Algorithms for Dense Matrices III

Stefan Lang

Interdisciplinary Center for Scientific Computing (IWR)
University of Heidelberg
INF 368, Room 532
D-69120 Heidelberg
phone: 06221/54-8264
email: Stefan.Lang@iwr.uni-heidelberg.de

WS 15/16

Topics

Data parallel algorithms for dense matrices

- LU decomposition

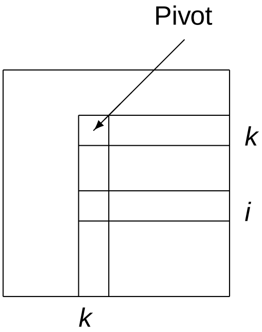
LU Decomposition: Problem Formulation

Be the linear equation system to solve

$$Ax = b \tag{1}$$

with a $N \times N$ matrix A and according vectors x and b.
Gaussian Elimination Method (sequential)

(1) **for** $(k = 0; k < N; k++)$
(2) **for** $(i = k+1; i < N; i++)$ {
(3) $l_{ik} = a_{ik}/a_{kk}$;
(4) **for** $(j = k+1; j < N; j++)$
(5) $a_{ij} = a_{ij} - l_{ik} \cdot a_{kj}$;
(6) $b_i = b_i - l_{ik} \cdot b_k$;
}



transforms the equation system (1) into the equation system

$$Ux = d \tag{2}$$

with an upper triangular matrix U.

LU Decomposition: Properties

Above formulation has the following properties:

- The matrix elements a_{ij} for $j \geq i$ contain the according entries of U, this means A will be overwritten.
- Vector b is overwritten with the elements of d.
- It is assumed, that the a_{kk} in line (3) is always non zero (no pivoting).

LU Decomposition: Derivation of Gaussian Elimination

The LU decomposition can be derived from Gaussian elimination:

- Each individual transformation step, that consists for fixed k and i from the lines (3) to (5), can be written as a multiplication of the equation system with a matrix $\hat{L}_{ik}$ from left:

$$\hat{L}_{ik} = \begin{array}{cc} & \begin{array}{cccccc} & & k & & & \end{array} \\ \mathrm{i} & \begin{pmatrix} 1 & & & & & \\ & 1 & & & & \\ & & \ddots & & & \\ & & & \ddots & & \\ & & -l_{ik} & & \ddots & \\ & & & & & 1 \end{pmatrix} \end{array} = I - l_{ik} E_{ik}$$

  E_{ik} is the matrix whose single element is $e_{ik} = 1$, and that otherwise consists of zeros, with l_{ik} from line (3) of the Gaussian elimination method.

LU Decomposition

- Thus applies

$$\hat{L}_{N-1,N-2} \cdot \dots \cdot \hat{L}_{N-1,0} \cdot \dots \cdot \hat{L}_{2,0}\hat{L}_{1,0}A = \\ = \hat{L}_{N-1,N-2} \cdot \dots \cdot \hat{L}_{N-1,0} \cdot \dots \cdot \hat{L}_{2,0}\hat{L}_{1,0}b \tag{3}$$

and because of (2) applies

$$\hat{L}_{N-1,N-2} \cdot \dots \cdot \hat{L}_{N-1,0} \cdot \dots \cdot \hat{L}_{2,0}\hat{L}_{1,0}A = U. \tag{4}$$

LU Decomposition: Properties

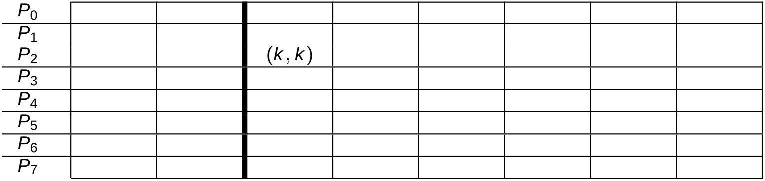
- There apply the following properties:
  1. $\hat{L}_{ik} \cdot \hat{L}_{i',k'} = I - l_{ik}E_{ik} - l_{i'k'}E_{i'k'}$ for $k \neq i'$ ($\Rightarrow E_{ik}E_{i'k'} = 0$) .
  2. $(I - l_{ik}E_{ik})(I + l_{ik}E_{ik}) = I$ für $k \neq i$, thus $\hat{L}_{ik}^{-1} = I + l_{ik}E_{ik}$.
- Because of 2 and the relationship (4)

$$A = \underbrace{\hat{L}_{1,0}^{-1} \cdot \hat{L}_{2,0}^{-1} \cdot \dots \cdot \hat{L}_{N-1,0}^{-1} \cdot \dots \cdot \hat{L}_{N-1,N-2}^{-1}}_{=:L} U = LU \tag{5}$$

