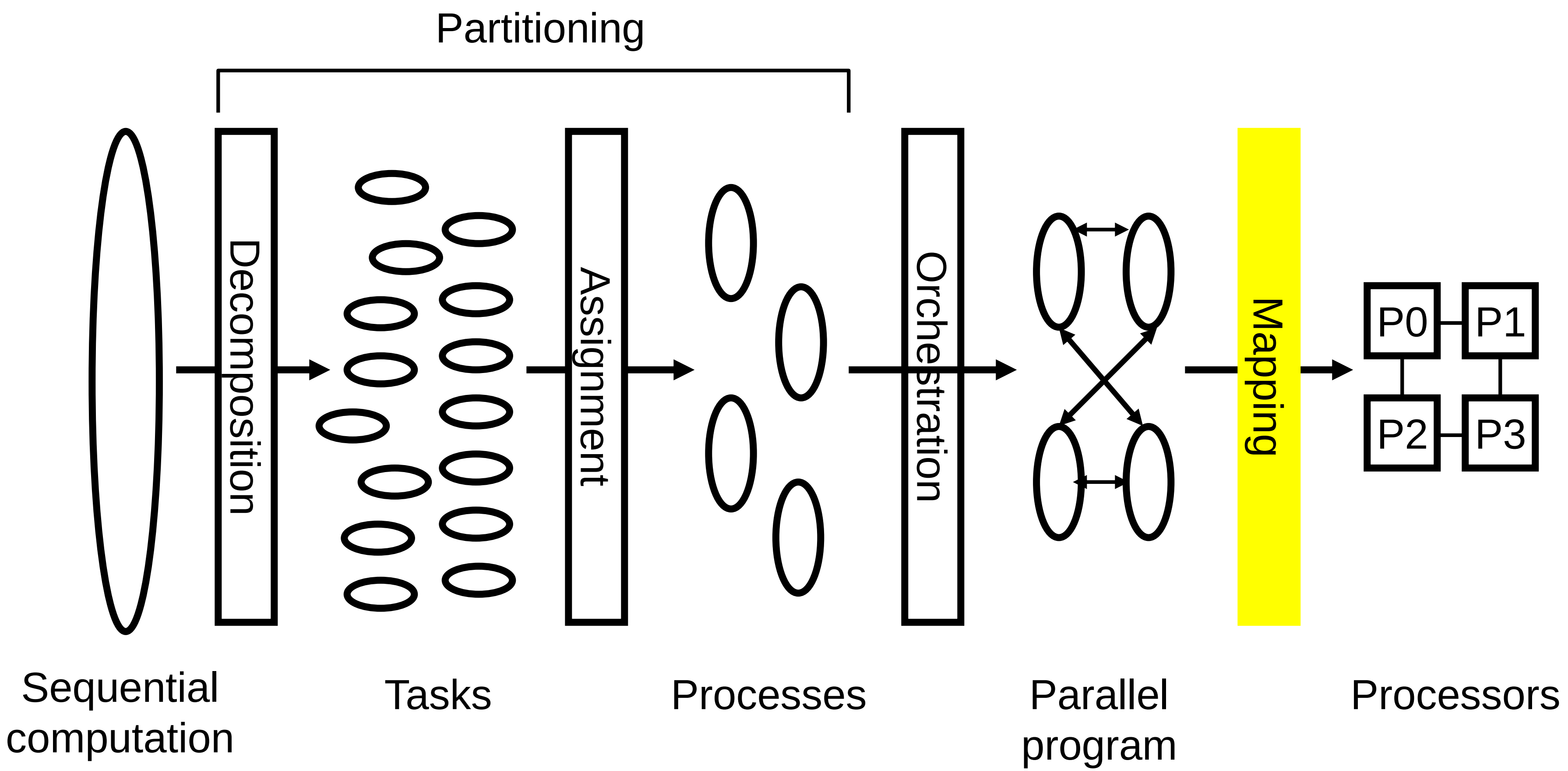
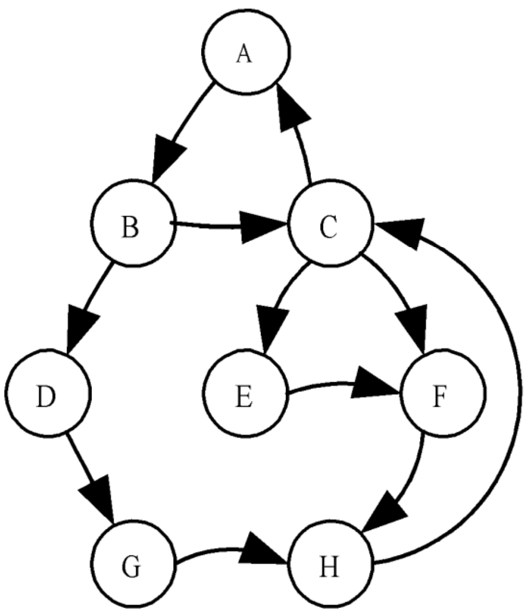
Phases in the Parallelization Process



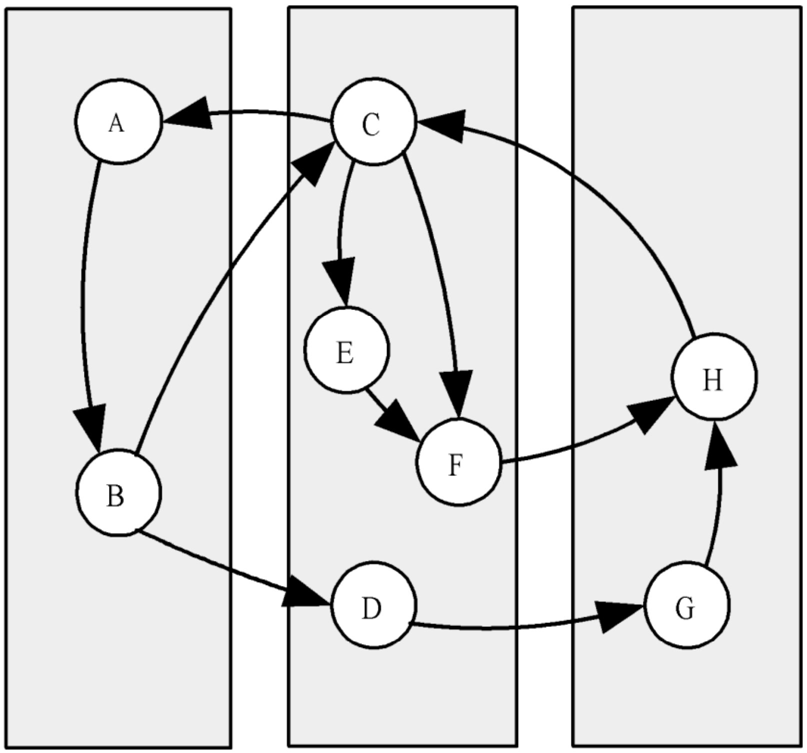
Mapping

- Mapping processes to processors
- Done by the program and/or operating system
- Shared memory system: mapping done by operating system
- Distributed memory system: mapping done by user
- Conflicting goals of mapping
  - Maximize processor utilization
  - Minimize interprocessor communication

Mapping Example



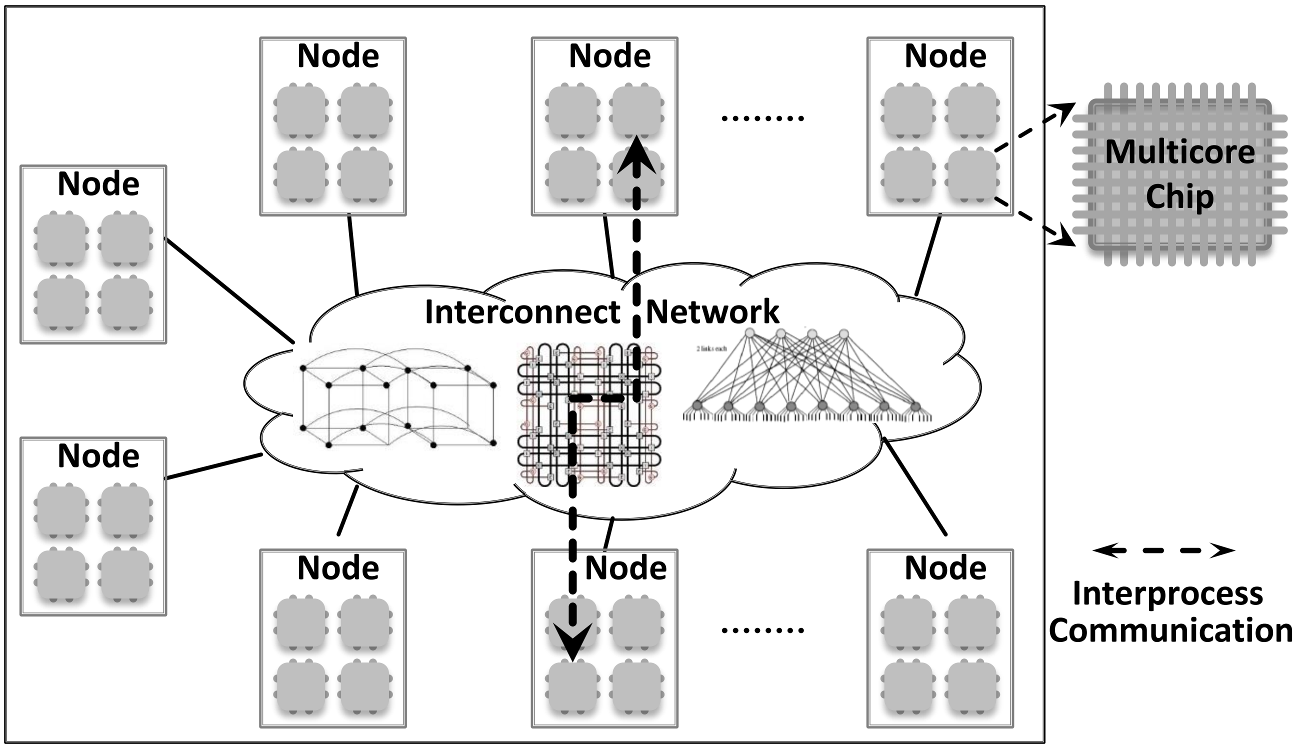
(a)



(b)

Mapping Problem

- Mapping problem to minimize execution time is **NP-complete** (definition?)
  - Hence resort to heuristics
- On modern computer systems, it is also a multilevel problem



