Solving Triangular Systems

1. Forward Substitution Formulas
   - processing the L of the LU factorization
   - a third type of pipeline
2. Parallel Solving
   - using an *n*-stage pipeline
   - rewriting the formulas
   - a parallel solver with OpenMP

MCS 572 Lecture 17
Introduction to Supercomputing
Jan Verschelde, 30 September 2016

Solving Triangular Systems

1. Forward Substitution Formulas
   - processing the L of the LU factorization
   - a third type of pipeline
2. Parallel Solving
   - using an *n*-stage pipeline
   - rewriting the formulas
   - a parallel solver with OpenMP

LU factorization

To solve an n-dimensional linear system $A\mathbf{x} = \mathbf{b}$
we factor A as a product of two triangular matrices, $A = LU$:

- L is lower triangular, $L = [\ell_{i,j}]$, $\ell_{i,j} = 0$ if $j > i$ and $\ell_{i,i} = 1$.
- U is upper triangular $U = [u_{i,j}]$, $u_{i,j} = 0$ if $i > j$.

Solving $A\mathbf{x} = \mathbf{b}$ is equivalent to solving $L(U\mathbf{x}) = \mathbf{b}$:

1. Forward substitution: $L\mathbf{y} = \mathbf{b}$.
2. Backward substitution: $U\mathbf{x} = \mathbf{y}$.

Factoring A costs $O(n^3)$, solving triangular systems costs $O(n^2)$.

formulas for forward substitution

Expanding the matrix-vector product $L\mathbf{y}$ in $L\mathbf{y} = \mathbf{b}$ leads to

$$\begin{cases} y_1 & = b_1 \\ \ell_{2,1} y_1 + y_2 & = b_2 \\ \ell_{3,1} y_1 + \ell_{3,2} y_2 + y_3 & = b_3 \\ & \vdots \\ \ell_{n,1} y_1 + \ell_{n,2} y_2 + \ell_{n,3} y_3 + \cdots + \ell_{n,n-1} y_{n-1} + y_n & = b_n \end{cases}$$

and solving for the diagonal elements gives

$$\begin{aligned} y_1 &= b_1 \\ y_2 &= b_2 - \ell_{2,1} y_1 \\ y_3 &= b_3 - \ell_{3,1} y_1 - \ell_{3,2} y_2 \\ &\vdots \\ y_n &= b_n - \ell_{n,1} y_1 - \ell_{n,2} y_2 - \cdots - \ell_{n,n-1} y_{n-1} \end{aligned}$$

formula and algorithm

For $k = 1, 2, \ldots, n$:

$$y_k = b_k - \sum_{i=1}^{k-1} \ell_{k,i} y_i.$$

As an algorithm:

for k from 1 to n do
  $y_k := b_k$;
  for i from 1 to $k-1$ do
    $y_k := y_k - \ell_{k,i} \star y_i$.

We count

$$1 + 2 + \cdots + n - 1 = \frac{n(n-1)}{2}$$

multiplications and subtractions.

