Parallel algorithms for Scientific Computations
On parallel factorization methods, communication-avoiding methods, pipelined iterative methods

Maya Neytcheva

PASC, November 2020

Plan of the lecture:

- Techniques to gain parallelism in matrix factorization methods
  - odd-even elimination (cyclic reduction)
  - odd-even simultaneous elimination
  - divide-and-conquer method
  - multi-frontal methods
- Communication avoiding techniques
- Pipelined versions of Krylov subspace iteration methods
- Extended Krylov subspace methods

How to gain parallelism in matrix factorization methods, in particular, for sparse matrices

LU factorization, again

Recall (in general):

```
for k := 1 : n
    c = a_kk^{-1}
    for j = k + 1 : n
        a_kj = a_kj * c
    for i = 1 : n and i ≠ k
        a_ij = a_ij − a_ik * a_kj
```

Note that the basic update step is the linked triad $a_{.j} - a_{kj}a_{.k}$.
To solve $A\mathbf{x} = \mathbf{b}$ use the (block-)matrix factorization $A = LU$:
$L\mathbf{y} = \mathbf{b}$ (forward substitution)
and
$U\mathbf{x} = \mathbf{y}$ (back substitution).

The problem is the recursion (purely sequential process).

Solutions with triangular matrices, again

Solution of $A\mathbf{x} = \mathbf{b}$,
where A has been already factorized as $A = LU$ or $A = LDU$. The matrices $L = \{l_{i,j}\}$ and $U = \{u_{i,j}\}$ are of the form

$$L = \begin{bmatrix} 1 & 0 & \dots & 0 \\ a_{21}^{(1)}/a_{21}^{(1)} & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}^{(1)}/a_{11}^{(1)} & a_{n2}^{(2)}/a_{22}^{(2)} & \dots & 1 \end{bmatrix} \quad \text{and} \quad U = \begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \dots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & \dots & a_{2n}^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_{nn}^{(n)} \end{bmatrix}$$

To compute $\mathbf{x}$, we must perform two steps:
forward substitution: $L\mathbf{z} = \mathbf{b}$, i.e.,

$$z_1 = b_1$$
$$z_i = b_i - \sum_{k=1}^{i-1} l_{i,k} z_k, \; i = 2, 3, \cdots, n. \tag{1}$$

backward substitution: $U\mathbf{x} = \mathbf{z}$, i.e.,

$$x_n = z_n$$
$$x_i = z_i - \sum_{k=i+1}^{n} u_{i,k} x_k, \; i = n-1, n-2, \cdots, 1. \tag{2}$$

For illustration purposes we consider (block-) tridiagonal matrices and their solution in parallel.

Odd-even elimination (cyclic reduction) method

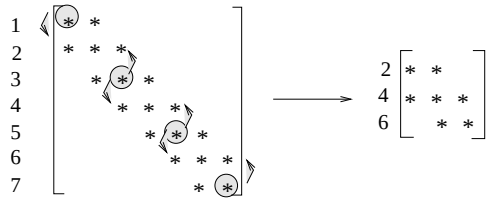


Figure: Odd-even elimination (reduction) method

Cyclic reduction in parallel

- Eliminate in row i (independently, n fine grain tasks)
- Communicate to 'adjacent' equations $\log n$ stages, increasing comm distance
- In this case it is better to assign adjacent rows to each process, instead of cyclic mapping.
- Local tridiagonal systems (of size n/p can be handled by serial cyclic reduction, LU or IC.

Odd-even elimination (reduction) method

$$\begin{bmatrix} a_{11} & a_{12} & & & & \cdots \\ a_{21} & a_{22} & a_{23} & & \cdots & \\ & a_{32} & a_{33} & a_{43} & \cdots & \\ & & a_{43} & a_{44} & a_{45} & \cdots \\ & & \vdots & \ddots & \ddots & \ddots \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ \vdots \end{bmatrix}$$

Eliminate x_1 from equation 1.
Eliminate x_3 from equations 2 and 4:

$$x_3 = \frac{1}{a_{23}}(b_2 - a_{22}^{(1)})x_2$$
$$(a_{32} - \frac{a_{33}a_{22}}{a_{23}})x_2 + a_{34}x_4 = b_2^{(1)}$$

Odd-even elimination (reduction) method

Advantages: No fill-in, $\log_2 n$ stages.
Drawbacks: the system size is reduced, thus, insufficient load and idleness of processors.

Can be done block-wise.