Performance Goals

Step	Architecture-Dependent?	Major Performance Goal
Decomposition	Mostly No	➢Expose enough concurrency but not too much
Assignment	Mostly no	➢Balance workload ➢Reduce communication volume
Orchestration	Yes	➢Reduce unnecessary communication via data locality ➢Reduce communication and synchronization cost as seen by the processor ➢Reduce serialization at shared resources
Mapping	Yes	➢Put related processes on the same processor if necessary ➢Exploit locality in network topology

Parallel Algorithm Design

Case Study: Tridiagonal Solvers

Xian-He Sun
Department of Computer Science
Illinois Institute of Technology
sun@iit.edu

Outline

- Problem Description
- Parallel Algorithms
  - The Partition Method
  - The PPT Algorithm
  - The PDD Algorithm
  - The LU Pipelining Algorithm
  - The PTH Method and PPD Algorithm
- Implementations

Problem Description

- Tridiagonal linear system

$$Ax = d$$

$$A = \begin{pmatrix} b_0 & c_0 & & & \\ a_1 & b & c_1 & & \\ & \ddots & \ddots & \ddots & \\ & & a_{n-2} & b_{n-2} & c_{n-2} \\ & & & a_{n-1} & b_{n-1} \end{pmatrix} = \tilde{A} + \Delta A$$

Sequential Solver

Problem

$$a_k x_{k-1} + b_k x_k + c_k x_{k+1} = d_k$$

(k=2, … N)

Forward step

$$\beta_1 = b_1 \qquad \beta_k = b_k - a_k c_{k-1} / \beta_{k-1}$$

$$\alpha_1 = d_1 / \beta_1 \qquad \alpha_1 = (-a_k \alpha_{k-1} + d_k) / \beta_k$$

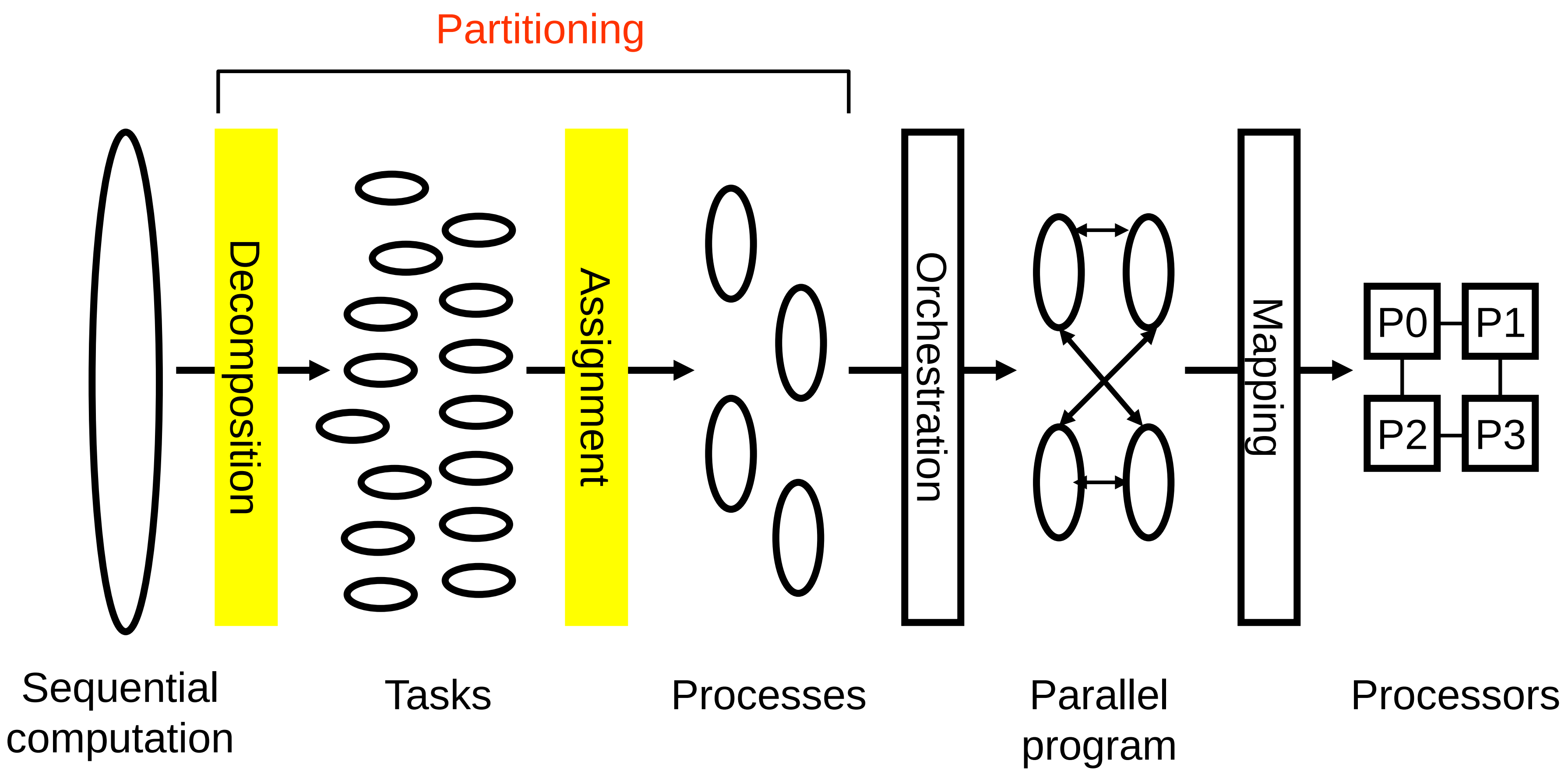
(k=2, … N)

Backward Step

$$x_n = \alpha_n \qquad x_k = (\alpha_k - x_{k-1} c_k) / \beta_k$$

(k=N-1, … 1)

Partition


Partitioning
Decomposition
Assignment
Orchestration
Mapping
P0
P1
P2
P3
Sequential computation
Tasks
Processes
Parallel program
Processors

The Matrix Modification Formula

$$x = \mathbf{A}^{-1}\mathbf{d} = \left(\tilde{\mathbf{A}} + \Delta\mathbf{A}\right)^{-1}\mathbf{d} = (\tilde{\mathbf{A}} + \mathbf{V}\mathbf{E}^{\mathrm{T}})^{-1}\mathbf{d}$$

$$x = \tilde{\mathbf{A}}^{-1}\mathbf{d} - \tilde{\mathbf{A}}^{-1}\mathbf{V}(\mathbf{I} + \mathbf{E}^{\mathrm{T}}\tilde{\mathbf{A}}^{-1}\mathbf{V})^{-1}\mathbf{E}^{\mathrm{T}}\tilde{\mathbf{A}}^{-1}\mathbf{d}$$

$$\tilde{\mathbf{A}} = \left[\begin{array}{c:c:c:c} A_0 & & & \\ \hdashline & A_1 & & \\ \hdashline & & \ddots & \\ \hdashline & & & A_{p-1} \end{array}\right] \qquad \Delta\mathbf{A} = \left[\begin{array}{c:c:c:c} & c_{m-1} & & \\ \hdashline a_m & & c_{2m-1} & \\ \hdashline & a_{2m} & \ddots & c_{m(p-1)-1} \\ \hdashline & & a_{m(p-1)} & \end{array}\right]$$

The Partition of Tridiagonal Systems

$$A = \tilde{A} + VE^T$$

$$\Delta A = [a_m e_m, c_{m-1} e_{m-1}, a_{2m} e_{2m}, c_{2m-1} e_{2m-1}, \ldots, c_{(p-1)m-1} e_{(p-1)m-1}] \cdot \begin{bmatrix} e_{m-1}^T \\ e_m^T \\ . \\ . \\ . \\ e_{(p-1)m-1}^T \\ e_{(p-1)m}^T \end{bmatrix} = VE^T$$

e_i are column vector with ith element being one and all the other entries being zero.

The Solving process

1. Solve the subsystems in parallel
2. Solve the reduced system
3. Modification

$$\tilde{\mathbf{A}}^{-1}\mathbf{A}\mathbf{x} = \tilde{\mathbf{A}}^{-1}\mathbf{d}$$

$$\begin{bmatrix} A_0^{-1} & & & \\ & A_1^{-1} & & \\ & & \ddots & \\ & & & A_{p-1}^{-1} \end{bmatrix} \begin{bmatrix} A_0 & * & & & \\ * & A_1 & * & & \\ & * & \ddots & & \\ & & & \ddots & * \\ & & & * & A_{p-1} \end{bmatrix} \begin{pmatrix} x_0 \\ x_1 \\ \cdot \\ \cdot \\ \cdot \\ x_{n-1} \end{pmatrix} = \begin{bmatrix} A_0^{-1} & & & \\ & A_1^{-1} & & \\ & & \ddots & \\ & & & A_{p-1}^{-1} \end{bmatrix} \begin{pmatrix} d_0 \\ d_1 \\ \cdot \\ \cdot \\ \cdot \\ d_{n-1} \end{pmatrix}$$

The Solving Procedure

$$\tilde{A}x = d$$

$$\tilde{A}Y = V$$

$$h = E^T \tilde{x}$$

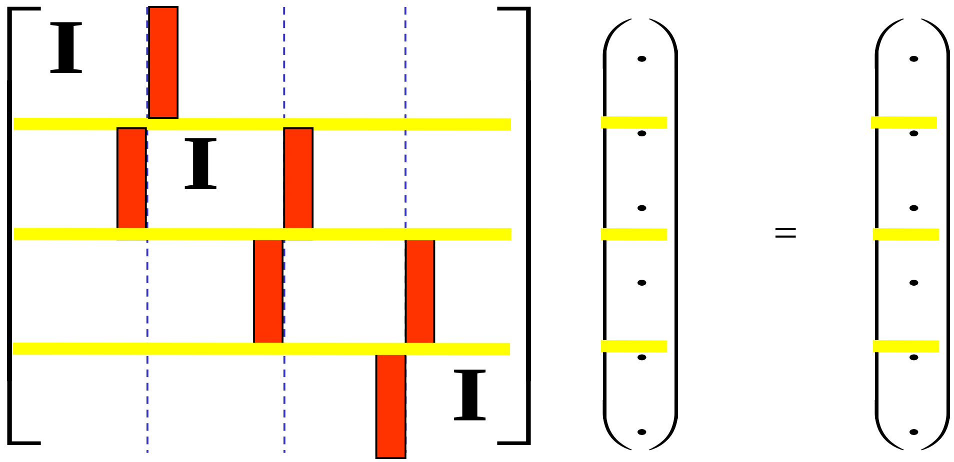
$$Z = I + E^T Y$$

$$Zy = h$$

$$\Delta x = Yy$$

$$x = \tilde{x} - \Delta x$$

The Reduced System ($Zy=h$)



Needs global communication

The Parallel Partition LU (PPT) Algorithm

Step 1. Allocate $A_i, d^{(i)}$ and elements $a_{im}, c_{(i+1)m-1}$ to the ith node, where $0 \le i \le p-1$.

