Plan

Last time

- some terminology
- parallel computing models (SISD, SIMD, MIMD)
- measures of ”goodness” (speedup, efficiency, scalability)
- Amdahl law and its variations (parallel fraction)

Today

- Functional decomposition and task scheduling
- Domain decomposition applied to matrix operations
- Gaussian elimination

Partitioning (or decomposition)

- break the problem into ”chunks” that can be assigned to concurrent tasks
- functional decomposition and domain decomposition.

Functional Decomposition:

- the problem is decomposed according to the types of work that must be performed, different types are assigned to different tasks.

Domain Decomposition:

- the data associated with the problem is decomposed, tasks works on portions of the data

Scheduling

Execute several functions on the same data array:

- average,
- minimum,
- geometric mean
- etc.

No dependencies between the tasks, so all can run in parallel

Video streaming

- Read data from server
- Parse data
- decode sound and decode video
- play sound and draw video frame

Domain deomposition

- Add two matrices
- Local filtering (convolve with a mask)
- Integration via Riemann sums
- etc.

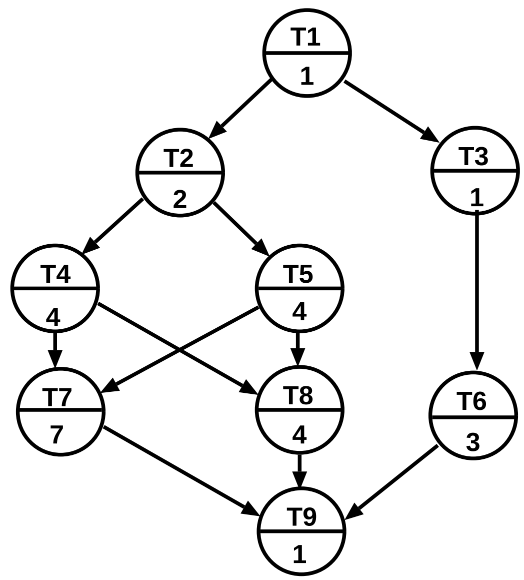
Scheduling is a scheme for assigning tasks to PEs

- so the parallel execution time is less than sequential execution time.

Task graph $G = (V, E)$

- $V = \{T_i\}$ set of nodes with $T_i \in V$ representing task i
- $E = \{e_{ij}\} = \{(i, j)\}$ set of directed edges representing dependencies among the tasks
- $w(T_i)$ is the (sequential) execution cost of T_i (or weight of T_i)
  - **may** include the cost of communicating data,
  - $w(T_i)$ determines the **granularity** of the dependence graph.
- no circular dependencies (DAG)

Task graph



For $e_{ij} \in E$

- T_i is a **predecessor** of T_j,
- the number of predecessors of T_j is called the **indegree** of T_j,
- T_j is a **successor** of T_i.
- the number of successors of T_i is called the outdegree of T_i,

Scheduling problem: assign T_i's to P PEs "optimally".

In general a very difficult problem (NP-problem).
Good heuristics exists.

List scheduling

1. assign tasks to **levels** - dynamic programming
2. based on levels create a **priority queue**
3. extract from the priority queue a **ready queue**
4. assign idle PEs to tasks from the ready queue (in some order)
5. after finishing a task update the priority and ready queues
6. repeat from (4) until done

Assume one start node T_0 and one terminal node T_{end}.

Def 3.1: A **path** $p(s,e)$ joining T_s and T_e is a chain of edges (i_k, i_{k+1}), $k = 0, .., n$ s.t. $i_0 = s$ and $i_{n+1} = e$,

$$p(s,e) = \{(i_0, i_1), ..., (i_n, i_{n+1})\}.$$

The **length** of $p(s,e)$ is the number of edges in $p(s,e)$,

$$\text{length}(p(s,e)) \stackrel{def}{=} |p(s,e)| = n + 1.$$

Def 3.2: The **weight** $w(p(s,e))$ of $p(s,e) = \{(i_0, i_1), ..., (i_n, i_{n+1})\}$ is

$$w(p(s,e)) = \sum_{j=0}^{n+1} w(T_{i_j})$$

. **Def 3.3**: Let $\mathcal{P}(s,e)$ be the set of all paths between T_s and T_e. A **critical path** between T_s and T_e is a path of the largest weight,

