

Matrix - Vector Multiplication

Parallel Formulation

Rowwise Striping

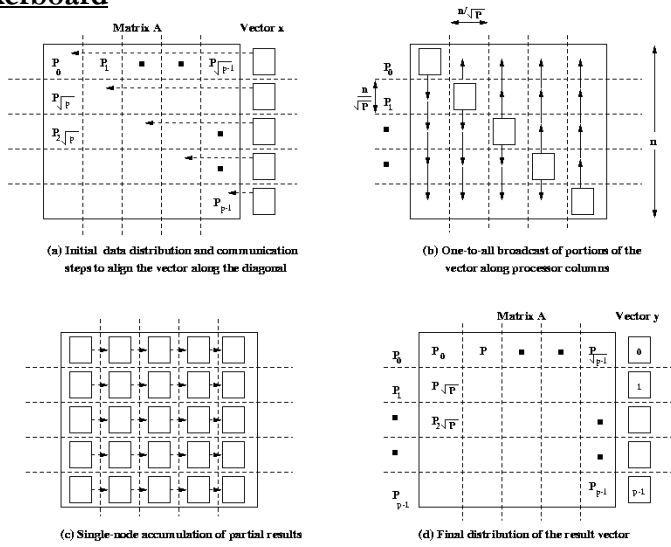


Entire vector distributed to each processor after the broadcast

Final distribution of the matrix and the result vector Y

Matrix - Vector Multiplication

Checkerboard



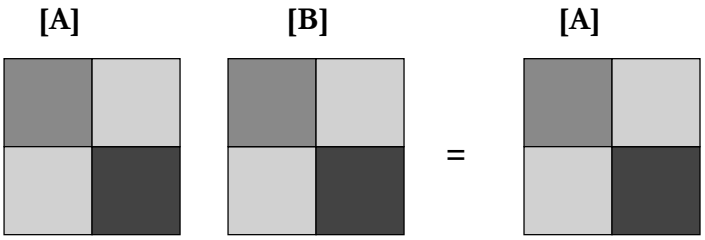
(a) Initial data distribution and communication steps to align the vector along the diagonal

(b) One-to-all broadcast of portions of the vector along processor columns

(c) Single-node accumulation of partial results

(d) Final distribution of the result vector

Matrix - Matrix multiplication



Checkerboard partitioning

Matrix - Matrix multiplication

- **A simple Parallel Algorithm**
- **Cannon's Algorithm**
- **Fox's Algorithm**
- **The DNS algorithm**

How to Decide the Partitioning ?

- **Efficient memory access pattern**
- **Most of the RISC based processors have hierarchical system of memory. Data access from high hierarchy is order of magnitude faster than lower hierarchy.**

  Checkerboard partitioning leads to efficient memory access pattern thus achieve better performance of serial part on parallel machines.
- **Minimum inter-processor communication**
- **On conflict between memory access and inter-processor communication, we give preference to memory access.**

  Reason : 1. Memory is no more a problem

  2. Even on shared memory computers inter-processor communication takes hundreds of cycles which is prohibitively expensive.

Mathematical Formulation

- **Acoustic Wave Equation in a Heterogeneous Medium**

$$\frac{1}{K}\frac{\partial^2 p}{\partial t^2} = \frac{\partial}{\partial x}\left(\frac{1}{\rho}\frac{\partial p}{\partial x}\right) + \frac{\partial}{\partial y}\left(\frac{1}{\rho}\frac{\partial p}{\partial y}\right) + \frac{\partial}{\partial z}\left(\frac{1}{\rho}\frac{\partial p}{\partial z}\right)$$

- **If u and w are x and z components of velocity vector, then**

$$\rho\frac{\partial u}{\partial t} = \frac{\partial p}{\partial x}, \quad \rho\frac{\partial u}{\partial t} = \frac{\partial p}{\partial x} \quad \text{and} \quad \rho\frac{\partial w}{\partial t} = \frac{\partial p}{\partial z}$$