Step 2. Use the LU decomposition method to solve

$$A_i[\tilde{x}^{(i)}, v^{(i)}, w^{(i)}] = [d^{(i)}, a_{im}e_0, c_{(i+1)m-1}e_{m-1}]$$

Step 3. Send $\tilde{x}_0^{(i)}, \tilde{x}_{m-1}^{(i)}, v_0^{(i)}, v_{m-1}^{(i)}, w_0^{(i)}, w_{m-1}^{(i)}$ from the ith node to the other nodes $0 \le i \le p-1$.

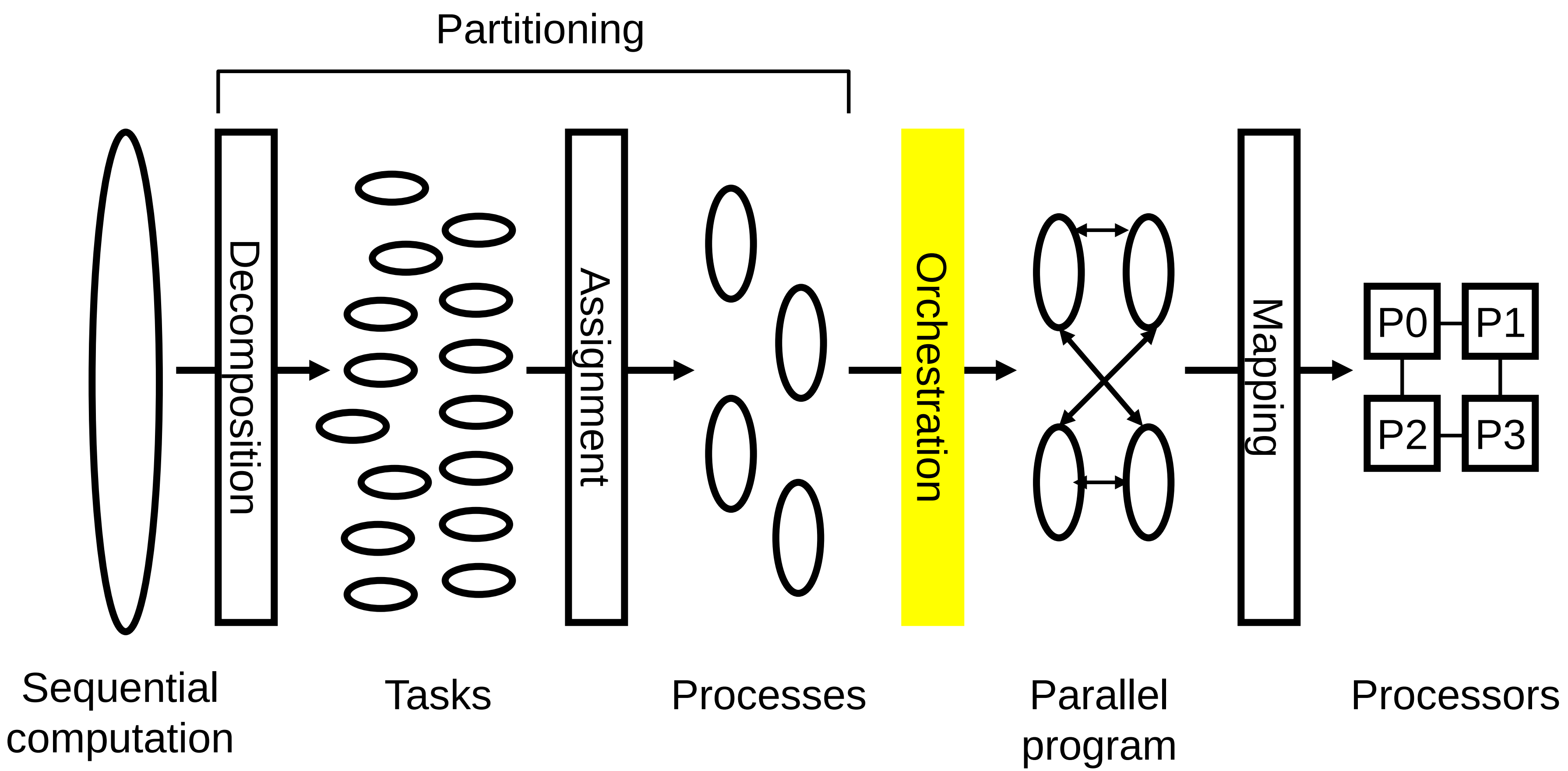
Step 4. Use the LU method to solve Zy = h on all nodes

Step 5. Compute in parallel on p processors

$$\Delta x^{(i)} = [v^{(i)}, w^{(i)}]\begin{bmatrix} y_{2i-1} \\ y_{2i} \end{bmatrix}$$

$$x^{(i)} = \tilde{x}^{(i)} - \Delta x^{(i)}$$

Orchestration

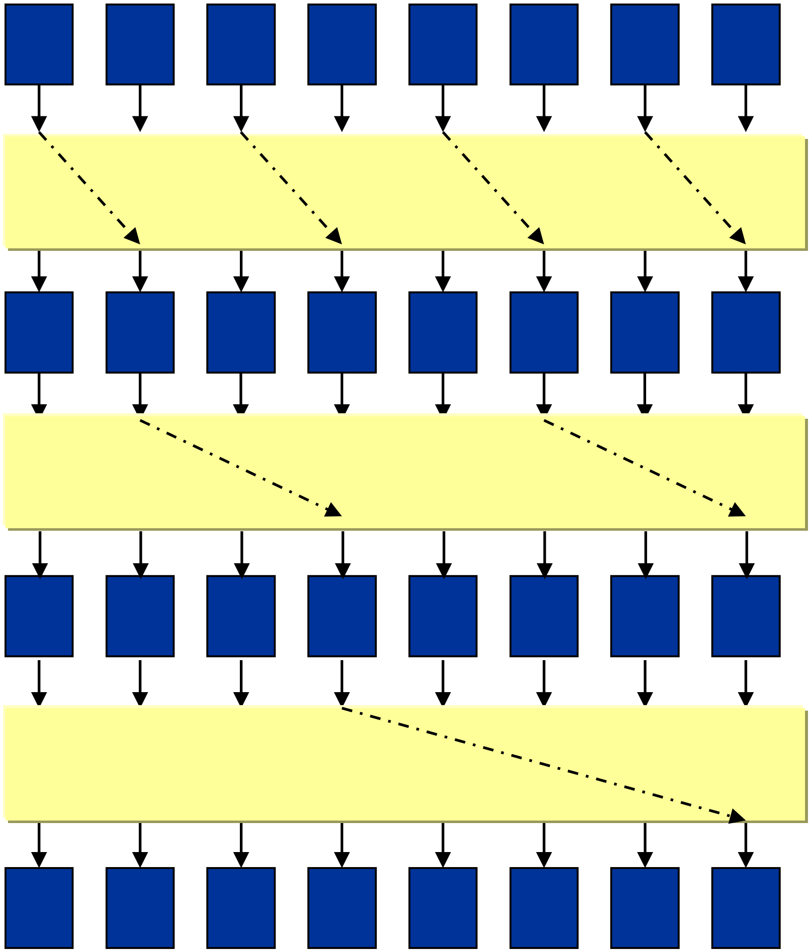


Orchestration

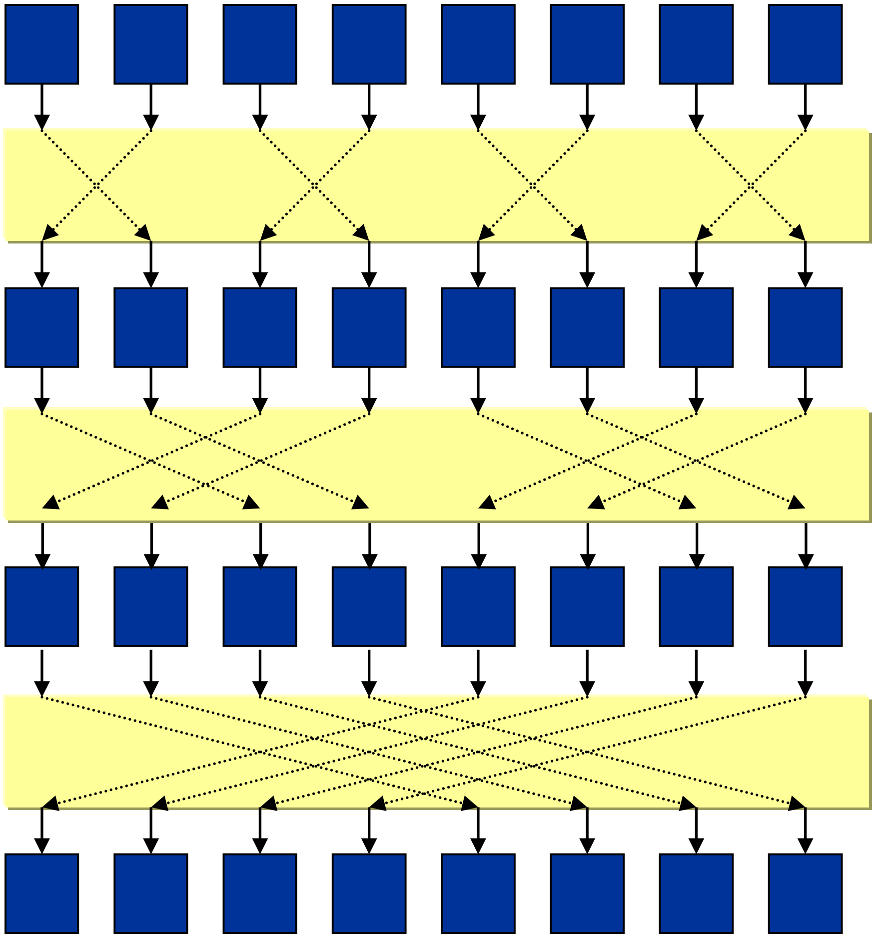
Orchestration is implied in the PPT algorithm