- Because of 1, which also holds in its meaning for $\hat{L}_{ik}^{-1} \cdot \hat{L}_{i'k'}^{-1}$, L is a lower triangular matrix with $L_{ik} = l_{ik}$ for $i > k$ and $L_{ii} = 1$.
- The algorithm for LU decomposition of A is obtained by leaving out line (6) in the Gaussian algorithm above. The matrix L will be stored in the lower triangle of A.

LU Decomposition: Parallel Variant with Row-wise Partitioning

Row-wise partitioning of a $N \times N$ matrix for the **case** $N = P$:



- In step k processor P_k sends the matrix elements $a_{k,k}, \dots, a_{k,N-1}$ to all processors P_j with $j > k$, and these eliminate in their row.
- Parallel runtime:

$$
\begin{aligned}
T_P(N) &= \underbrace{\sum_{m=N-1}^{1}}_{\text{Number of rows to eliminate}} \Big(t_s + t_h + \underbrace{t_w \cdot m}_{\text{Rest of row } k}\Big) \underbrace{\operatorname{ld} N}_{\text{Broadcast}} + \underbrace{m 2 t_f}_{\text{Elimination}} \qquad (6) \\
&= \frac{(N-1)N}{2} 2 t_f + \frac{(N-1)N}{2} \operatorname{ld} N t_w + N \operatorname{ld} N (t_s + t_h) \\
&\approx N^2 t_f + N^2 \operatorname{ld} N \frac{t_w}{2} + N \operatorname{ld} N (t_s + t_h)
\end{aligned}
$$

LU Decomposition: Analysis of Parallel Variant

- Sequential runtime of LU decomposition:

$$
\begin{aligned}
T_S(N) &= \sum_{m=N-1}^{1} \underbrace{m}_{\text{rows are to elim.}} \quad \underbrace{2mt_f}_{\text{Elim. of a row}} = \\
&= 2t_f \frac{(N-1)\,(N(N-1)+1)}{6} \approx \frac{2}{3} N^3 t_f .
\end{aligned}
\tag{7}
$$

- As you can see from (6), $N \cdot T_P = O(N^3 \,\mathrm{ld}\, N)$ (consider $P = N$!) increases asymptotically faster than $T_S = O(N^3)$.
- The algorithm is thus not cost optimal (efficiency cannot be kept constant for $P = N \longrightarrow \infty$).
- The reason is, that processor P_k waits within its broadcast until all other processors have received the pivot row.
- We describe now an *asynchronous* variant, where a processor immediately starts calculating as soon as it receives the pivot row.

LU Decomposition: Asynchronous Variant

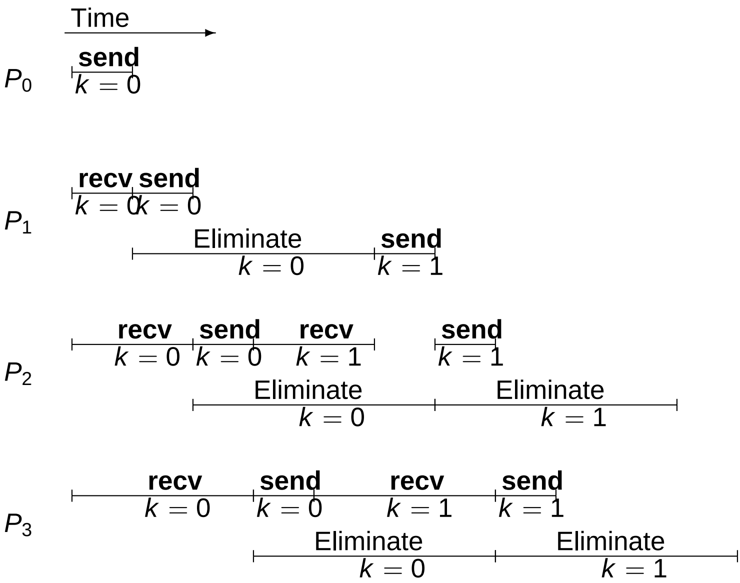
Program (Asynchronous LU decomposition for $P = N$)

```
parallel lu-1
{
    const int N = ...;
    process Π[int p ∈ {0, ..., N − 1}]
    {
        double A[N];                                  // my row
        double rr[2][N];                              // buffer for pivot row
        double ∗r;
        msgid m;
        int j, k;

        if (p > 0) m = arecv(Π_{p−1}, rr[0]);
        for (k = 0; k < N − 1; k++)
        {
            if (p == k) send(Π_{p+1}, A);
            if (p > k)
            {
                while (¬success(m));                  // wait for pivot row
                if (p < N − 1) asend(Π_{p+1}, rr[k%2]);
                if (p > k + 1) m = arecv(Π_{p−1}, rr[(k + 1)%2]);
                r = rr[k%2];
                A[k] = A[k]/r[k];
                for (j = k + 1; j < N; j++)
                    A[j] = A[j] − A[k] · r[j];
            }
        }
    }
}
```