- **Hyperbolic System of Equations**

$$\frac{\partial P}{\partial t} = A\frac{\partial P}{\partial x} + B\frac{\partial P}{\partial y} + C\frac{\partial P}{\partial z}$$

$$P = \begin{bmatrix} p \\ u \\ v \\ w \end{bmatrix} \quad A = \begin{bmatrix} 0 & \lambda & 0 & 0 \\ \rho^{-1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 & \lambda & 0 \\ 0 & 0 & 0 & 0 \\ \rho^{-1} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad C = \begin{bmatrix} 0 & 0 & 0 & \lambda \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \rho^{-1} & 0 & 0 & 0 \end{bmatrix}$$

Finite Difference Formulation

Explicit finite difference predictor :

$$(1+A\Delta t)\hat{P}_{i,j,k} = 2P^{n}_{i,j,k} - (1-A\Delta t)P^{n-1}_{i,j,k} + a\left(P^{n}_{i+1,j,k} + P^{n}_{i-1,j,k} + P^{n}_{i,j+1,k} + P^{n}_{i,j-1,k} + P^{n}_{i,j,k+1} + P^{n}_{i,j,k-1} - 6P^{n}_{i,j,k}\right)$$

Explicit finite difference corrector :

$$(1+A\Delta t)P^{n+1}_{i,j,k} = (1+A\Delta t)(1-\gamma)\hat{P}_{i,j,k} + 2\gamma\hat{P}_{i,j,k} - \gamma[(1-A\Delta t)P^{n-1}_{i,j,k} + a(\hat{P}_{i+1,j,k} + \hat{P}_{i-1,j,k} + \hat{P}_{i,j+1,k} + \hat{P}_{i,j-1,k} + \hat{P}_{i,j,k+1} + \hat{P}_{i,j,k-1} - 6\hat{P}_{i,j,k})]$$

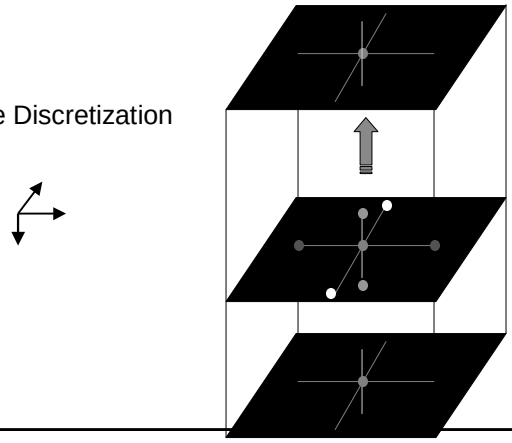
where, $a = \left(\frac{\Delta t}{\Delta x} v\right)^2$ We assume $\Delta x = \Delta y = \Delta z$

3D Acoustic Wave Modeling

3D Acoustic Wave Equation

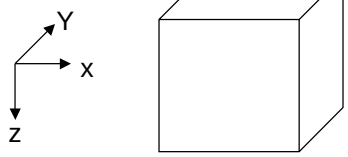
$$\frac{1}{V^2}\frac{\partial^2 P}{\partial t^2} = \frac{\partial^2 P}{\partial x^2} + \frac{\partial^2 P}{\partial y^2} + \frac{\partial^2 P}{\partial z^2} + f(t)\delta(x-x_s)(y-y_s)(z-z_s)$$

Finite Difference Discretization

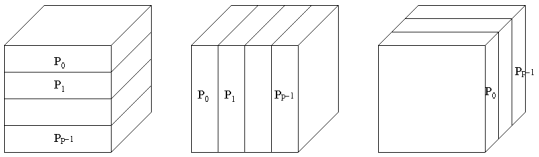


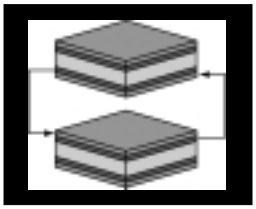
Domain Decomposition & Interprocessor Communication