- Intuitively, the reduced system should be solved on one node
  - A tree-reduction communication to get the data
  - Solve
  - A reversed tree-reduction communication to set the results
  - 2 log(p) communication, one solving
- In PPT algorithm (step 3)
  - One total data exchange
  - All nodes solve the reduced system concurrently
  - 1 log(p) communication, one solving

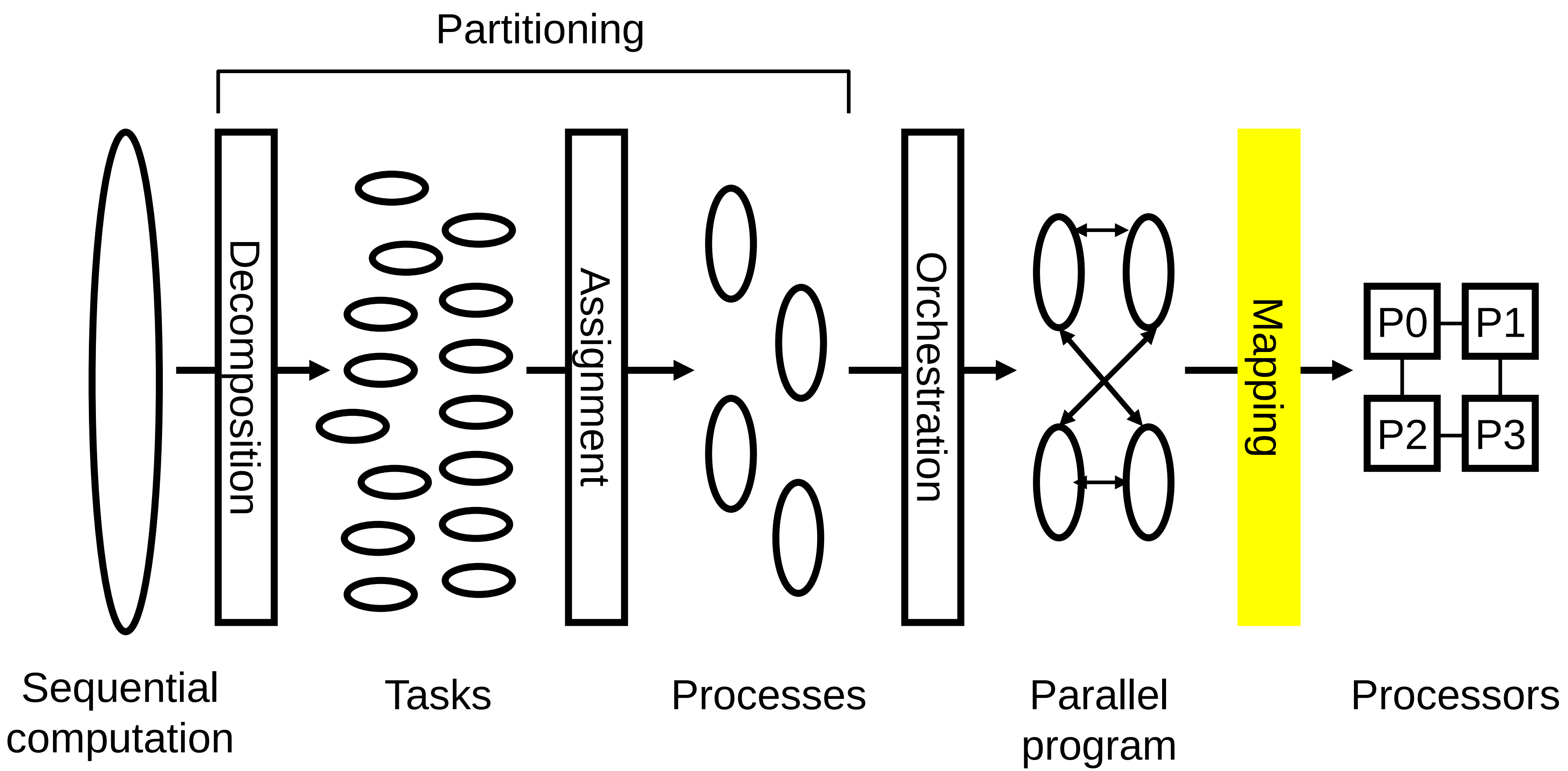
Tree Reduction (Data gathering/scattering)



All-to-All Total Data Exchange



Mapping



Partitioning
Decomposition
Assignment
Orchestration
Mapping
P0
P1
P2
P3
Sequential computation
Tasks
Processes
Parallel program
Processors

Mapping

- Try to reduce the communication
  - Reduce time
  - Reduce message size
  - Reduce cost: distance, contention, congestion, etc
- In total data exchange
  - Try to make every comm. a direct comm.
  - Can be achieved in hypercube architecture

The PPT Algorithm

- Advantage
  - Perfect parallel
- Disadvantage
  - Increased computation (vs. sequential alg.)
  - Global communication
  - Sequential bottleneck

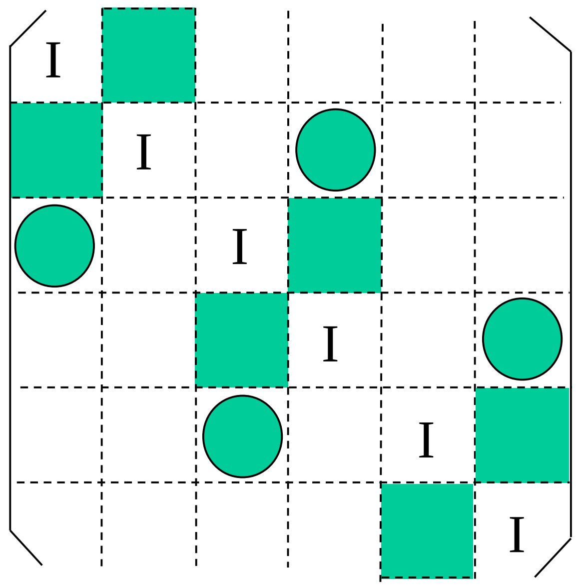
Problem Description

- Parallel codes have been developed during last decade
- The performances of many codes suffer in a scalable computing environment
- Need to identify and overcome the scalability bottlenecks

Diagonal Dominant Systems

$$
\begin{pmatrix}
1 & \frac{2}{9} & & & & \\
\frac{2}{7} & 1 & \frac{1}{5} & & & \\
. & . & . & & & \\
 & . & . & . & & \\
 & & & \frac{1}{6} & 1 & \frac{3}{7} \\
 & & & & \frac{3}{8} & 1
\end{pmatrix}
\begin{pmatrix}
x_0 \\ x_1 \\ . \\ . \\ . \\ x_{n-1}
\end{pmatrix}
=
\begin{pmatrix}
d_0 \\ . \\ . \\ . \\ . \\ d_{n-1}
\end{pmatrix}
$$