$$p_{crit}(s,e) = \text{argmax}_{p(s,e)\in\mathcal{P}(s,e)} w(p(s,e))$$

Def 3.4: The **level** l_i of task T_i is defined as

$$l_i = |p_{crit}(T_i, T_{end})|$$

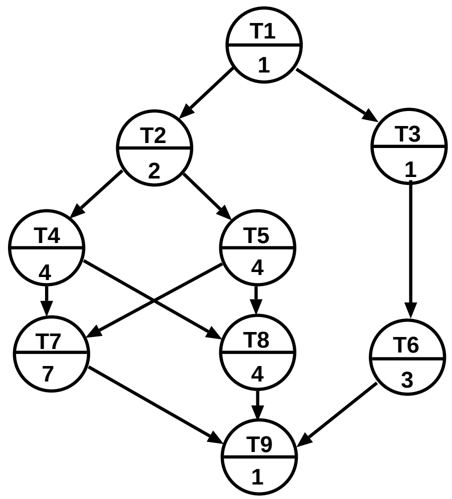
Step 1: Get the critical path for all tasks. Use Dynamic Programming.

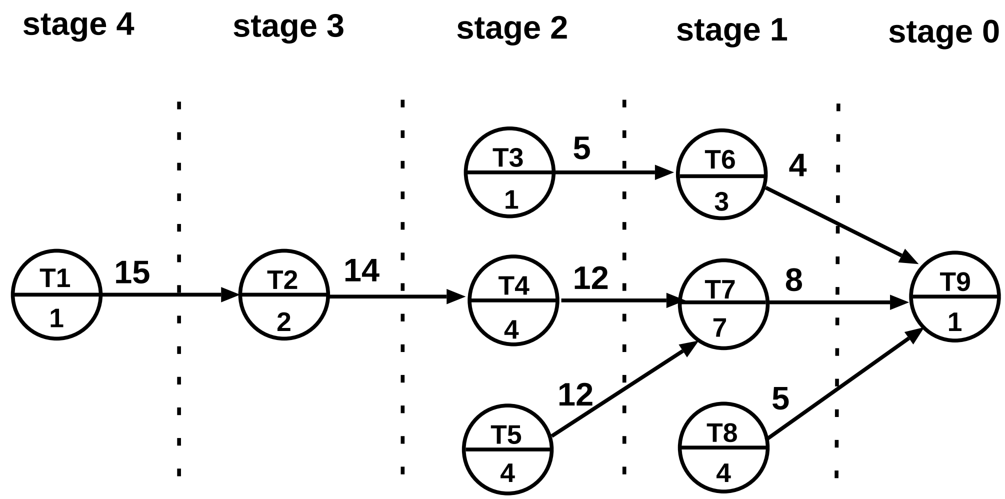
In stage s of DP nodes from set $V^{(s)}$ are considered.

Start with stage $s = 0$ and set $V^{(0)} = \{T_{end}\}$.
Loop:

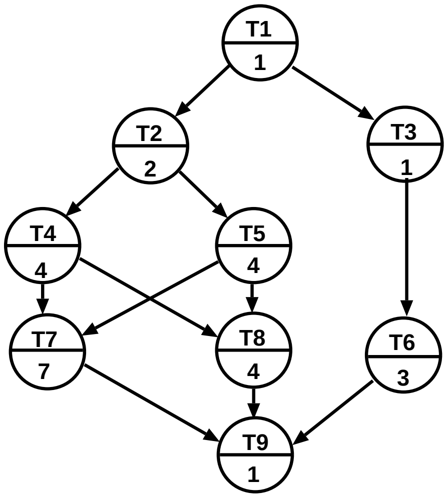
1. if the nodes in stage s have indegree 0 then exit
2. $s = s + 1$,
3. set $V^{(s)}$ as the set of all predecessor of nodes in $V^{(s-1)}$ (whose successors had their critical path already evaluated)
4. for $T_i^{(s)} \in V^{(s)}$ set
$$w(p_{crit}(T_i^{(s)}, T_{end})) = w(T_i^{(s)}) + \max_{T_j^{(s-1)},\ (T_i^{(s)}, T_j^{(s-1)}) \in E} w(p_{crit}(T_j^{(s-1)}, T_{end}))$$
5. go back to *loop*