LU Decomposition: Temporal Sequence

How does the parallel algorithm behave over time?



LU Decomposition: Parallel Runtime and Efficiency

- After a fill-in time of p message transmissions the pipeline is filled completely, and all processors are always busy with elimination. Then one obtains the following runtime ($N = P$, still!):

$$
\begin{aligned}
T_P(N) &= \underbrace{(N-1)(t_s + t_h + t_w N)}_{\text{fil-in time}} + \sum_{m=N-1}^{1} \Big(\underbrace{2mt_f}_{\text{elim.}} + \underbrace{t_s}_{\substack{\text{setup time}\\ \text{(compute+}\textbf{send}\\ \text{parallel)}}} \Big) = \qquad (8)\\
&= \frac{(N-1)N}{2} 2t_f + (N-1)(2t_s + t_h) + N(N-1)t_w \approx \\
&\approx N^2 t_f + N^2 t_w + N(2t_s + t_h).
\end{aligned}
$$

- The factor $\operatorname{ld} N$ of (6) is now vanished. For the efficiency we obtain

$$
\begin{aligned}
E(N,P) &= \frac{T_S(N)}{NT_P(N,P)} = \frac{\frac{2}{3}N^3 t_f}{N^3 t_f + N^3 t_w + N^2(2t_s + t_h)} = \qquad (9)\\
&= \frac{2}{3} \frac{1}{1 + \frac{t_w}{t_f} + \frac{2t_s + t_h}{Nt_f}}.
\end{aligned}
$$

- The efficiency is such limited by $\frac{2}{3}$. The reason for this is, that processor k remains after k steps idle. This can be avoided by more rows per processor (coarser granularity).

LU Decomposition: The Case $N \gg P$

LU decomposition for the **case** $N \gg P$:

- Program 0.1 from above can be easily extended to the case $N \gg P$. Herefore the *rows* are distributed cyclicly onto the processors $0, \ldots, P-1$. A processor's current pivot row is obtained from the predecessor in the ring.
- The parallel runtime is

$$
\begin{aligned}
T_P(N,P) &= \underbrace{(P-1)(t_s + t_h + t_w N)}_{\text{fill-in time of pipeline}} + \sum_{m=N-1}^{1} \Big(\underbrace{\frac{m}{P}}_{\substack{\text{rows per}\\ \text{processor}}} \cdot m 2 t_f + t_s \Big) = \\
&= \frac{N^3}{P}\frac{2}{3} t_f + N t_s + P(t_s + t_h) + N P t_w
\end{aligned}
$$

and thus one has the efficiency

$$
E(N,P) = \frac{1}{1 + \frac{P t_s}{N^2 \frac{2}{3} t_f} + \ldots}.
$$

LU Decomposition: The case $N \gg P$

- Because of row-wise partitioning applies however in average, that some processors have a row more than others.
- A still better load balancing is achieved by a two-dimensional partitioning of the matrix. Herefore we assume that the segmentation of the row and column index set

  $$I = J = \{0, \ldots, N-1\}$$

  is done with the mappings p and μ for I and q and ν for J.