- The Reduced System of Diagonal Dominant Systems

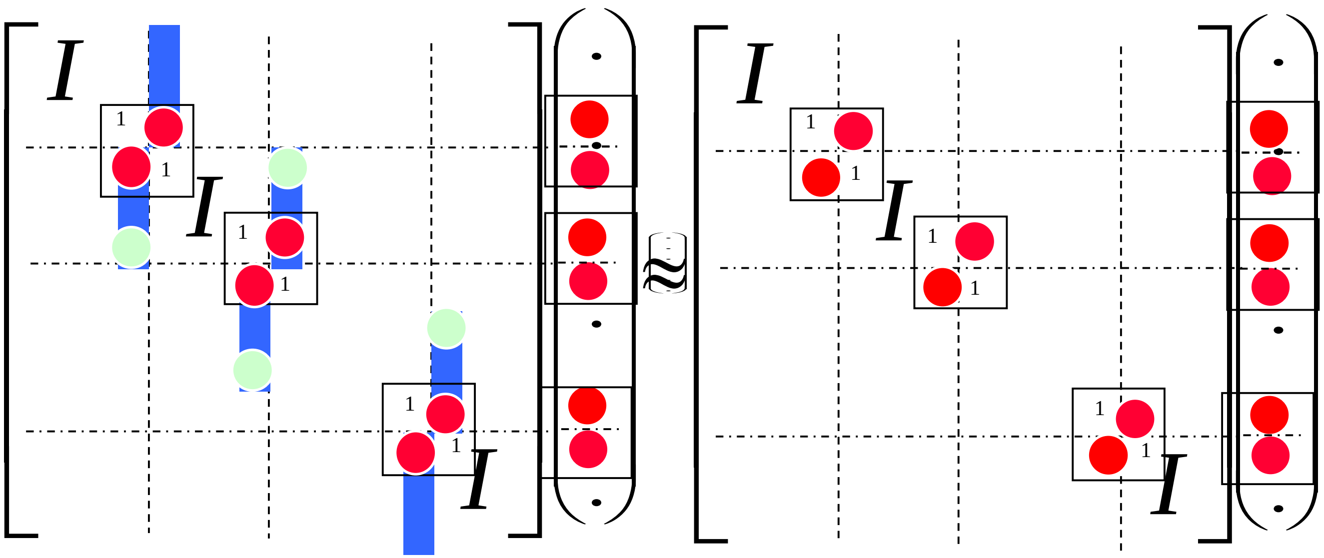


- Decay Bound for Inverses of Band Matrices

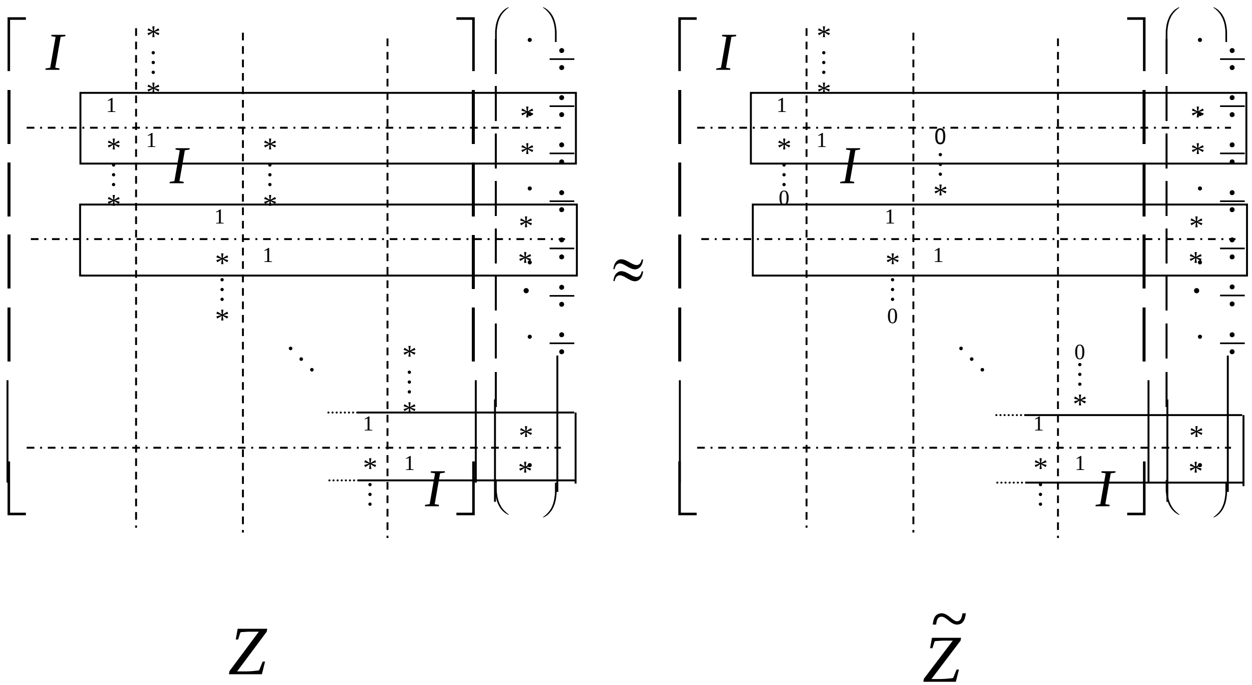
$$\left|A^{-1}(i,j)\right| \le Cq^{|i-j|/m}, \qquad 0 < q < 1$$

$$q = q(r) = \frac{\sqrt{r}-1}{\sqrt{r}+1} \qquad r = \frac{\lambda_{\max}}{\lambda_{\min}}$$

The Reduced communication



Generally needs global communication, Decay for diagonal dominant systems


I
I
I
Z
≈
Z̃

The Parallel Diagonal Dominant (PDD) Algorithm

Step 1. Allocate $A_i, d^{(i)}$ and elements $a_{im}, c_{(i+1)m-1}$ to the ith node, where $0 \le i \le p-1$.

Step 2. Use the LU decomposition method to solve

$$A_i[\tilde{x}^{(i)}, v^{(i)}, w^{(i)}] = [d^{(i)}, a_{im}e_0, c_{(i+1)m-1}e_{m-1}]$$

Step 3. Send $\tilde{x}_0^{(i)}, v_0^{(i)}$ to the $(i-1)th$ node.

Step 4. Solve

$$\begin{pmatrix} w_{m-1}^{(i)} & 1 \\ 1 & v_0^{(i+1)} \end{pmatrix} \begin{pmatrix} y_{2i} \\ y_{2i+1} \end{pmatrix} = \begin{pmatrix} \tilde{x}_{m-1}^{(i)} \\ \tilde{x}_0^{(i+1)} \end{pmatrix}$$

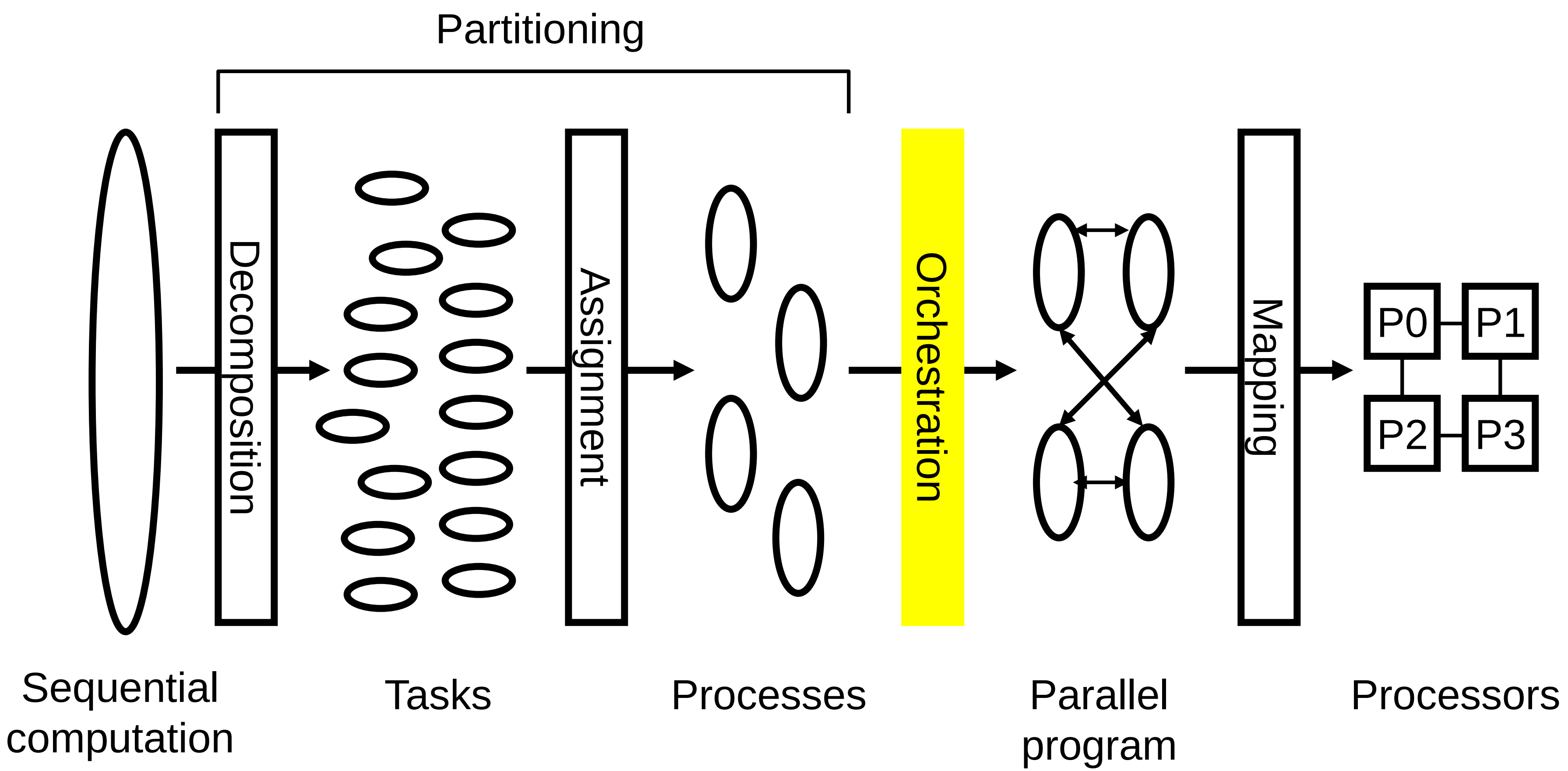
in parallel and send y_{2i+1} to $(i+1)th$ node

Step 5. Compute in parallel on p processors

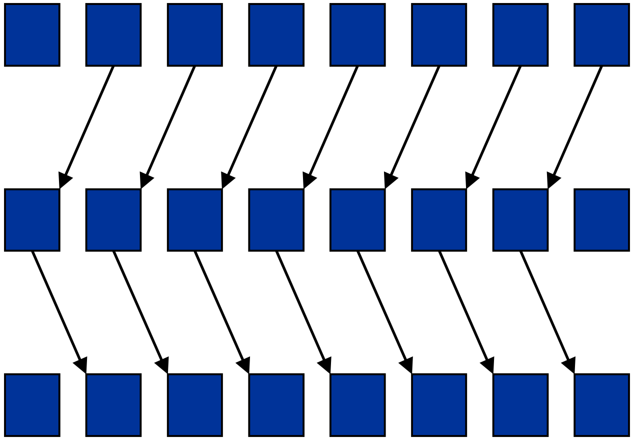
$$\Delta x^{(i)} = [v^{(i)}, w^{(i)}] \begin{bmatrix} y_{2i-1} \\ y_{2i} \end{bmatrix}$$

$$x^{(i)} = \tilde{x}^{(i)} - \Delta x^{(i)}$$

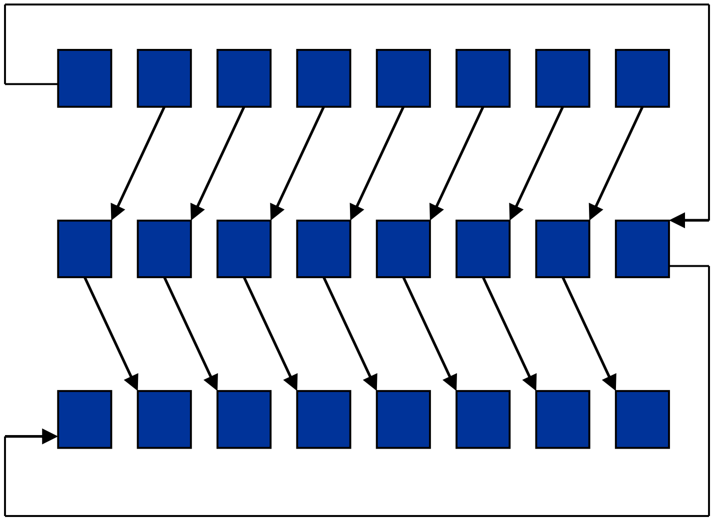
Orchestration



Computing/Communication of PDD



Non-periodic



Periodic

Orchestration

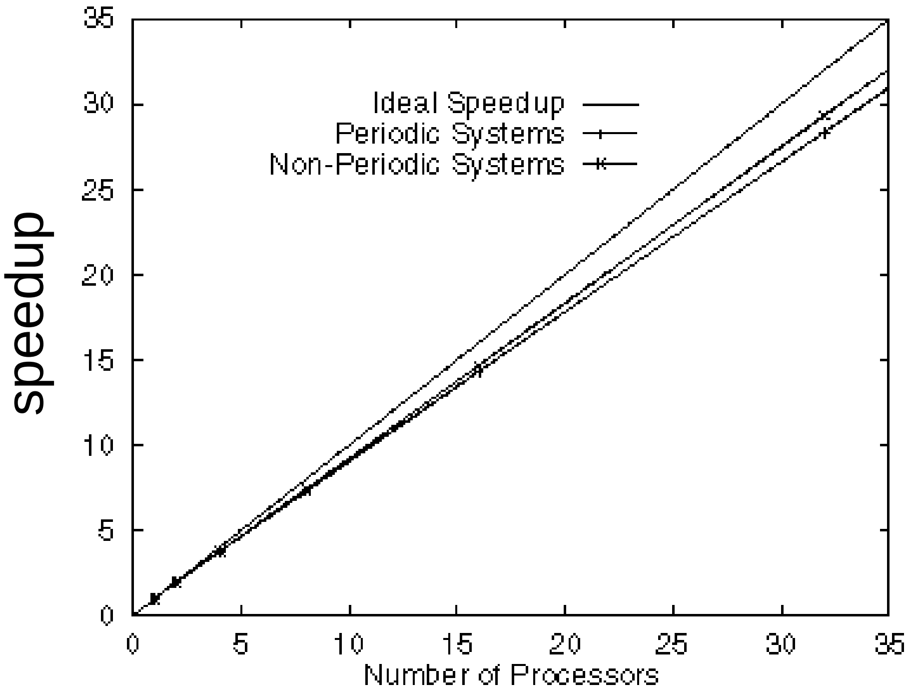
- Orchestration is implied in the algorithm design
- Only two one-to-one neighboring communication