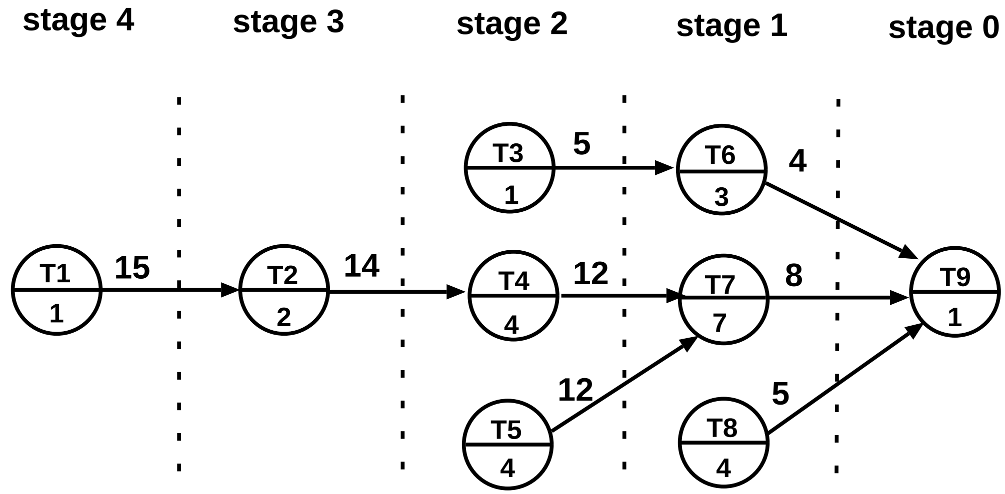
Task levels


T1
1
T2
2
T3
1
T4
4
T5
4
T7
7
T8
4
T6
3
T9
1


stage 4
stage 3
stage 2
stage 1
stage 0
T1
1
15
T2
2
14
T3
1
5
T6
3
4
T4
4
12
T7
7
8
T9
1
T5
4
12
T8
4
5

Priority queue





Order task by stage (level)

Within level order by weight

Priority queue: $\{T1, T2, T4, T5, T3, T7, T8, T6, T9\}$

Priority queue

(1) Form a **ready queue** from the tasks in the priority queue which do not have any predecessors in the task graph still in the queue.

(2) Assign tasks from the ready queue to processors in decreasing order of priorities.

(3) When a task is executed remove the task from the list and update the ready queue.

Scheduling example

Ready queue	updated priority list	PE1	PE2	time
$\{T1\}$	$\{T2, T4, T5, T3, T7, T8, T6, T9\}$	$T1$	idle	0
$\{T2, T3\}$	$\{T2, T4, T5, T7, T8, T6, T9\}$	$T2$	$T3$	1
$\{T6\}$	$\{T2, T4, T5, T7, T8, T6, T9\}$	$T2$	$T6$	2
$\{T4, T5\}$	$\{T4, T5, T7, T8, T6, T9\}$	$T4$	$T6$	3
$\{T5\}$	$\{T4, T5, T7, T8, T9\}$	$T4$	$T5$	5
$\{\}$	$\{T5, T7, T8, T9\}$	idle	$T5$	7
$\{T7, T8\}$	$\{T7, T8, T9\}$	$T7$	$T8$	9
$\{\}$	$\{T7, T9\}$	$T7$	idle	13
$\{T9\}$	$\{T9\}$	$T9$	idle	16
$\{\}$	$\{\}$	idle	idle	17

Does the heuristic produce the optimal schedule?

The worst case performance of the heuristic for m processors is

$$\frac{t - t_{opt}}{t_{opt}} \leq 1 - \frac{1}{P} \; .$$

The optimal schedule is at most twice as long as the derived one.

The lower bound on the perfomance is given by the critical path of T_0.

Domain Decomposition:

- the data associated with the problem is decomposed, tasks work on portions of the data
- common for matrix like operations
  - assign rows, columns or submatrices of a matrix to different PEs

PRAM matrix-matrix multiply

$C = A \cdot B \Rightarrow c(i,j) \leftarrow \sum_{k=1}^{n} a(i,k)b(k,j)$

For a PRAM with "infinite" number of PEs:

- cost of a single "dot" product

For a PRAM with P PEs:

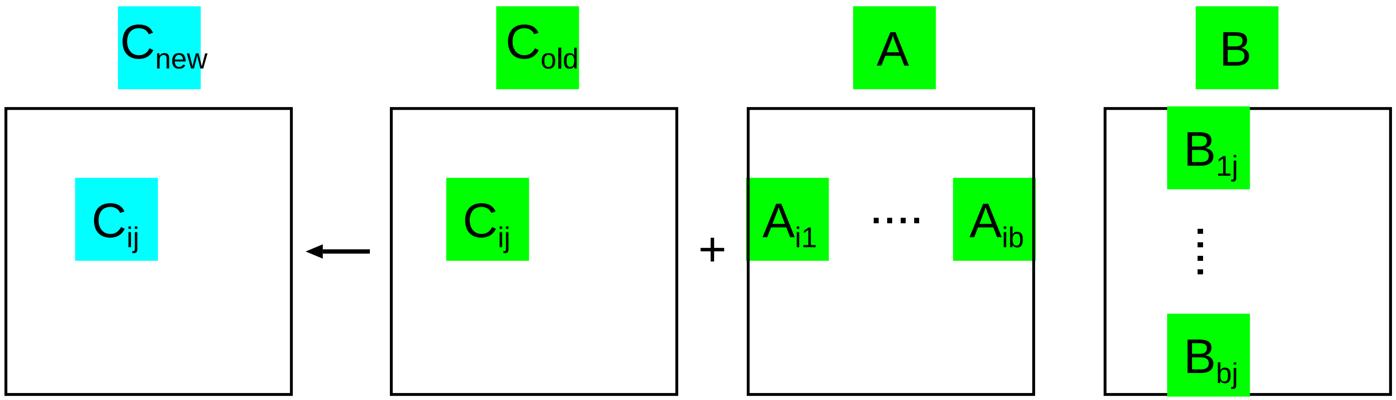
- assign $\frac{n}{P}$ elements $c(i,j)$ of C to each PE
- compute dot products defining $c(i,j)$

Now consider a shared memory system composed of P CPUs, each with individual cache.

Block matrix-matrix multiply

For a single CPU we split A, B, C into blocks of size $b \times b$ (thus $n = b \cdot p$).

$C(i,j) \leftarrow C(i,j) + \sum_{k=1}^{p} A(i,k)B(k,j)$ still holds.



Block matrix-matrix multiply

Sequential algorithm $C(i,j) \leftarrow C(i,j) + \sum_{k=1}^{p} A(i,k)B(k,j)$

for $i = 1 : p$
    for $j = 1 : p$
        load block $C(i,j)$ into fast memory
        for $k = 1 : p$
            load block $A(i,k)$ into fast memory
            load block $B(k,j)$ into fast memory
            $C(i,j) \leftarrow C(i,j) + A(i,k)B(k,j)$
        store $C(i,j)$ into slow memory

Parallel algorithm with P PEs:

- how do we distribute work over PEs ?

Block matrix-matrix multiply

- Hopefully $P = p^2$
- Split C into P subblocks of size $\frac{n}{p} \times \frac{n}{P}$.
- Execute the sequential block matrix-matrix multiply on each CPU
- P CPUs want to load $A(i,k)$ and $B(k,j)$ simultaneously. Does this create congestion between shared memory and individual caches?
- Good final project.

All node shortest path

$G = (V, E)$, $V = \{1, 2, ..., n\}$. Each edge $(i, j) \in E$ is assigned a non-negative distance (weight) $w(i, j)$.

Find shortest paths joining any pair of nodes.

Define a matrix $W = \{w_{ij}\}$ as follows

$$w_{ij} = \begin{cases} 0 & i = j \\ w_{ij} & i \neq j, \ \ (i, j) \in E \\ \infty & i \neq j, \ \ (i, j) \notin E \end{cases}$$

Also define another matrix $D^{(m)} = \{d_{ij}^{(m)}\}$ where

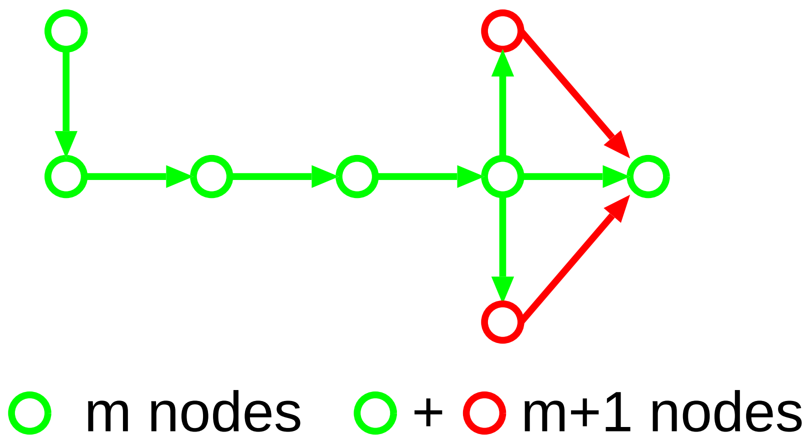
$d_{ij}^{(m)}$ is a shortest path joining i and j containing at most m nodes

All node shortest path

Assume $D^{(m)}$ is known. We want to compute $D^{(m+1)}$.

When $m = n - 1$ then $D^{(m+1)}$ gives us all node shortest path.

How can we get $D^{(m+1)}$ from $D^{(m)}$?