- Balance the workload
- Minimize the perimeters of the Subdomain boundaries



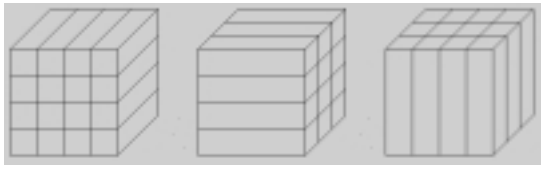
Stripe Partitioning

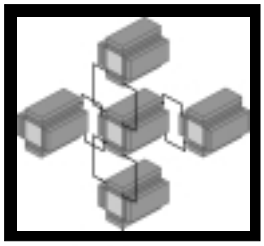




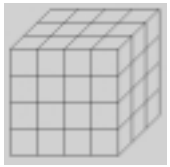
Domain Decomposition & Interprocessor Communication

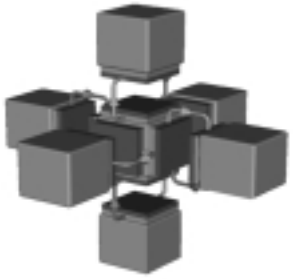
Hybrid Stripe Partitioning





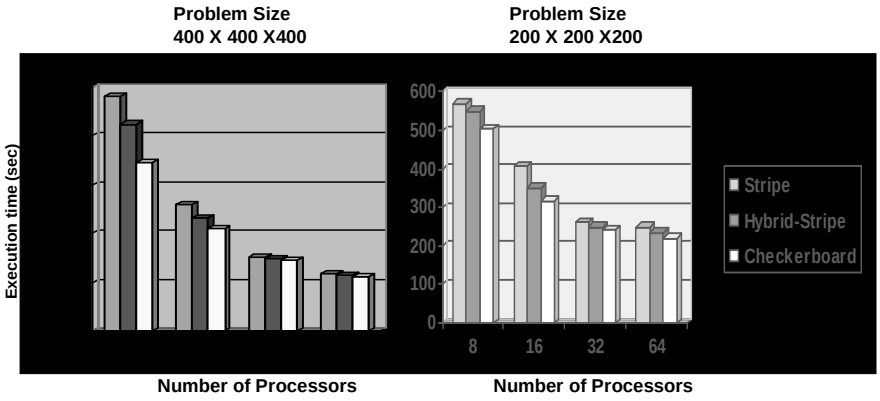
Checkerboard Partitioning





3D Wave Propagation

Performance Analysis



[A]{X} = [B]

Matrix Computations

Dense Matrix
Full Matrix with few zero entries

Sparse Matrix
Majority of elements zero

Solvers

1. **Direct Solvers**
2. **Iterative Solvers**

Why do we need direct solvers

- **Direct solvers give accuracy to the machine precision**
- **Iterative Solvers in general are very inefficient unless used with good preconditioners. Selection of preconditioner is one of the most critical phase of designing. They may lead to large overheads on matrix multiplication.**

Example : In general most of the iterative solvers act like smoothers, which reduces high frequency errors very fast but their convergence is very slow for low frequency error.

For unsymmetrical matrices arises from convective terms in fluid flows (They have directional biasness) ⟶ Direct methods are more reliable.

Why we don't use direct solvers ?

- **High memory requirements**

Even in the case of sparse matrices Gauss elimination or LU factorization may lead to storage requirements well beyond most of the present machines.

- **Hard to Parallelize**

The parallelization of best serial algorithm is often leads to poor efficiency on distributed parallel machines.

- **Good serial algorithm may not be good parallel algorithm**

We need a non- conventional thinking for developing efficient parallel algorithm for direct solvers.

LU Factorization

System of Linear Equations

$$Ax = b,$$

A is nxn matrix, b is given n-vector, and x is unknown solution n-vector to be determined.

To solve a linear system, we transform it into one whose solution is same but easier to compute.

One such form is LU factorization, $A = LU$, where L is unit lower triangular and U is upper triangular.