Mapping

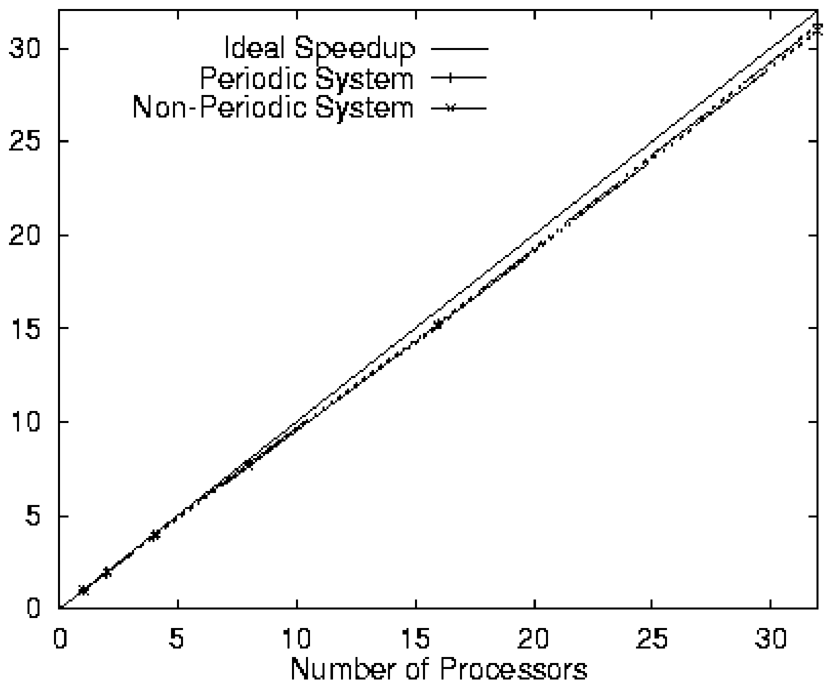
- Communication has reduced
  - Take the special mathematical property
  - Formal analysis can be performed based on the mathematical partition formula
- Two neighboring communication
  - Can be achieved on array communication network

The PDD Algorithm

- Advantage
  - Perfect parallel
  - Constant, minimum communication
- Disadvantage
  - Increased computation (vs. sequential alg.)
  - Applicability
    - Diagonal dominant
    - Subsystems are reasonably large



Scaled Speedup of the PDD Algorithm on Paragon. *1024 System of order 1600, periodic & non-periodic*



Scaled Speedup of the Reduced PDD Algorithm on SP2. *1024 System of Order 1600, periodic & non-periodic*

Problem Description

- For tridiagonal systems we may need new algorithms

Problem Description

- Tridiagonal linear systems

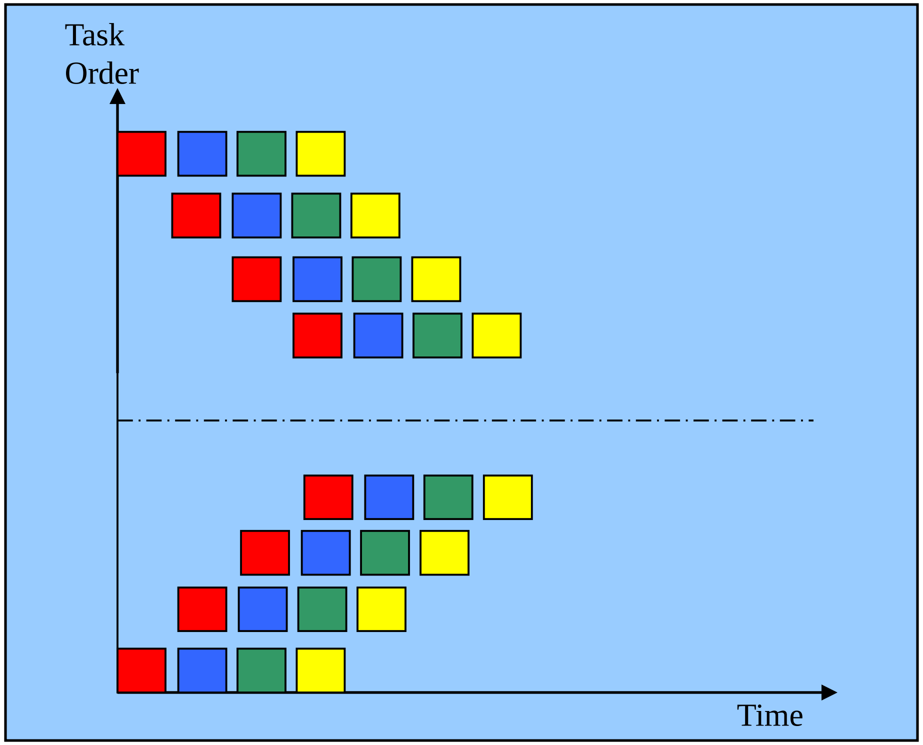
$$AX = D$$

$$A = \begin{pmatrix} b_0 & c_0 & & & \\ a_1 & b_1 & c_1 & & \\ & \ddots & \ddots & \ddots & \\ & & a_{n-2} & b_{n-2} & c_{n-2} \\ & & & a_{n-1} & b_{n-1} \end{pmatrix}$$

$$X = \begin{pmatrix} x_{00} & x_{01} & . & . & x_{0,n-1} \\ x_{10} & x_{11} & x_{12} & . & . \\ . & \ddots & \ddots & \ddots & . \\ . & . & x_{n-2,n-3} & x_{n-2,n-2} & x_{n-2,n-1} \\ x_{n-1,0} & . & . & x_{n-1,n-2} & x_{n-1,n-1} \end{pmatrix}$$

$$D = \begin{pmatrix} d_{00} & d_{01} & . & . & d_{0,n-1} \\ d_{10} & d_{11} & d_{12} & . & . \\ . & \ddots & \ddots & \ddots & . \\ . & . & d_{n-2,n-3} & d_{n-2,n-2} & d_{n-2,n-1} \\ d_{n-1,0} & . & . & d_{n-1,n-2} & d_{n-1,n-1} \end{pmatrix}$$

The Pipelined Method

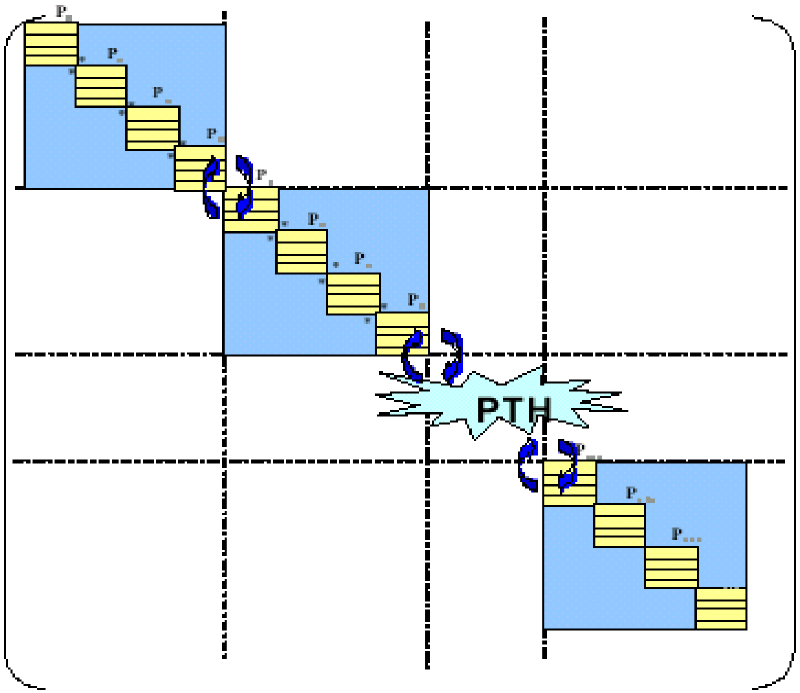


- Exploit temporal parallelism of multiple systems
- Passing the results form solving a subset to the next before continuing
- Communication is high, 3p
- Pipelining delay, p
- Optimal computing