Solving Triangular Systems

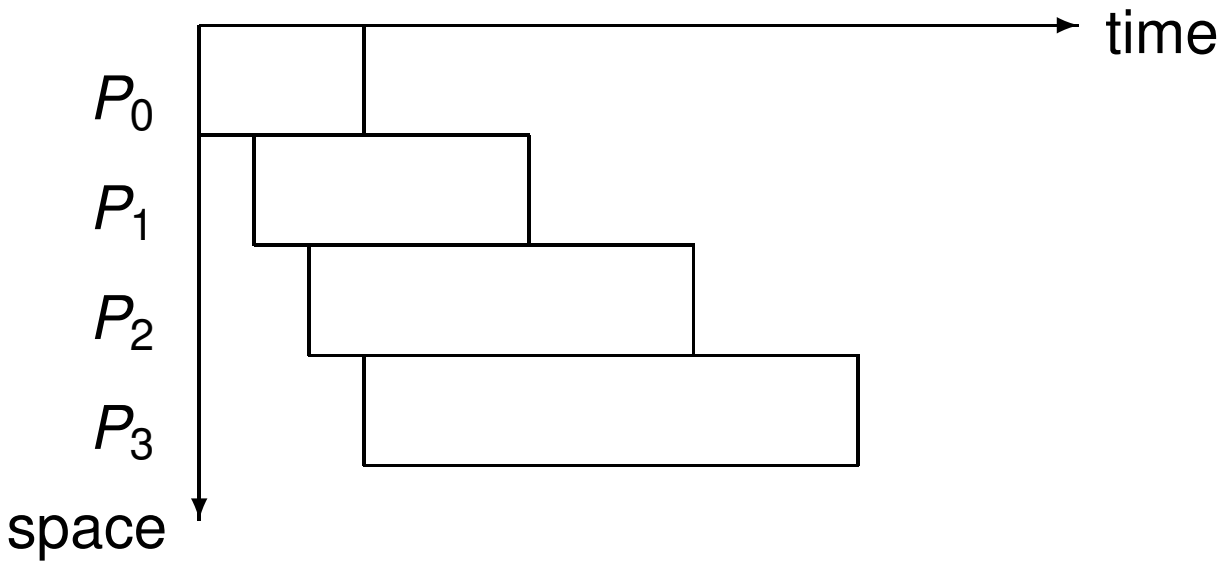
1. Forward Substitution Formulas
   - processing the L of the LU factorization
   - a third type of pipeline
2. Parallel Solving
   - using an *n*-stage pipeline
   - rewriting the formulas
   - a parallel solver with OpenMP

a third type of pipeline

Three types of pipelines:

1. Speedup only if multiple instances. Example: instruction pipeline.
2. Speedup already if one instance. Example: pipeline sorting.
3. Worker continues after passing information through.
   Example: solve $L\mathbf{y} = \mathbf{b}$.

Typical for the 3rd type of pipeline is the varying length of each job.



Solving Triangular Systems

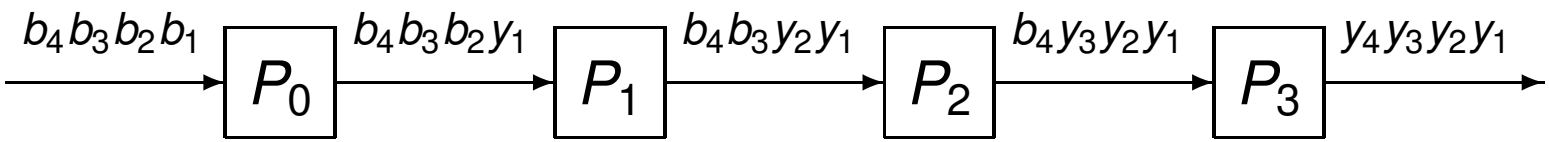
1. Forward Substitution Formulas
   - processing the L of the LU factorization
   - a third type of pipeline