LU factorization of general nonsingular matrix A can be computed by Gaussian elimination.

LU Factorization

System of Linear Equations

If $A = LU$, then system $Ax = b$ becomes

$$Ax = LU\,x = b,$$

Which can be solved by forward-substitution in lower triangular system

$$L\,y = b,$$

followed by back-substitution in upper triangular system

$$U\,x = y.$$

In general, row interchanges (Pivoting) may be necessary for existence and numerical stability of LU factorization

In the case of irreducible, Symmetric Positive Definite (SPD) matrix, the system can be solved by Cholesky factorization.

Difficulties in efficient Parallel Implementations

Substantial parallelism inherent in Sparse direct methods, but limited success has been achieved in developing efficient general purpose parallel formulation because :

- **The amount of computation relative to the size of the system to be solved is very small.**
- **Even the modest communication might lead to poor efficiency due to poor computation to communication ratio. This becomes even severe for parallel computation on NOW / COW.**
- **Poor design criterion: Most of the serial formulation give stress on minimizing memory use and operation count. These criteria may not lead to scalable parallel formulation.**

Gaussian elimination

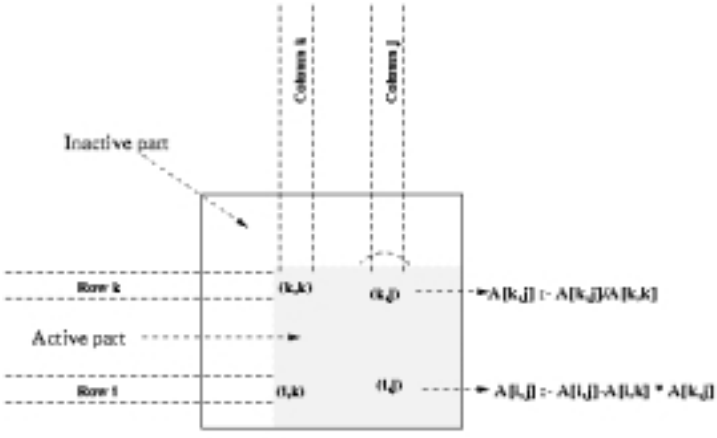


Fig. A typical computation in Gaussian elimination

Gaussian elimination

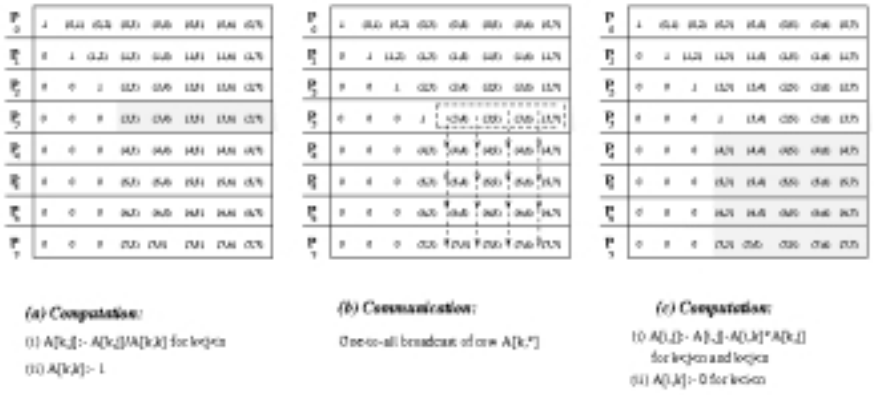


Fig. Computation load on different processors in block and cyclic-striped partitioning of an 8 X 8 matrix onto 16 processors during Gaussian elimination iteration corresponding to $k = 3$.