The Parallel Two-Level Hybrid Method

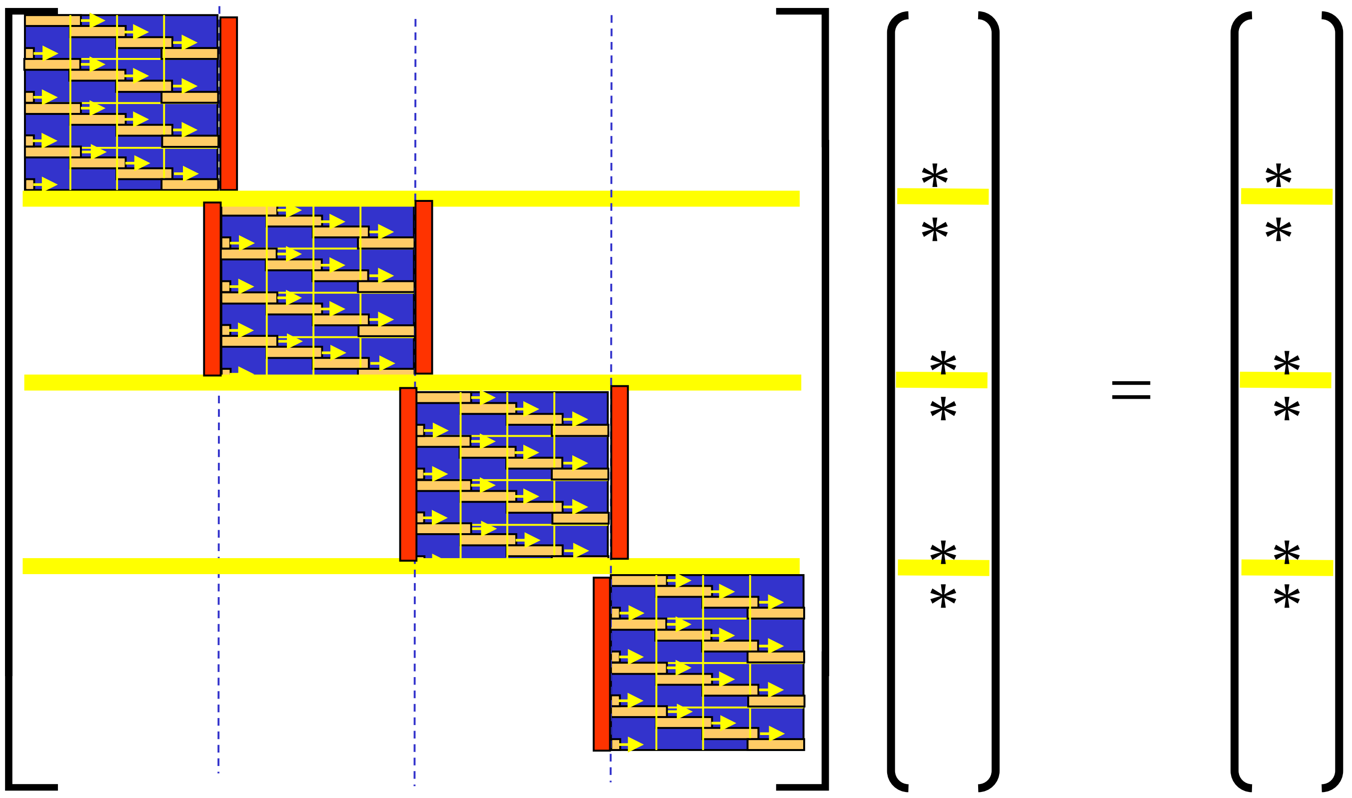
- PDD is scalable but has limited applicability
- The pipelined method is mathematically efficient but not scalable
- Combine these two algorithms, outer PDD, inner pipelining
- Can combine with other algorithms too

The Parallel Two-Level Hybrid Method



- Use an accurate parallel tridiagonal solver to solve the *m* super-subsystems concurrently, each with k processors
- Modify PDD algorithm and consider communications only between the *m* super-subsystems.

The Partition Pipeline diagonal Dominant (PPD) algorithm



•Evaluation of Algorithms

System	Algorithm	Computation	Communication
Multiple systems	Best Sequential	$8n-7$	0
	Pipelining	$\frac{(n_1-1+p)(8n-7)}{p}$	$3(n_1-1+p)(\alpha+4\beta)$
	PDD	$(17\frac{n}{p}-14)*n_1$	$(2\alpha+12*n_1*\beta)$
	PPD	$(n_1-1+k)\frac{13n}{p}+4n_1(\frac{n}{p}+1)$	$3(2\alpha+12\beta)+[2+\log(k)](\alpha+12n_1\beta)$

Practical Motivation

- NLOM (NRL Layered Ocean Model) is a well-used naval parallel ocean simulation code (see http://www7320.nrlssc.navy.mil/global_nlom/index.html).
- Fine tuned with the best algorithms available at the time
- Efficiency goes down when the number of processors increases.
- Poisson solver is the identified scalability bottleneck

Project Objectives

- Incorporate the best scalable solver, the PDD algorithm, into NLOM
- Increase the scalability of NLOM
- Accumulate experience for a general toolkits solution for other naval simulation codes

Experimental Testing

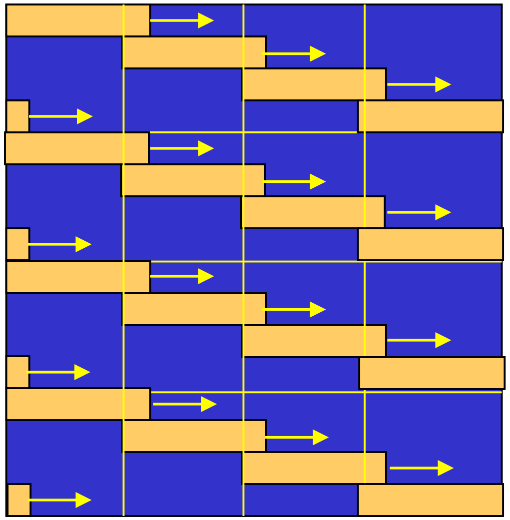
- Fast Poisson solvers (FACR) (Hockney, 1965)
- One of the most successful rapid elliptic solvers

$$f_q \xrightarrow{FFT} \bar{f}_q^{\,k} \xrightarrow[\text{Tridiagonal System}]{\text{Diagonal Dominant}} \overline{\varphi}_q^{\,k} \xrightarrow{FFT} \varphi_q$$

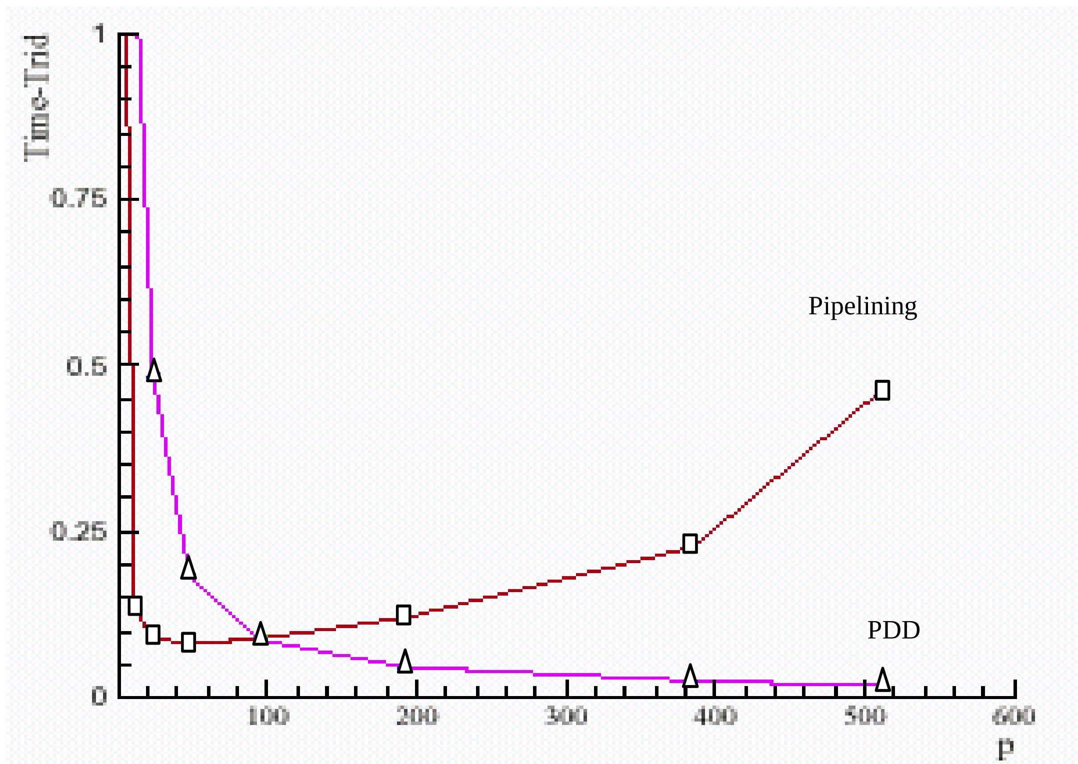
- Large number of systems, each node has a piece of each system
- NLOM implementation, highly optimized pipelining
- Burn At Both Ends (BABE), trade computation with comm. (p, 2p)

NLOM Implementation

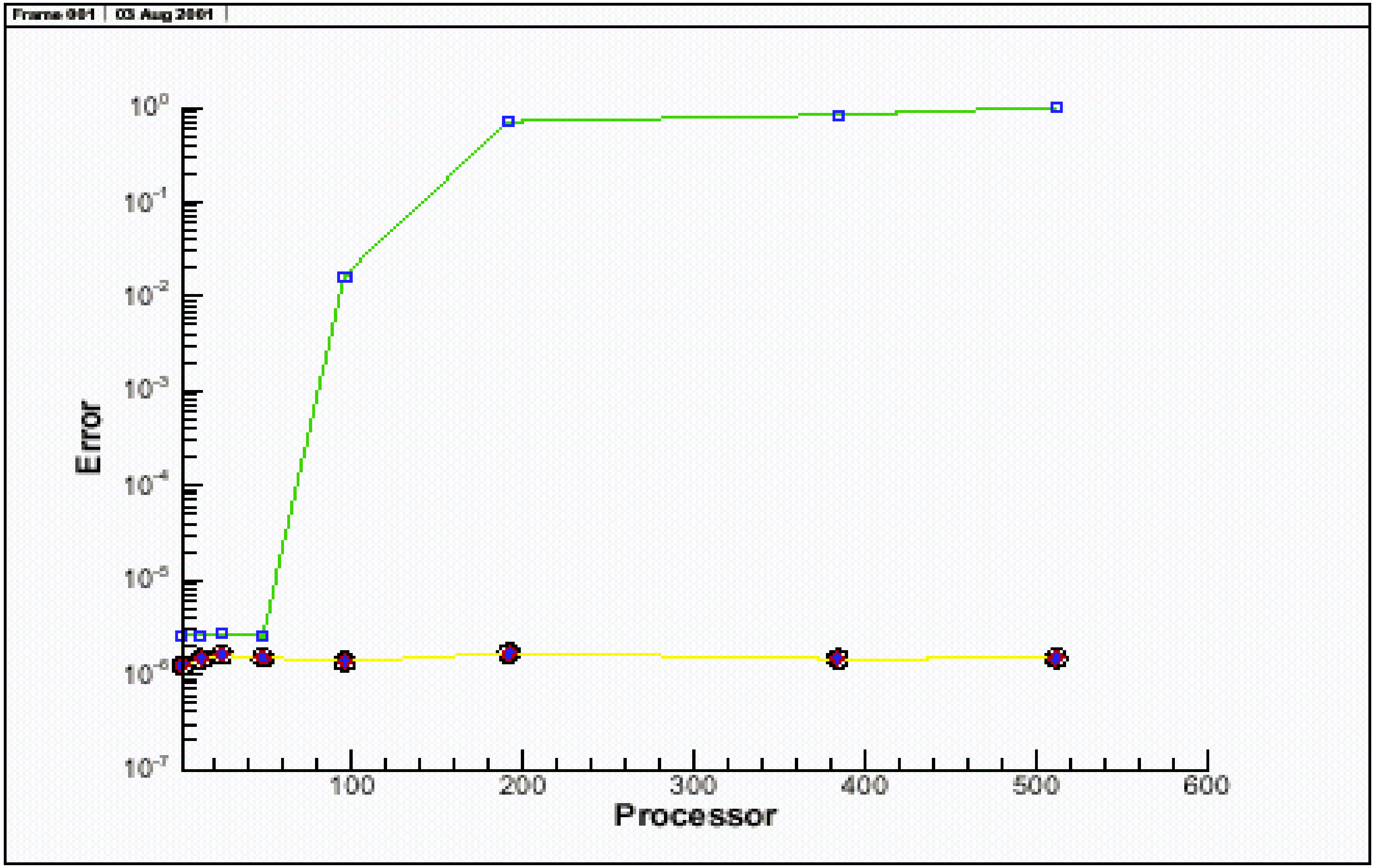
- NLOM has a special data structure and partition
  - Large number of systems, each node has a piece of each system
- Pipelined method, highly optimized
- Burn At Both Ends (BABE), pipelining at both sides, trade computation with comm. (p, 2p)