LU decomposition: General Partitioning

- The following implementation is simplified, if we additonally assume, that the data partitioning fulfills the following monotony condition:

$$\begin{array}{lll} \text{Ist } i_1 < i_2 \text{ and } p(i_1) = p(i_2) & \text{such applies} & \mu(i_1) < \mu(i_2) \\ \text{ist } j_1 < j_2 \text{ and } q(j_1) = q(j_2) & \text{such applies} & \nu(j_1) < \nu(j_2) \end{array}$$

- Therefore an interval of global indices $[i_{min}, N-1] \subseteq I$ corresponds to a number of intervals of local indices in different processors, that can be calculated by:

$$\begin{array}{rcl} & \text{Set} & \\ \tilde{I}(p,k) & = & \{m \in \mathbf{N} \mid \exists i \in I, i \geq k : p(i) = p \wedge \mu(i) = m\} \\ & \text{and} & \\ ibegin(p,k) & = & \begin{cases} \min \tilde{I}(p,k) & \text{if } \tilde{I}(p,k) \neq \emptyset \\ N & \text{otherwise} \end{cases} \\ iend(p,k) & = & \begin{cases} \max \tilde{I}(p,k) & \text{if } \tilde{I}(p,k) \neq \emptyset \\ 0 & \text{otherwise.} \end{cases} \end{array}$$

- Then one can substitute a loop

  for $(i = k; i < N; i++) \ldots$

  by local loops in the processors p of shape

  for $(i = ibegin(p,k); i \leq iend(p,k); i++) \ldots$

LU Decomposition: General Partitioning

Analogous we perform with the column indices:

$$
\begin{aligned}
&\text{Set} \\
\tilde{J}(q,k) &= \{n \in \mathbf{N} \mid \exists j \in j, j \geq k : q(j) = q \wedge \nu(j) = n\} \\
&\text{and} \\
jbegin(q,k) &= \begin{cases} \min \tilde{J}(q,k) & \text{if } \tilde{J}(q,k) \neq \emptyset \\ N & \text{otherwise} \end{cases} \\
jend(q,k) &= \begin{cases} \max \tilde{J}(q,k) & \text{if } \tilde{J}(q,k) \neq \emptyset \\ 0 & \text{otherwise.} \end{cases}
\end{aligned}
$$

Now we can go on with the implementation of the *LU* decomposition for a general data partitioning.

LU Decomposition: Algorithm with General Partitioning

Program (Synchronous *LU* decompositon with general data partitioning)

```
parallel lu-2
{
    const int N = ..., √P = ...;

    process Π[int (p, q) ∈ {0, ..., √P − 1} × {0, ..., √P − 1}]
    {
        double A[N/√P][N/√P], r[N/√P], c[N/√P];
        int i, j, k;

        for (k = 0; k < N − 1; k++)
        {
            I = μ(k); J = ν(k);                                   // local indices

            // distribute pivot row:
            if (p == p(k))
            {                                                     // I have pivot row
                for (j = jbegin(q, k); j ≤ jend(q, k); j++)
                    r[j] = A[I][j];                               // copy segment of pivot row
                Send r to all processors (x, q) ∀x ≠ p
            }
            else recv(Π_{p(k),q}, r);

            // distribute pivot column:
            if (q == q(k))
            {                                                     // I have part of column k
                for (i = ibegin(p, k + 1); i ≤ iend(p, k + 1); i++)
                    c[i] = A[i][J] = A[i][J]/r[J];
                Send c to all processors (p, y) ∀y ≠ q
            }
            else recv(Π_{p,q(k)}, c);

            // elimination:
            for (i = ibegin(p, k + 1); i ≤ iend(p, k + 1); i++)
                for (j = jbegin(q, k + 1); j ≤ jend(q, k + 1); j++)
                    A[i][j] = A[i][j] − c[i] · r[j];
        }
    }
}
```

LU Decomposition: Analysis I

- Let us analyse this implementation (synchronous variant):

$$T_P(N,P) = \sum_{m=N-1}^{1} \underbrace{\left(t_s + t_h + t_w \frac{m}{\sqrt{P}}\right) \operatorname{ld}\sqrt{P}\, 2}_{\text{Broadcast pivot row/-column}} + \left(\frac{m}{\sqrt{P}}\right)^2 2t_f =$$

$$= \frac{N^3}{P}\frac{2}{3}t_f + \frac{N^2}{\sqrt{P}} \operatorname{ld}\sqrt{P} t_w + N \operatorname{ld}\sqrt{P}\, 2(t_s + t_h).$$