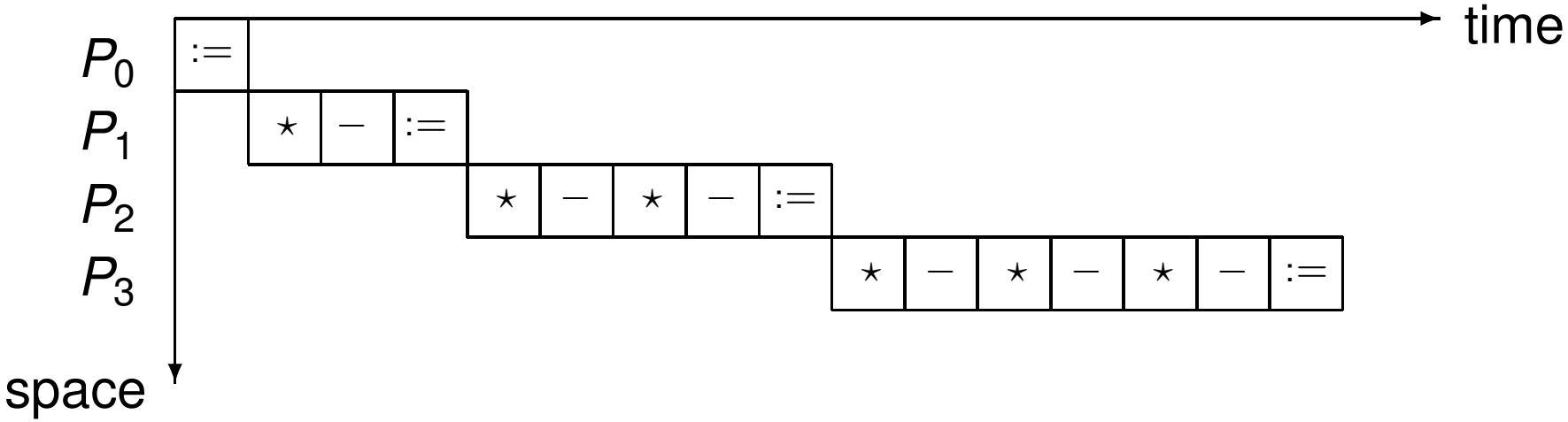
2. Parallel Solving
   - using an *n*-stage pipeline
   - rewriting the formulas
   - a parallel solver with OpenMP

using an n-stage pipeline

We assume that L is available on every processor.

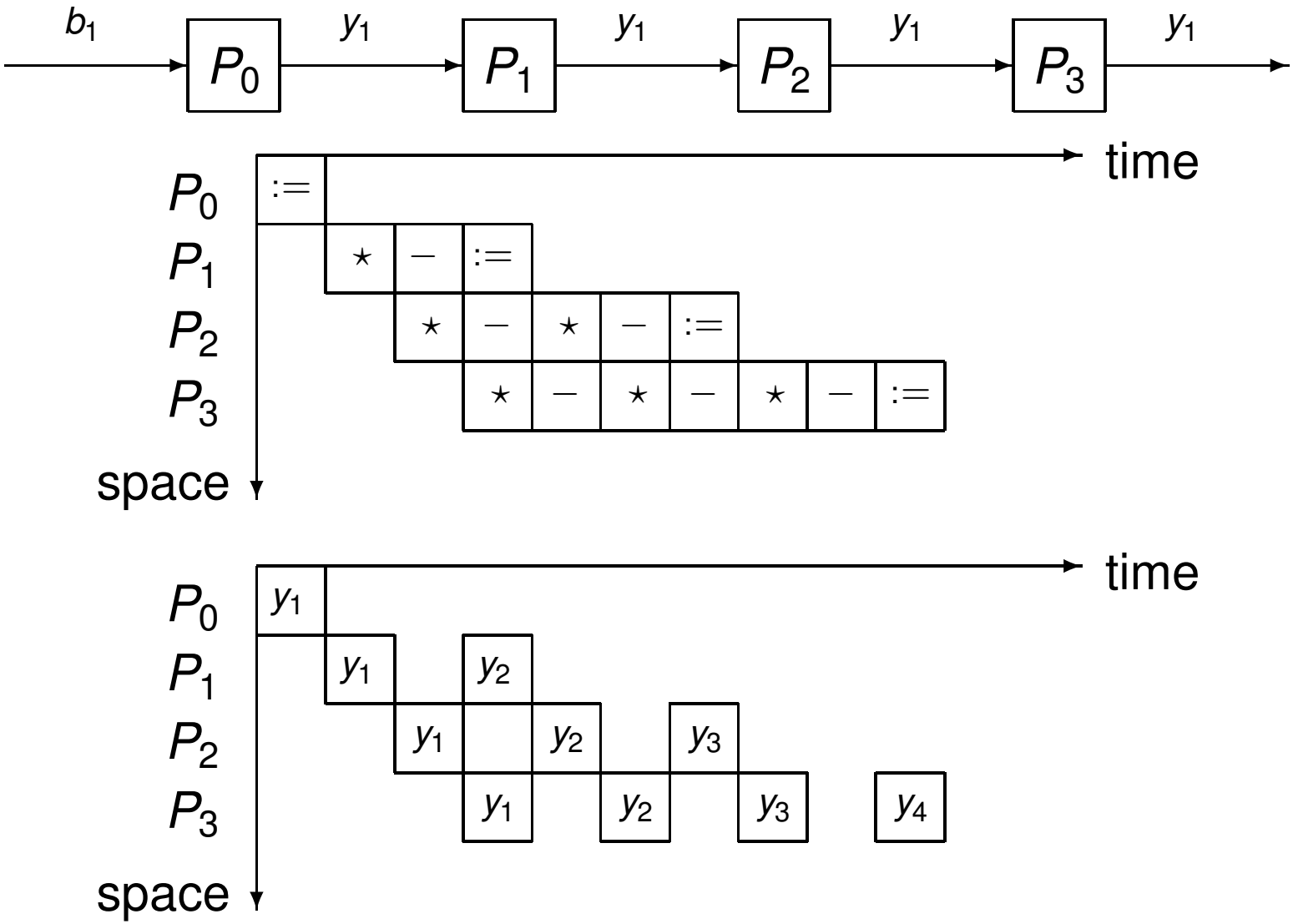
$$\text{for } n = 4 = p: \quad \begin{aligned} y_1 &:= b_1 \\ y_2 &:= b_2 - \ell_{2,1} \star y_1 \\ y_3 &:= b_3 - \ell_{3,1} \star y_1 - \ell_{3,2} \star y_2 \\ y_4 &:= b_4 - \ell_{4,1} \star y_1 - \ell_{4,2} \star y_2 - \ell_{4,3} \star y_3 \end{aligned}$$