Consider a path from i to j through k where

- $p(i,k)$ has at most $m-1$ edges
- $p(k,j)$ has a single edge

We have

$$d_{ij}^{(m+1)} = \min_{1\leq k\leq n}(d_{ik}^{(m)} + w_{kj})$$

Note, $d_{ik}^{(m)}$ can be ∞.

Notice the correspondence

	matrix multiply	$\Leftrightarrow$		minimum sum
L:	$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$	$\Leftrightarrow$	R:	$d_{ij}^{(m+1)} = \min_{1 \le k \le n}(d_{ik}^{(m)} + w_{kj})$

"$\cdot$" on the left corersponds to "+" on the right
"+" on the left corresponds to "min" on the right

Computation of $D^{(m+1)}$ can be viewed as multiplication of $D^{(m+1)}$ with W with appropriate interpretation of elementary operations involved.

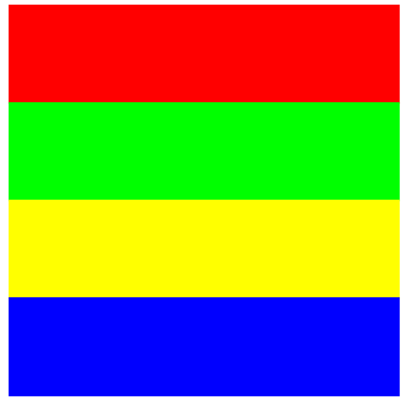
Cost ?

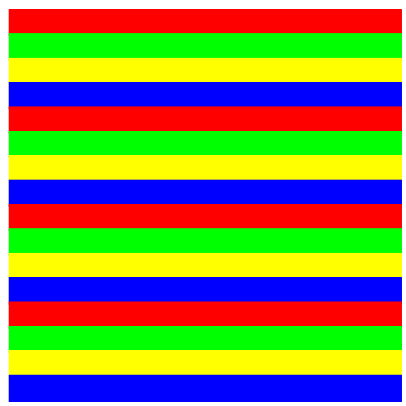
Note that

$$D^{(2)} =?, \;\; D^{(m)} =?$$

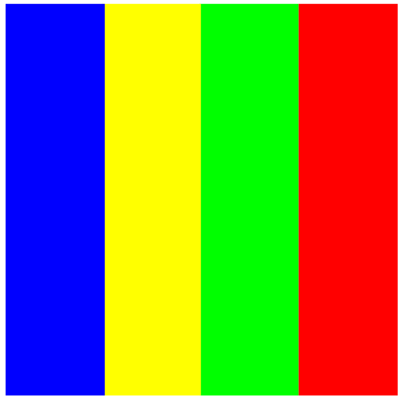
Good matrix-matrix multiplication algorithm needed.