- Mit $W = \frac{2}{3}N^3 t_f$, d.h. $N = \left(\frac{3W}{2t_f}\right)^{\frac{1}{3}}$, gilt

$$T_P(W,P) = \frac{W}{P} + \frac{W^{\frac{2}{3}}}{\sqrt{P}} \operatorname{ld}\sqrt{P} \frac{3^{2/3} t_w}{(2t_f)^{\frac{2}{3}}} + W^{\frac{1}{3}} \operatorname{ld}\sqrt{P} \frac{3^{1/3} 2(t_s+t_h)}{(2t_f)^{\frac{1}{3}}}.$$

LU Decomposition: Analysis II

- The isoefficiency function can be obtained from $PT_P(W,P) - W \stackrel{!}{=} KW$:

$$\sqrt{P} W^{\frac{2}{3}} \operatorname{ld} \sqrt{P} \frac{3^{2/3} t_w}{(2t_f)^{\frac{2}{3}}} = KW$$

$$\iff \quad W = P^{\frac{3}{2}} (\operatorname{ld} \sqrt{P})^3 \frac{9t_w^3}{4t_f^2 K^3}$$

  thus

$$W \in O(P^{3/2} (\operatorname{ld} \sqrt{P})^3).$$

- Program 0.2 can also be realized in an asynchronous variant. Hereby the communication shares can be effectively hidden behind the calculation.

LU Decomposition: Pivoting

- The *LU* factorisation of general, invertible matrices requires pivoting and is also meaningful by reasons of minimisation of rounding errors.
- One speaks of full pivoting, if the pivot element in step k can be choosen from all $(N-k)^2$ remaining matrix elements, resp. of partial pivoting, if the pivot element can only be choosen from a part of the elements. Usual for example is the maximal row- or column pivot this means one chooses a_{ik}, $i \geq k$, with $|a_{ik}| \geq |a_{mk}| \quad \forall m \geq k$.
- The implementation of *LU* decomposition has now to consider the choice of the new pivot element during the elimination. Herefore one has two possibilites:
  - Explicit exchange of rows and/or columns: Here a rest of the algorithm then remains unchanged (for row exchanges the righthand side has to be permuted).
  - The actual data is not moved, but one remembers the interchange of indices (in an integer array, that maps old indices to new).

LU Decomposition: Pivoting

- The parallel versions have different properties regarding pivoting. The following points have to be considered for the parallel *LU* partitioning with partial pivoting:
  - If the area, in which the pivot element is searched, is stored in a single processor (e.g. row-wise partitioning with maximal row pivot), then the search is to be performed purely sequential. In the other case it can be parallelized.
  - But this parallel search for a pivot element requires communication (and such synchronisation), that renders the pipelining in the asynchronous variant impossible.
  - To permute the indices is faster than explicit exchange, especially if the exchange requries data exchange between processors. Besides that a favourable load balancing can such be distroyed, if randomly the pivot elements reside always in the same procesor.
- A quite good compromise is given by the row-wise cyclic partitioning with maximal row pivot and and explicit exchange, since:
  - pivot search in row k is pure sequential, but needs only $O(N-k)$ operations (compared to $O((N-k)^2/P)$ for the elimination); besides the pipelining is not destroyed.
  - explicit exchange requires only communication of the index of the pivot column, but no exchange of matrix elements between processors. The pivot column index is sent with the pivot row.
  - load balancing is not influenced by the pivoting.

LU Decomposition: Solution of Triangular Systems

- We assume the matrix A be factorized into $A = LU$ as above, and continue with the solution of the system of the form

$$LUx = b. \tag{10}$$

  This happens in two steps:

$$\begin{aligned} Ly &= b \qquad (11) \\ Ux &= y. \qquad (12) \end{aligned}$$

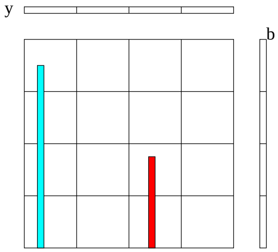
- We shortly consider the sequential algorithm:

```
// Ly = b:
for (k = 0; k < N; k++) {
      y_k = b_k;          l_kk = 1
      for (i = k+1; i < N; i++)
            b_i = b_i - a_ik y_k;
}
// Ux = y:
for (k = N-1; k ≥ 0; k--) {
      x_k = y_k / a_kk
      for (i = 0; i < k; i++)
            y_i = y_i - a_ik x_k;
}
```

- This is a column oriented version, since after calculation of y_k resp. x_k immediately the righthand side is modified for all indices $i > k$ resp. $i < k$.