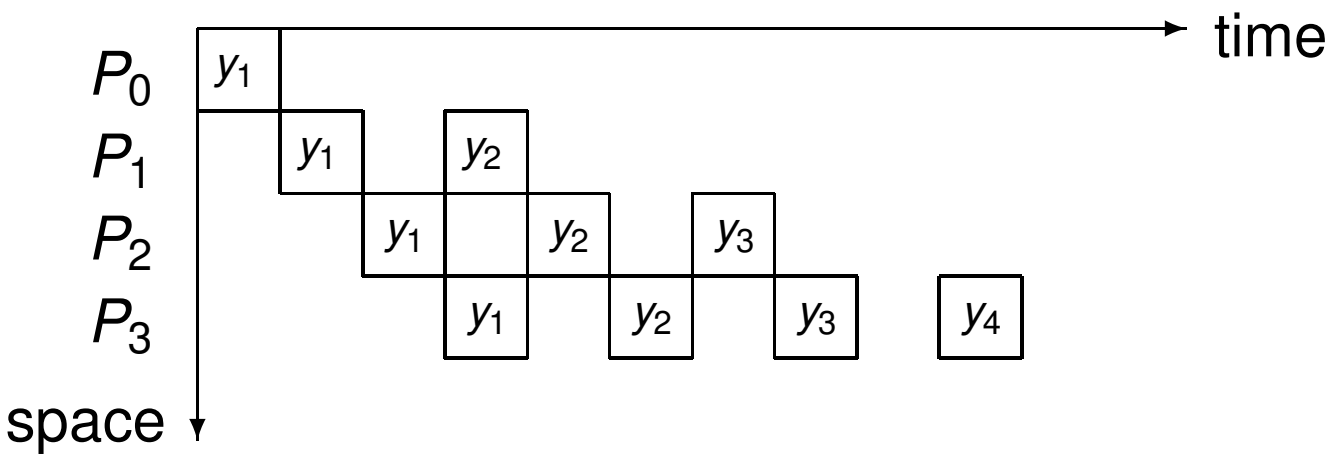


type 3 pipelining

Make y_1 available in the next pipeline cycle:



counting the steps



We count the steps for $p = 4$ or in general, for $p = n$:

1. The latency takes 4 steps for y_1 to be at P_4,
   or in general: n steps for y_1 to be at P_n.
2. It takes then 6 additional steps for y_4 to be computed by P_4,
   or in general: $2n - 2$ additional steps for y_n to be computed by P_n.

So it takes $n + 2n - 2 = 3n - 2$ steps to solve
an n-dimensional triangular system by an n-stage pipeline.

Solving Triangular Systems

1. Forward Substitution Formulas
   - processing the L of the LU factorization
   - a third type of pipeline
2. Parallel Solving
   - using an *n*-stage pipeline
   - rewriting the formulas
   - a parallel solver with OpenMP

rewriting the formulas

Solving $L\mathbf{y} = \mathbf{b}$ for $n = 5$:

1. $y := \mathbf{b}$;
2. $y_2 := y_2 - \ell_{2,1} \star y_1$;
   $y_3 := y_3 - \ell_{3,1} \star y_1$;
   $y_4 := y_4 - \ell_{4,1} \star y_1$;
   $y_5 := y_5 - \ell_{5,1} \star y_1$;
3. $y_3 := y_3 - \ell_{3,2} \star y_2$;
   $y_4 := y_4 - \ell_{4,2} \star y_2$;
   $y_5 := y_5 - \ell_{5,2} \star y_2$;
4. $y_4 := y_4 - \ell_{4,3} \star y_3$;
   $y_5 := y_5 - \ell_{5,3} \star y_3$;
5. $y_5 := y_5 - \ell_{5,4} \star y_4$;

$\mathbf{y} := \mathbf{b}$;
for i from 2 to n do
    for j from i to n do
        $y_j := y_j - \ell_{j,i-1} \star y_{i-1}$;

$\Rightarrow$ all instructions in the j loop are independent from each other!

data parallelism

Consider the inner loop in the algorithm to solve $L\mathbf{y} = \mathbf{b}$:

$\mathbf{y} := \mathbf{b}$;
for i from 2 to n do
  for j from i to n do
    $y_j := y_j - \ell_{j,i-1} \star y_{i-1}$;

We distribute the update of $y_i, y_{i+1}, \ldots, y_n$ among p processors.

If $n \gg p$, then we expect a close to optimal speedup.

Solving Triangular Systems

1. Forward Substitution Formulas
   - processing the L of the LU factorization
   - a third type of pipeline

2. Parallel Solving
   - using an *n*-stage pipeline
   - rewriting the formulas
   - a parallel solver with OpenMP

a parallel solver

For our parallel solver for triangular systems:

- For $L = [\ell_{i,j}]$, we generate random numbers for $\ell_{i,j} \in [0, 1]$.
  The exact solution $\mathbf{y}$: $y_i = 1$, for $i = 1, 2, \ldots, n$.
  We compute the right hand side $\mathbf{b} = L\mathbf{y}$.

- For dimensions $n > 800$, hardware doubles are insufficient.
  With hardware doubles, the accumulation of round off is such that we loose all accuracy in y_n.
  We recall that condition numbers of n-dimensional triangular systems can grow as large as 2^n.
  Therefore, we use quad double arithmetic.