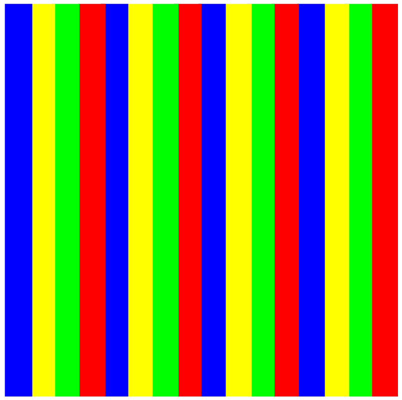
Matrix distributions



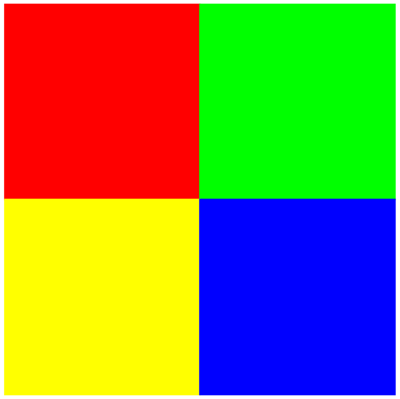


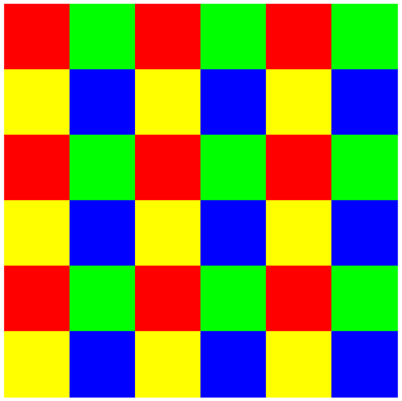
Matrix distributions





Matrix distributions





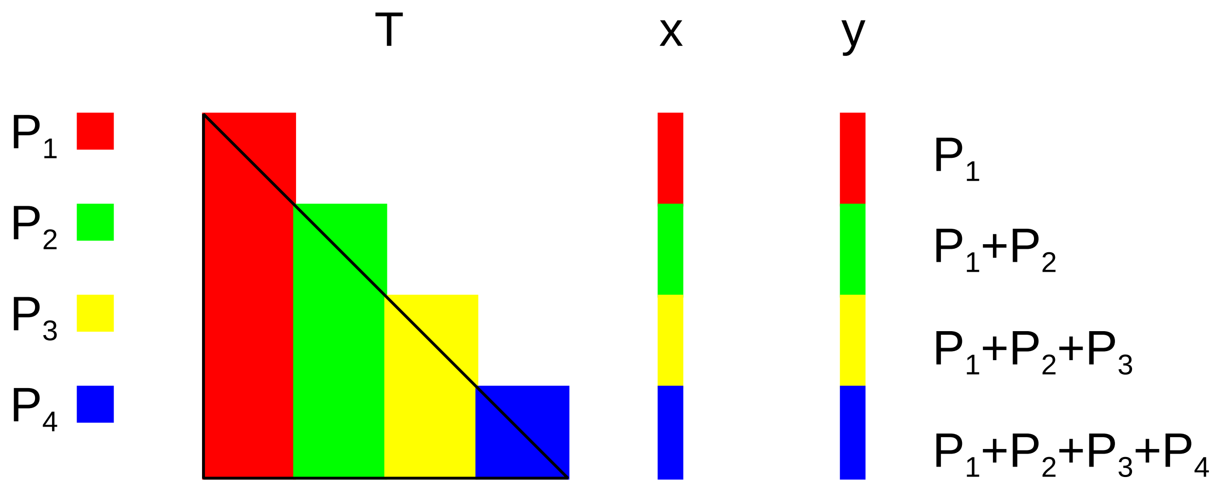
Triangular matrix multiply

Matrices that are not "uniform".

$$
y = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} t_{11} & & & & \\ t_{21} & t_{22} & & & \\ t_{31} & t_{32} & t_{33} & & \\ \vdots & & & \ddots & \\ t_{n1} & t_{n2} & & \cdots & t_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} = Tx
$$

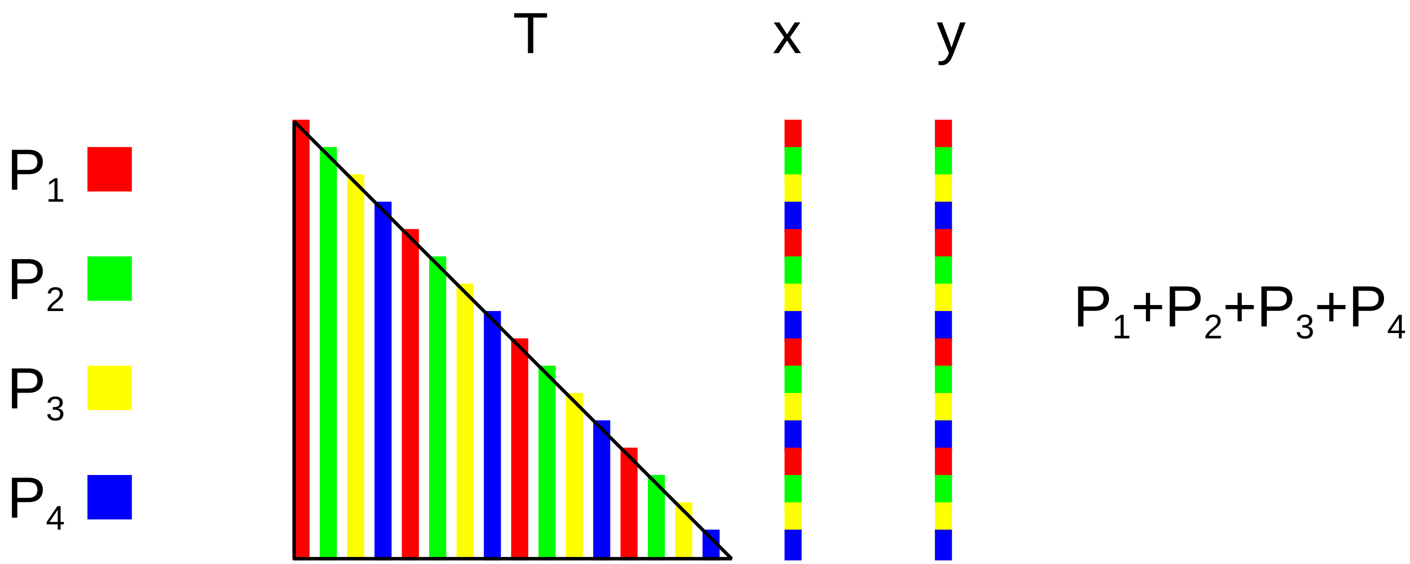
How do we distribut work among PEs ?