Gaussian elimination

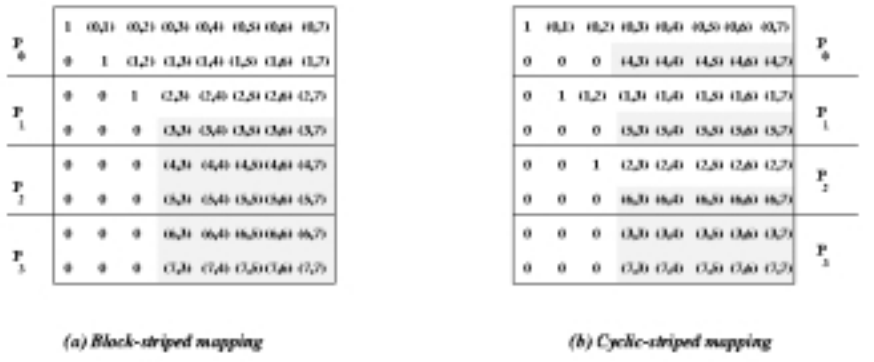


Fig. Computation load on different processors in block and cyclic-striped partitioning of an 8 X 8 matrix onto 16 processors during Gaussian elimination iteration corresponding to $k = 3$.

Gaussian elimination

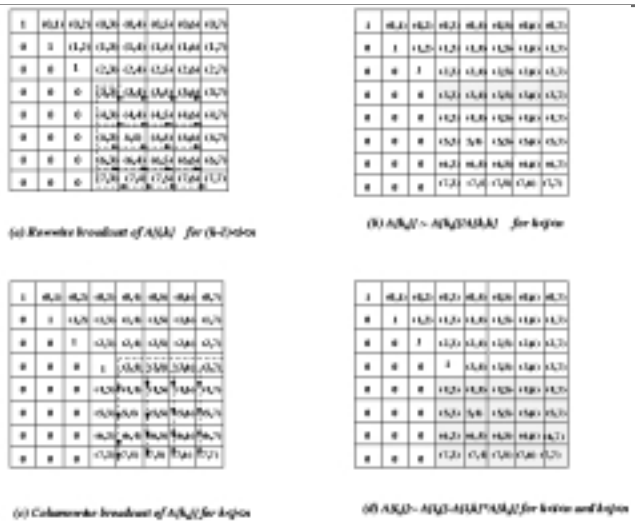


Fig. Various steps in the Gaussian elimination iteration corresponding to k = 3 for an 8 X 8 matrix on 64 processors of a two dimensional mesh

Cholesky Factorization

A : *nxn* Symmetric and Positive Definite (SPD) matrix.

Cholesky factorization : $A = L L^T$

Where L is lower triangular matrix with positive diagonal entries.

Given Cholesky factorization, linear system Ax = b, can be solved by successive forward and backward substitutions

$$Ly = b \quad \text{and} \quad L^T x = y$$

Cholesky Factorization

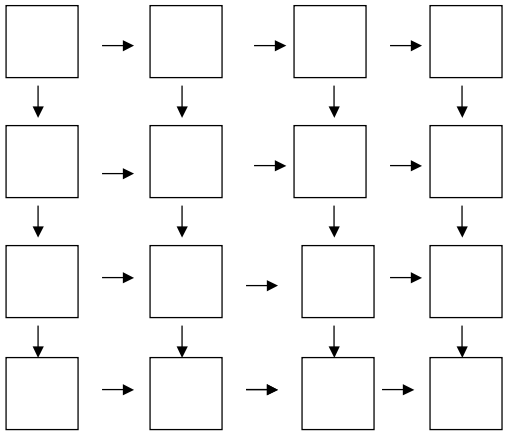
Algorithm Features

Features of Cholesky make it attractive for SPD matrices :

- **All n square roots involved are of positive numbers, so algorithms is well defined.**
- **No pivoting is required for numerical stability.**
- **Only lower triangular portion of *A* is accessed, and hence upper triangular portion need not to be stored.**
- **Factor *L* is computed in place, overwriting lower triangle of *A***
- **Only $n^3/6$ multiplications and similar number of additions are required.**