LU Decomposition: Parallelisation

- The parallelisation has of course to be oriented at the data partitioning of the *LU* decomposition (if one wants to avoid copying, which seems not to be meaningful because of $O(N^2)$ data and $O(N^2)$ operations We consider for this a two-dimensional block-wise partitioning of the matrix:



- The sections of b are copied across processors rows and the sections of y are copied across the processor columns. Obviously after calculation of y_k only the processors of column $q(k)$ can be busy with the modification of b. According to that during the solution of $Ux = y$ only the processors $(*, q(k))$ can be busy at a time. Thus, with a row-wise partitioning ($Q = 1$) always all processors can be kept busy.

LU Decomposition: Parallelisation for General Partitioning

Program (Resolving of $LUx = b$ for general data partitioning)

```
parallel lu-solve
{
    const int N = ...;
    const int √P = ...;
    process Π[int (p, q) ∈ {0, ..., √P − 1} × {0, ..., √P − 1}]
    {
        double A[N/√P][N/√P];
        double b[N/√P]; x[N/√P];
        int i, j, k, I, K;

        // Solve Ly = b, store y in x.
        // b column-wise distributed onto diagonal processors.
        if (p == q) send b to all (p, ∗);
        for (k = 0; k < N; k++)
        {
            I = μ(k); K = ν(k);
            if(q(k) == q)                                   // only they have something to do
            {
                if (k > 0 ∧ q(k) ≠ q(k − 1))                // need current b
                    recv(Π_{p,q(k−1)}, b);
                if (p(k) == p)
                {                                           // have diagonal element
                    x[K] = b[I];                            // store y in x!
                    send x[K] to all (∗, q);
                }
                else recv(Π_{p(k),q(k)}, x[k]);
                for (i = ibegin(p, k + 1); i ≤ iend(p, k + 1); i++)
                    b[i] = b[i] − A[i][K] · x[K];
                if (k < N − 1 ∧ q(k + 1) ≠ q(k))
                    send(Π_{p,q(k+1)}, b);
            }
        }
        ...
```

LU Decomposition: Parallelisation

Program (Resolving of $LUx = b$ for general data partitioning cont.)

```
parallel lu-solve cont.
{

        . . .
        // { y is stored in x; x is distributed colum-wise and is copied row-wise. For Ux = y we want to store y in b.
           { It is such to copy x into b, where b shall be distributed row-wise and copied column-wise.
        for (i = 0; i < N/√P; i++)                                         // extinquish
                b[i] = 0;
        for (j = 0; j < N − 1; j++)
                if (q(j) = q ∧ p(j) = p)                                   // one has to be it
                        b[μ(j)] = x[ν(j)];
        sum b across all (p, ∗), result in (p, p);

        // Resolving of Ux = y (y is stored in b)
        if (p == q) send b and all (p, ∗);
        for (k = N − 1; k ≥ 0; k − −)
        {
                I = μ(k); K = ν(k);
                if (q(k) == q)
                {
                        if (k < N − 1 ∧ q(k) ≠ q(k + 1))
                                recv(Π_{p,q(k+1)}, b);
                        if (p(k) == p)
                        {
                                x[K] = b[I]/A[I][K];
                                send x[K] to all (∗, q);
                        }
                        else recv(Π_{p(k),q(k)}, x[K]);
                        for (i = ibegin(p, 0); i ≤ iend(p, 0); i++)
                                b[i] = b[i] − A[i][K] · x[K];
                        if (k > 0 ∧ q(k) ≠ q(k − 1))
                                send(Π_{p,q(k−1)}, b);
                }
        }
    }
}
```

LU Decomposition: Parallelisation

- Since at a time always only $\sqrt{P}$ processors are busy, the algorithm cannot be cost optimal. The total scheme consisting of LU decomposition and solution of triangular systems can still always be scaled iso-efficiently, since the sequential complexity of solution is only $O(N^2)$ compared to $O(N^3)$ for the factorisation.
- If one needs to solve the equation system for many righthand sides, one should use a rectangular processor array $P \times Q$ with $P > Q$, or in the extreme case choose as $Q = 1$. If pivoting has been required, this was already a meaningful configuration.