1D block-column distribution:



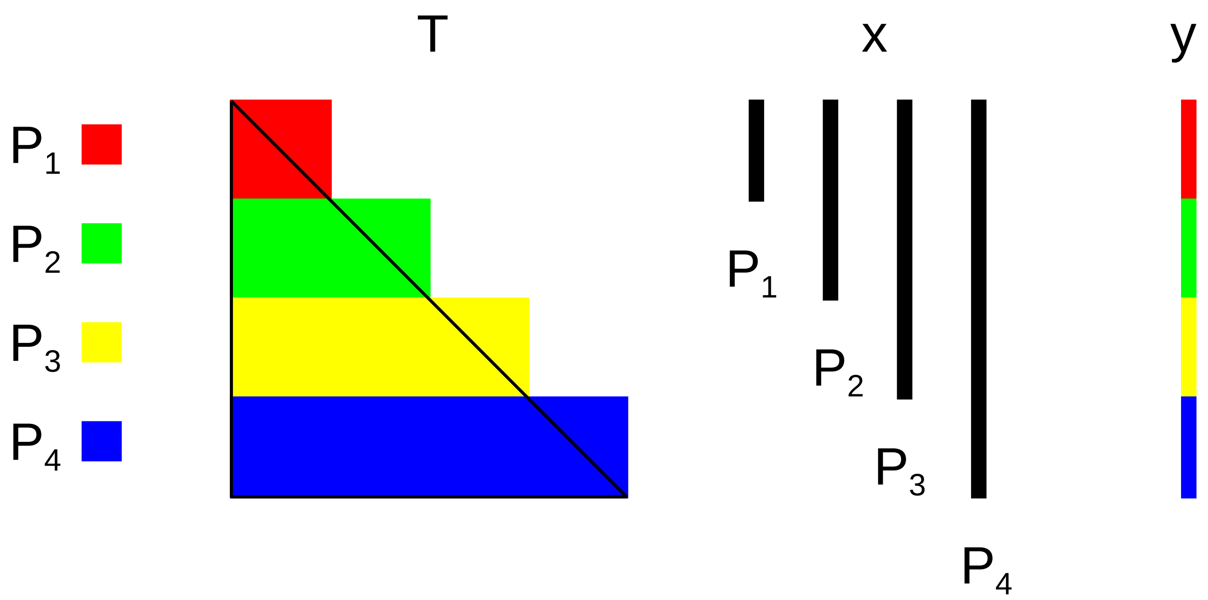
- Do we need to synchronize work?
- Load balance ?
- Idle PEs?
- Time on P CPUs?

1D cyclic distribution:



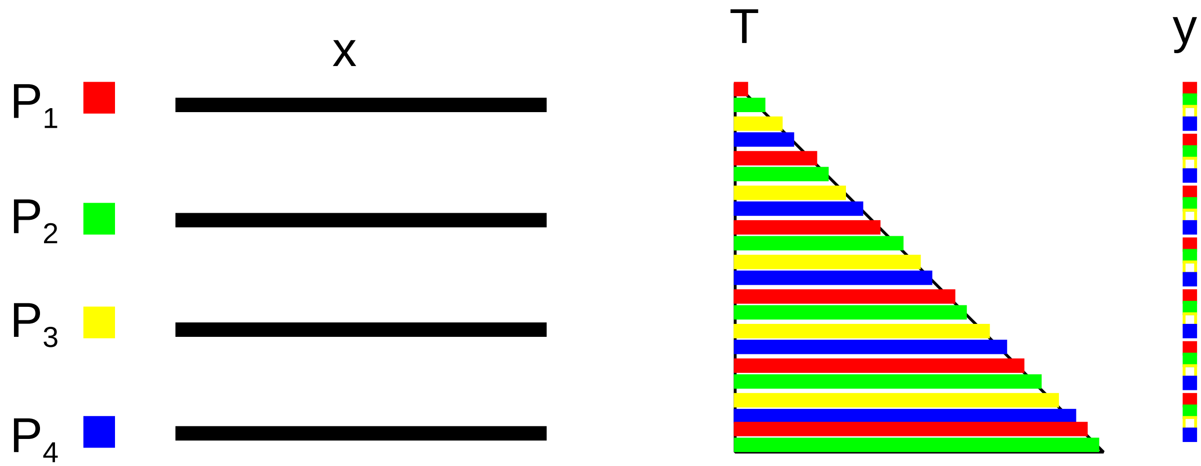
- Do we need to synchronize work?
- Load balance ?
- Idle PEs?
- Time on P CPUs?