Odd-even simultaneous elimination method

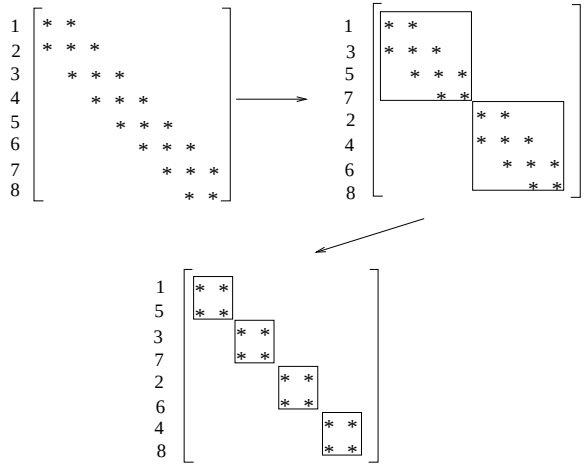
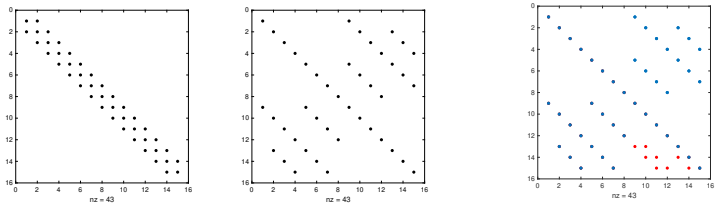


Figure: Odd-even elimination (reduction) method.

Odd-even reordering (nested dissection for triangular systems



(a) A (b) A(w,w) (c) L

$$w_0 = [1, 3, 5, 7, 9, 11, 13, 15, 2, 4, 6, 8, 10, 12, 14]$$
$$w = [1, 5, 9, 13, 3, 7, 11, 15, 2, 6, 10, 14, 4, 8, 12]$$

Divide and conquer approach when solving

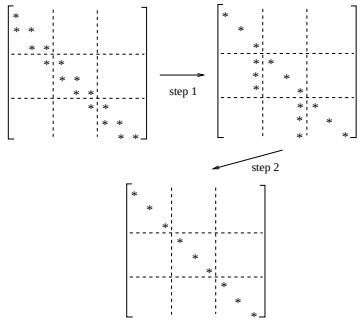


Figure: Divide-and-conquer algorithm

As a first step, all off-diagonal entries in all diagonal blocks are eliminated in parallel. The side effect is that a fill-in in the off-diagonal blocks occur. However, the fill-in entries in the off-diagonal blocks reside in only one column.

Multifrontal elimination

(Domain decomposition, matrix-graph based approach)
The multifrontal methods utilizes the structure to find parts to be treated concurrently.

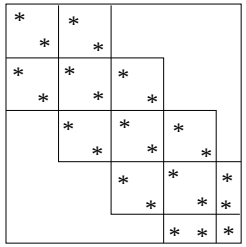


Figure: Matrix for two-way factorization

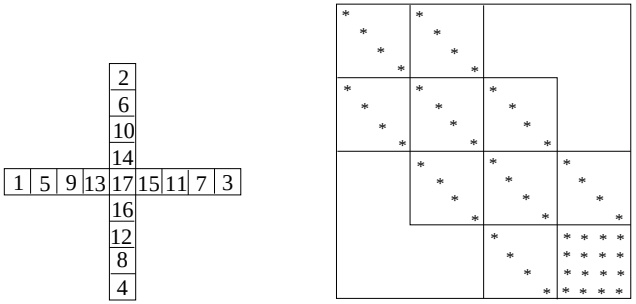


Figure: Plus-shaped domain and matrix for four-way factorization

Solve on each diagonal block in parallel using four processors.
Final block corresponding to the reduced system for the middle part of the structure we solve the corresponding linear system of equations
after Again perform the back substitution in parallel for the four separate parts of the structure.

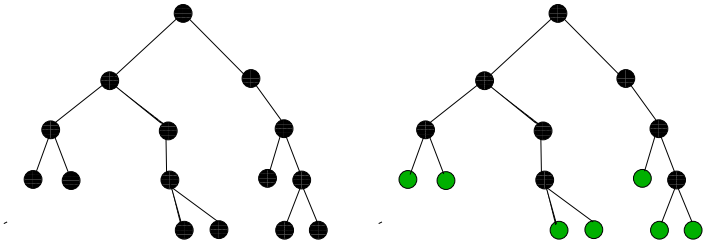
Multifrontal elimination, using nested dissection

Ideal structure: the 'arrow' matrix:

$$\begin{bmatrix} A_{11} & & & A_{14} \\ & A_{22} & & A_{24} \\ & & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix}$$

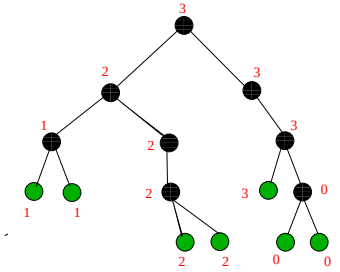
In practice: much more complicated structures for general sparse matrices.