- We use OpenMP, because it is Friday and OpenMP is easy.

hardware doubles are too inaccurate

Relying on hardware doubles is problematic:

```
$ time /tmp/trisol 10
last number : 1.0000000000000009e+00

real    0m0.003s    user    0m0.001s    sys    0m0.002s

$ time /tmp/trisol 100
last number : 9.9999999999974221e-01

real    0m0.005s    user    0m0.001s    sys    0m0.002s

$ time /tmp/trisol 1000
last number : 2.7244600009080568e+04

real    0m0.036s    user    0m0.025s    sys    0m0.009s
```

a matrix of quad doubles

Allocating data in the main program:

```
{
   qd_real b[n],y[n];
   int i,j;

   qd_real **L;
   L = (qd_real**) calloc(n,sizeof(qd_real*));
   for(i=0; i<n; i++)
      L[i] = (qd_real*) calloc(n,sizeof(qd_real));

   srand(time(NULL));
   random_triangular_system(n,L,b);
```

a random triangular system

```
void random_triangular_system
  ( int n, qd_real **L, qd_real *b )
{
   int i,j;
   for(i=0; i<n; i++)
   {
      L[i][i] = 1.0;
      for(j=0; j<i; j++)
      {
         double r = ((double) rand())/RAND_MAX;
         L[i][j] = qd_real(r);
      }
      for(j=i+1; j<n; j++)
         L[i][j] = qd_real(0.0);
   }
   for(i=0; i<n; i++)
   {
      b[i] = qd_real(0.0);
      for(j=0; j<n; j++)
         b[i] = b[i] + L[i][j];
   }
}
```

solving the system

```
void solve_triangular_system_swapped
  ( int n, qd_real **L, qd_real *b, qd_real *y )
{
   int i,j;

   for(i=0; i<n; i++) y[i] = b[i];

   for(i=1; i<n; i++)
   {
      for(j=i; j<n; j++)
         y[j] = y[j] - L[j][i-1]*y[i-1];
   }
}
```

using OpenMP

```
void solve_triangular_system_swapped
  ( int n, qd_real **L, qd_real *b, qd_real *y )
{
   int i,j;

   for(i=0; i<n; i++) y[i] = b[i];

   for(i=1; i<n; i++)
   {
      #pragma omp parallel shared(L,y) private(j)
      {
         #pragma omp for
         for(j=i; j<n; j++)
            y[j] = y[j] - L[j][i-1]*y[i-1];
      }
   }
}
```

experimental timings

running `time /tmp/trisol_qd_omp n p`
On dimension $n = 8,000$ for varying number p of cores.

p	cpu time	real	user	sys
1	21.240s	35.095s	34.493s	0.597s
2	22.790s	25.237s	36.001s	0.620s
4	22.330s	19.433s	35.539s	0.633s
8	23.200s	16.726s	36.398s	0.611s
12	23.260s	15.781s	36.457s	0.626s

The serial part is the generation of the random numbers for L and the computation of $\mathbf{b} = L\mathbf{y}$. Recall Amdahl's Law.

We can compute the serial time, subtracting for $p = 1$, from the real time the cpu time spent in the solver, i.e.: $35.095 - 21.240 = 13.855$.
For $p = 12$, time spent on the solver is $15.781 - 13.855 = 1.926$.
Compare 1.926 to 21.240/12 = 1.770.

Summary + Exercises

We ended chapter 5 in the book of Wilkinson and Allen.

Exercises:

1. Consider the upper triangular system $U\mathbf{x} = \mathbf{y}$, with $U = [u_{i,j}]$, $u_{i,j} = 0$ if $i > j$. Derive the formulas and general algorithm to compute the components of the solution $\mathbf{x}$. For $n = 4$, draw the third type of pipeline.
2. Write a parallel solver with OpenMP to solve $U\mathbf{x} = \mathbf{y}$. Take for U a matrix with random numbers in $[0, 1]$, compute $\mathbf{y}$ so all components of $\mathbf{x}$ equal one. Test the speedup of your program, for large enough values of n and a varying number of cores.
3. Describe a parallel solver for upper triangular systems $U\mathbf{y} = \mathbf{b}$ for distributed memory computers. Write a prototype implementation using MPI and discuss its scalability.