1D block-row distribution:



- Do we need to synchronize work?
- Load balance ?
- Idle PEs?
- Time on P CPUs?

1D cyclic distribution:



- Do we need to synchronize work?
- Load balance ?
- Idle PEs?
- Time on P CPUs?

Solution of triangular systems

$$T^{(0)}x = \begin{pmatrix} t_{11} & & & & \\ t_{21} & t_{22} & & & \\ t_{31} & t_{32} & t_{33} & & \\ \vdots & & & \ddots & \\ t_{n1} & t_{n2} & t_{n3} & \cdots & t_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{pmatrix} = b^{(0)}$$

- sequential algorithm ?
- parallel algorithm ?

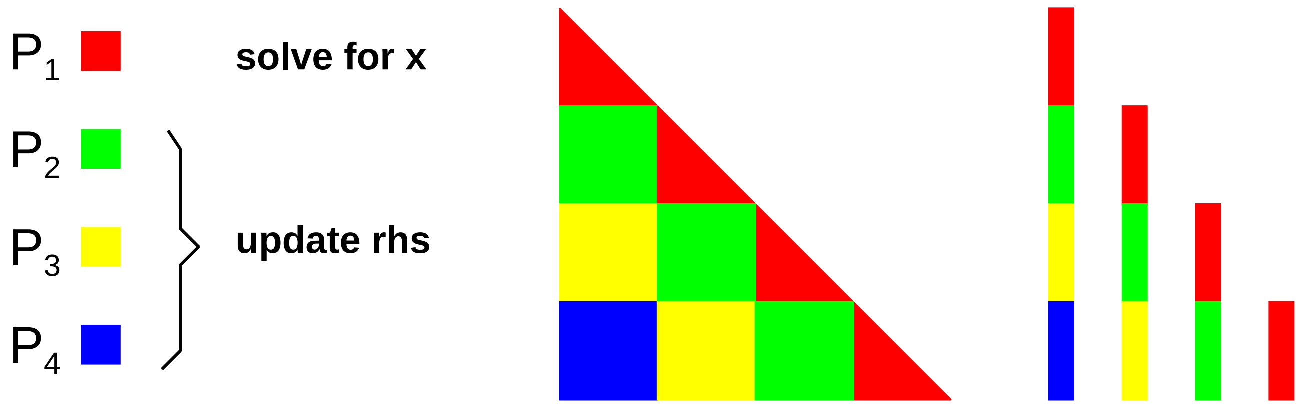
Solution of triangular systems

$$x_1 = \frac{b_1}{a_{11}},\ b^{(1)} = b^{(0)} - t_{:,1} \cdot x_1,\ \begin{pmatrix} t_{22} & & & \\ t_{32} & t_{33} & & \\ \vdots & & \ddots & \\ t_{n2} & t_{n3} & \cdots & t_{nn} \end{pmatrix} \begin{pmatrix} x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} = b^{(1)}_{2:n}$$

$$x_2 = \frac{b^{(1)}_2}{a_{22}},\ b^{(2)} = b^{(1)} - t_{:,2} \cdot x_1,\ \begin{pmatrix} t_{33} & & & \\ t_{43} & t_{44} & & \\ \vdots & & \ddots & \\ t_{n3} & t_{n4} & \cdots & t_{nn} \end{pmatrix} \begin{pmatrix} x_3 \\ x_4 \\ \vdots \\ x_n \end{pmatrix} = b^{(2)}_{3:n}$$

How do we distribute work among PEs ?

Difficulty - dependencies between tasks.



- Do we need to synchronize work?
- Load balance ?
- Idle PEs?
- Time on P CPUs?

What if the matrix is upper triangular ?

$$T^{(0)}x = \begin{pmatrix} t_{11} & t_{12} & t_{13} & \cdots & t_{1n} \\ & t_{22} & t_{23} & \cdots & t_{2n} \\ & & t_{33} & \cdots & t_{3n} \\ & & & \ddots & \vdots \\ & & & & t_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{pmatrix} = b^{(0)}$$

- Model problem: Gaussian elimination
- Introduction to Pthreads