Multifrontal



(a) Elimination tree (b) Elimination fronts

Multifrontal: Distributed memory setting



Constraints:

- Enough work per node
- Dependencies defined by the tree
- PEs must get roughly the same amount of work
- Minimize communication and idling of processors

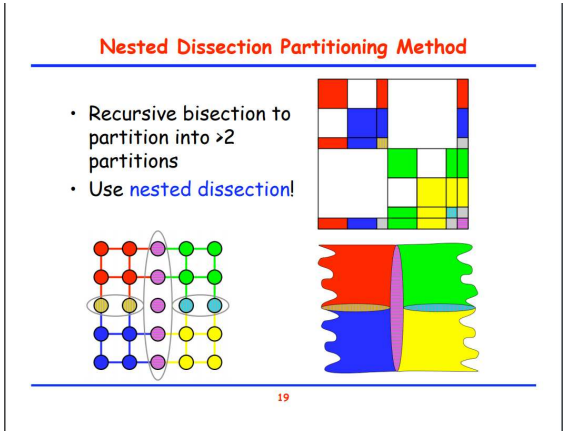
Multifrontal: What do we have per node?

Frontal matrix:

$$A1 = \begin{bmatrix} F1 & B1 \\ C1 & D1 \end{bmatrix}$$

$$A2 = D2 - C1 * inv(F1) * B1$$

Multifrontal:



Erik Boman, Sandia National Laboratories
Michael Wolf, University of Illinois at Urbana-Champaign

https://www.sandia.gov/~mmwolf/presentations/Conferences/mmwolfPP08.pdf

Communication avoiding techniques

Communication-avoiding, dense matrices

Idea: *Tall skinny QR, TSQR*

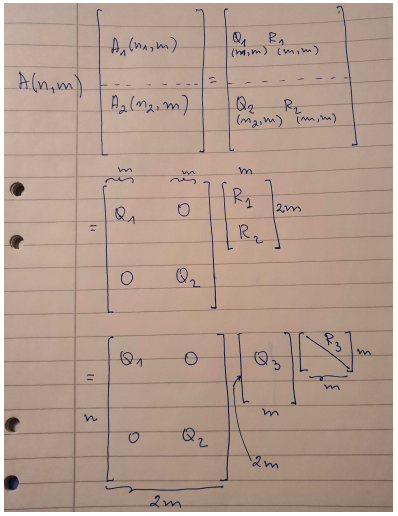
$$A = \begin{bmatrix} A0 \\ A1 \\ A2 \\ A3 \end{bmatrix} = \begin{bmatrix} Q0*R00 \\ Q1*R01 \\ Q2*R02 \\ Q3*R03 \end{bmatrix} = \begin{bmatrix} Q0 & & & \\ & Q1 & & \\ & & Q2 & \\ & & & Q3 \end{bmatrix} \begin{bmatrix} R00 \\ R01 \\ R02 \\ R03 \end{bmatrix}$$

$$\begin{bmatrix} R00 \\ R01 \\ R02 \\ R03 \end{bmatrix} = \begin{bmatrix} Q10*R10 \\ \\ Q11*R11 \end{bmatrix} \ldots$$

J. Demmel, LG, M. Hoemmen, J. Langou, 08 References: Golub, Plemmons, Sameh 88, Pothen, Raghavan, 89,

Da Cunha, Becker, Patterson, 02

Communication-avoiding, dense matrices, TSQR



Q is available as a product.

Communication-avoiding

Communication-avoiding versions exist for many algorithms:

- QR, SVD
- LU, Cholesky
- Sym, nonsym eigenproblems

Attaining the lower bounds of number-of-words-moved and number-of-messages

Communication-avoiding ideas for iterative solvers

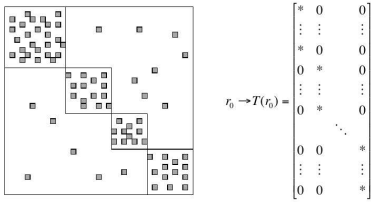
Two ideas:

- Decrease the number of iterations to decrease the number of global communications
- Increase arithmetic intensity - compute sparse matrix-set of vectors product

Enlarged Krylov subspace methods
2014-2016, 2019
Based on replacing the Krylov subspace with a larger subspace, specifically the enlarged Krylov subspace, leading to better convergence.