Cholesky Factorization

Parallel Algorithm



Cholesky Factorization

Parallel Algorithm

for k = 1 to min(i,j) - 1
  recv $a_{k,j}$
  recv $a_{i,k}$
  $a_{i,j} = a_{i,j} - a_{i,k} * a_{j,k}$
end
if i = j then
  $a_{i,i} = \sqrt{a_{i,i}}$
  broadcast $a_{i,i}$ to task (k,i) and (i,k), k = i+1, ……,n
else if I < j then
  recv $a_{i,j}$
  $a_{i,j} = a_{i,j} / a_{i,i}$
  broadcast $a_{i,j}$ to task (k,j) k = i+1, ……,j
else
  recv $a_{j,j}$
  $a_{i,j} = a_{i,j} / a_{j,j}$
  broadcast $a_{i,j}$ to task (i,k) k = j+1, ……,i
end

Cholesky Implementations

Three choices of index for outer loop yield different algorithms with memory access patterns

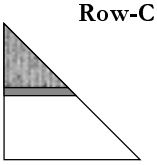
- **Row-Cholesky: With *i* in outer loop, inner loops solve triangular system for each new row in term of previous computed rows.**
- **Column-Cholesky: with *j* in outer loop, inner loops compute matrix-vector product that gives effect of previously computed columns on column currently being computed.**
- **Checkerboard-Cholesky: With *k* in outer loop, inner loops apply current columns as *rank-1* update to remaining unreduced submatrix.**

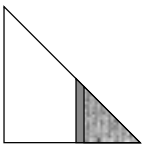
Cholesky Implementations

Memory Access Patterns

Row-Cholesky:

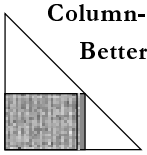
Good for well ordered equation: pivot selection is easy





Column-Cholesky

Better but less intitutive



Checkerboard-Cholesky

Modified

Used for Modification

Cholesky Implementations

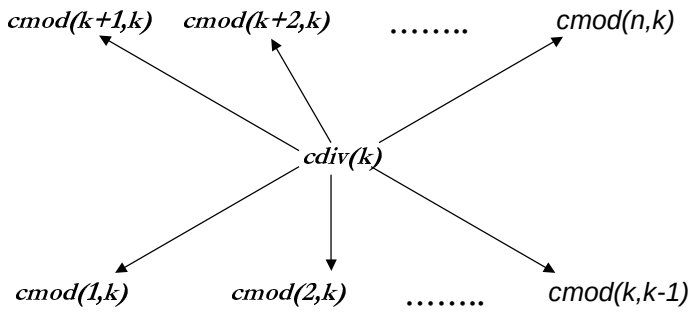
Column Operations

cmod(j,k) : column j is modified by a multiple of prior column k.

cdiv(j) : column j is scaled by square root of its diagonal elements

Cholesky Implementations

Data Dependencies



Cholesky Implementations

Data Dependencies

- ***cmod(k,*)*** **operations along bottom can be done in any order, but they all have same target column, so updating must be coordinated to preserve data integrity.**
- ***cmod(*,k)*** **operations along top can be done in any order, and they all have different target columns, so updating can be done simultaneously.**
- **Performing *cmods* concurrently is most important source of parallelism in column-oriented factorization.**
- **For dense matrix, each *cdiv(k)* depends on immediately preceding column, so only one *cdiv* can be done at a time.**

References

- Albert Y.H. Zomaya, Parallel and distributed Computing Handbook, McGraw-Hill Series on Computing Engineering, New York (1996).
- Ernst L. Leiss, Parallel and Vector Computing A practical Introduction, McGraw-Hill Series on Computer Engineering, New York (1995).
- Ian T. Foster, Designing and Building Parallel Programs, Concepts and tools for Parallel Software Engineering, Addison-Wesley Publishing Company (1995).
- Kai Hwang, Zhiwei Xu, Scalable Parallel Computing (Technology Architecture Programming) McGraw Hill New York (1997)
- Vipin Kumar, Ananth Grama, Anshul Gupta, George Karypis, Introduction to Parallel Computing, Design and Analysis of Algorithms, Redwood City, CA, Benjmann/Cummings (1994).