Enlarged Krylov subspaces

- Partition the matrix into t domains
- Split the initial residual into t vectors corresponding to the t domains



$$r_0 \rightarrow T(r_0) = \begin{bmatrix} * & 0 & 0 \\ \vdots & \vdots & \vdots \\ * & 0 & 0 \\ 0 & * & 0 \\ \vdots & \vdots & \vdots \\ 0 & * & 0 \\ & \ddots & \\ 0 & 0 & * \\ \vdots & \vdots & \vdots \\ 0 & 0 & * \end{bmatrix}$$

- Generate t new basis vectors, obtain an enlarged Krylov subspace

$$K_{t,k}(A,r_0) = span\left\{T(r_0), AT(r_0), \ldots, A^{k-1}T(r_0)\right\}$$

- Search for the solution of the system $Ax = b$ in $K_{t,k}(A,r_0)$

Enlarged Krylov methods, convergence

Laura Grigori, Sophie Moufawad, Frederic Nataf
Enlarged Krylov Subspace Conjugate Gradient Methods for Reducing Communication
SIAM Journal on Matrix Analysis and Applications 37(2):744-773, 2016
* Convergence at least as fast as the standard CG.

Pipelined versions of Krylov subspace iteration methods

Pipelined unpreconditioned CG

Given A, $\mathbf{b}$ and an initial guess $\mathbf{x}^{(0)}$.

$$
\begin{array}{rrcl}
 & \mathbf{r}^{(0)} & = & \mathbf{b} - A\mathbf{x}^{(0)}, \mathbf{w}^{(0)} = A\mathbf{x}^{(0)} \\
1 & \textit{for } k = 0, 1, \cdots & & \textit{until convergence} \\
2 & \gamma_k & = & (\mathbf{r}^{(k)}, \mathbf{r}^{(k)}) \\
3 & \delta & = & (\mathbf{w}^{(k)}, \mathbf{r}^{(k)}) \\
4 & \mathbf{q} & = & A\mathbf{w}^{(k)} \\
5 & \textit{if } k > 0, & & \\
6 & \beta_k & = & \gamma_k/\gamma_{k-1}; \alpha_k = \gamma_k/(\delta - \beta_k\gamma_k/\alpha_{k-1}) \\
7 & \textit{else} & & \\
8 & \beta_k & = & 0; \alpha_k = \gamma_k/\delta : \\
9 & \textit{endif} & & \\
10 & \mathbf{z}^{(k)} & = & \mathbf{q} + \beta_k\mathbf{z}^{(k-1)} \\
11 & \mathbf{s}^{(k)} & = & \mathbf{w}^{(k)} + \beta_k\mathbf{s}^{(k-1)} \\
12 & \mathbf{p}^{(k)} & = & \mathbf{r}^{(k)} + \beta_k\mathbf{p}^{(k-1)} \\
13 & \mathbf{x}^{(k+1)} & = & \mathbf{x}^{(k)} + \alpha_k\mathbf{p}^{(k)} \\
14 & \mathbf{r}^{(k+1)} & = & \mathbf{r}^{(k)} - \alpha_k\mathbf{s}^{(k)} \\
15 & \mathbf{w}^{(k+1)} & = & \mathbf{w}^{(k)} - \alpha_k\mathbf{z}^{(k)} \\
16 & \textit{endfor} & &
\end{array}
$$

Pipelined preconditioned CG

$$
\begin{array}{rrcl}
1 & \mathbf{r}^{(0)} & = & \mathbf{b} - A\mathbf{x}^{(0)}, \mathbf{u}^{(0)} = M^{-1}\mathbf{r}^{(0)}, \mathbf{w}^{(0)} = A\mathbf{u}^{(0)} \\
2 & \textit{for } k = 0, 1, \cdots & & \textit{until convergence} \\
3 & \gamma_k & = & (\mathbf{r}^{(k)}, \mathbf{u}^{(k)}) \\
4 & \delta & = & (\mathbf{w}^{(k)}, \mathbf{u}^{(k)}) \\
5 & \mathbf{m} & = & M^{-1}\mathbf{w}^{(k)} \\
6 & \mathbf{n} & = & A\mathbf{m} \\
7 & \textit{if } k > 0, & & \\
8 & \beta_k & = & \gamma_k/\gamma_{k-1}; \alpha_k = \gamma_k/(\delta - \beta_k\gamma_k/\alpha_{k-1}) \\
9 & \textit{else} & & \\
10 & \beta_k & = & 0; \alpha_k = \gamma_k/\delta : \\
11 & \textit{endif} & & \\
12 & \mathbf{z}^{(k)} & = & \mathbf{n} + \beta_k\mathbf{z}^{(k-1)} \\
13 & \mathbf{q}^{(k)} & = & \mathbf{m} + \beta_k\mathbf{q}^{(k-1)} \\
14 & \mathbf{s}^{(k)} & = & \mathbf{w}^{(k)} + \beta_k\mathbf{s}^{(k-1)} \\
15 & \mathbf{p}^{(k)} & = & \mathbf{u}^{(k)} + \beta_k\mathbf{p}^{(k-1)} \\
16 & \mathbf{x}^{(k+1)} & = & \mathbf{x}^{(k)} + \alpha_k\mathbf{p}^{(k)} \\
17 & \mathbf{r}^{(k+1)} & = & \mathbf{r}^{(k)} - \alpha_k\mathbf{s}^{(k)} \\
18 & \mathbf{u}^{(k+1)} & = & \mathbf{u}^{(k)} - \alpha_k\mathbf{q}^{(k)} \\
19 & \mathbf{w}^{(k+1)} & = & \mathbf{w}^{(k)} - \alpha_k\mathbf{z}^{(k)} \\
20 & \textit{endfor} & &
\end{array}
$$

Convergence rate, computational cost per iteration

Pipelined preconditioned Conjugate Residuals

$$
\begin{array}{rrcl}
1 & \mathbf{r}^{(0)} & = & \mathbf{b} - A\mathbf{x}^{(0)}, \mathbf{u}^{(0)} = M^{-1}\mathbf{r}^{(0)}, \mathbf{w}^{(0)} = A\mathbf{u}^{(0)} \\
2 & \textit{for } k = 0, 1, \cdots & & \textit{until convergence} \\
3 & \mathbf{m} & = & M^{-1}\mathbf{w}^{(k)} \\
4 & \gamma_k & = & (\mathbf{w}^{(k)}, \mathbf{u}^{(k)}) \\
5 & \delta & = & (\mathbf{m}^{(k)}, \mathbf{u}^{(k)}) \\
6 & \mathbf{n} & = & A\mathbf{m} \\
7 & \textit{if } k > 0, & & \\
8 & \beta_k & = & \gamma_k/\gamma_{k-1}; \alpha_k = \gamma_k/(\delta - \beta_k\gamma_k/\alpha_{k-1}) \\
9 & \textit{else} & & \\
10 & \beta_k & = & 0; \alpha_k = \gamma_k/\delta : \\
11 & \textit{endif} & & \\
12 & \mathbf{z}^{(k)} & = & \mathbf{n} + \beta_k\mathbf{z}^{(k-1)} \\
13 & \mathbf{q}^{(k)} & = & \mathbf{m} + \beta_k\mathbf{q}^{(k-1)} \\
14 & \mathbf{p}^{(k)} & = & \mathbf{u}^{(k)} + \beta_k\mathbf{p}^{(k-1)} \\
15 & \mathbf{x}^{(k+1)} & = & \mathbf{x}^{(k)} + \alpha_k\mathbf{p}^{(k)} \\
16 & \mathbf{u}^{(k+1)} & = & \mathbf{u}^{(k)} - \alpha_k\mathbf{q}^{(k)} \\
17 & \mathbf{w}^{(k+1)} & = & \mathbf{w}^{(k)} - \alpha_k\mathbf{z}^{(k)} \\
18 & \textit{endfor} & &
\end{array}
$$

Convergence rate, computational cost per iteration

s-step Krylov subspace iteration methods

Erin Carson
The adaptive *s*-step Conjugate Gradient Method
SIAM J. MATRIX ANAL. APPL., Vol. 39 2018, pp. 1318–1338
See the paper.

Erin C. Carson, Miroslav Rozloznik, Zdenek Strakos, Petr Tichy, Miroslav Tuma
The numerical stability analysis of pipelined conjugate gradient methods: historical context and methodology
SIAM J. Sci. Comput., 40(5), A3549–A3580, 2018
Conclusions: "... We give an expression for the residual gap which is applicable to any such variant, including the pipelined CG method of Ghysels and Vanroose [18]. We show that, similar to 3-term recurrence CG, any variant of CG which uses an auxiliary recurrence for Api in updating the residual vector can suffer from amplifications of local rounding errors due to large residual oscillations.
... modifications borne out of efforts to minimize time per iteration. While an important piece of the puzzle, one must be cautious about using time per iteration as a metric of efficiency. Alone, this fails to capture the whole picture of the method's effectiveness within the context of a scientific application. If the modifications

Bottom line: be precautious when trading numerical efficiency for parallel efficiency!
There should be a theoretical support for the algoritmic changes to ensure the quality of the numerically computed solutions.