Tridiagonal solver runtime: Pipelining (square) and PDD (delta)



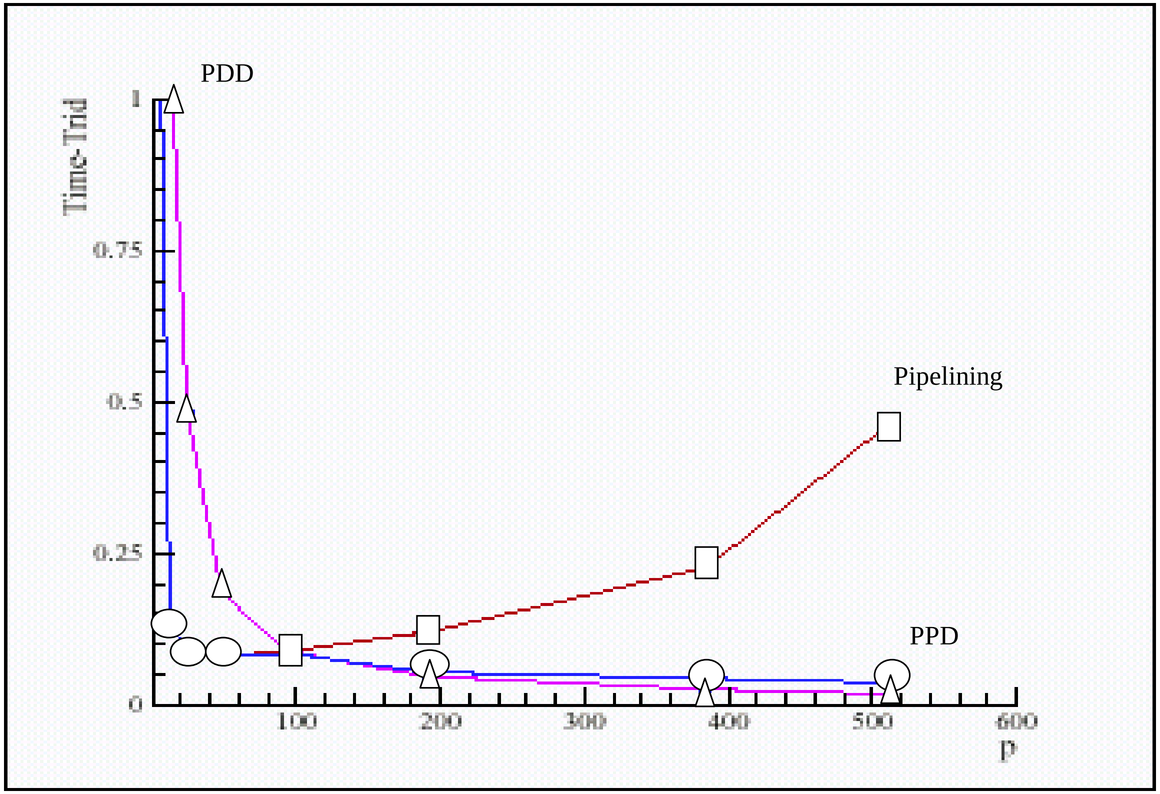
Accuracy: Circle - BABE Square - PDD Diamond - PPD



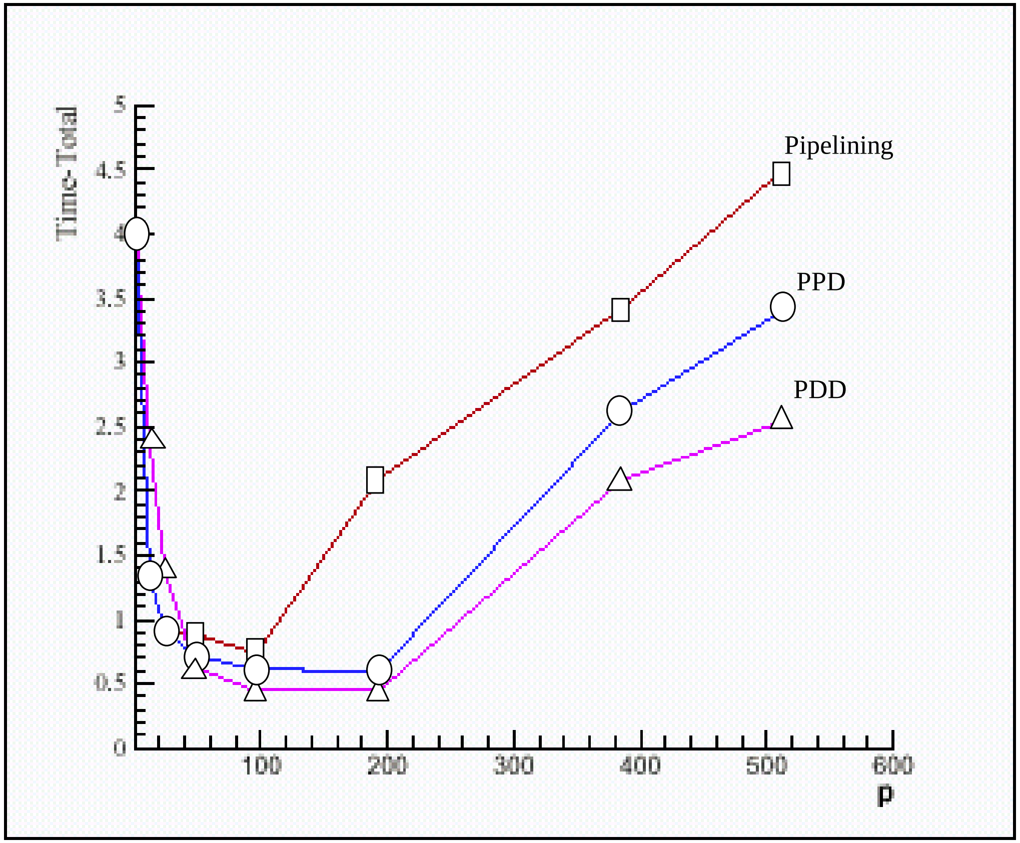
NLOM Application

- Pipelined method is not scalable
- PDD is scalable but loses accuracy, due to the subsystems are very small
- Need the two-level combined method

Trid. Solver Time: Pipelining (square), PDD (delta), PPD (circle)



Total runtime: Pipelining (square), PDD (delta), PPD (circle)



Parallel Two-Level Hybrid (PTH) Method

- Use an accurate parallel tridiagonal solver to solve the *m* super-subsystems concurrently, each with *k* processors, where $p = l \cdot k$, and solving three unknowns as given in the *Step 2* of PDD algorithm.

- Modify the solutions of *Step* 1 with *Steps* 3-5 of PDD algorithm, or of PPT algorithm if PPT is chosen as the outer solver.

The PTH method and related algorithms

Abbreviation	Full Name	Explanation
PPT	Parallel ParTition LU Algorithm	A parallel solver based on rank-one modification
PDD	Parallel Diagonal Dominant Alg	A variant of PPT for diagonal dominant system
PTH	Parallel Two-level Hybrid Method	A novel two-level approach
PPD	Partition Pipelined diagonal Dominant Algorithm	A PTH uses PDD and pipelining as outer/inner solver

Perform Evaluation

- Evaluation of Algorithms

Comparison of computation and communication (non periodic)			
System	Algorithm	Computation	Communication
Single system	Best Sequential	$8n-7$	0
	PPT	$17\frac{n}{p}+16p-23$	$(2\alpha+8p\beta)(\sqrt{p}-1)$
	PDD	$17\frac{n}{p}-4$	$2\alpha+12\beta$
	Reduced PDD	$11\frac{n}{p}+6j-4$	$2\alpha+12\beta$
Multiple system	Best Sequential	$(5n-3).n1$	0
	PPT	$(9\frac{n}{p}+10p-11).n1$	$(2\alpha+8p..n1.\beta)(\beta-1)$
	PDD	$(9\frac{n}{p}+1).n1$	$(2\alpha+8n1.\beta)$
	Reduced PDD	$(5\frac{n}{p}+4j+1).n1$	$(2\alpha+8n1.\beta)$

Algorithm Analysis:

1. LU-Pipelining

$$(n_1 - 1 + p)[(8n-7)\frac{\tau_{comp}}{p} + 3(\alpha + 4\beta)]$$

2. The PDD Algorithm

$$(17\frac{n}{p} - 14) * n_1 \tau_{comp} + (2\alpha + 12 * n_1 * \beta)$$

3. The PPD Algorithm

$$(n_1 - 1 + p_1)[\frac{13n}{p}\tau_{comp} + 3(2\alpha + 12\beta)] +$$

$$4n_1(\frac{n}{p} + 1)\tau_{comp} + [2 + \log(p_1)](\alpha + 12n_1\beta)$$

Where

n - the order of each system

n_1 - the number of systems

p - the number of processors

p_1 - the number of processors used for LU-pipelining

τ_{comp} - the computing speed

α - the communication start time

β - the transmission time

Parameters on IBM Blue Horizon at SDSC

$$\tau_{comp} = 0.01696\,\mu s$$

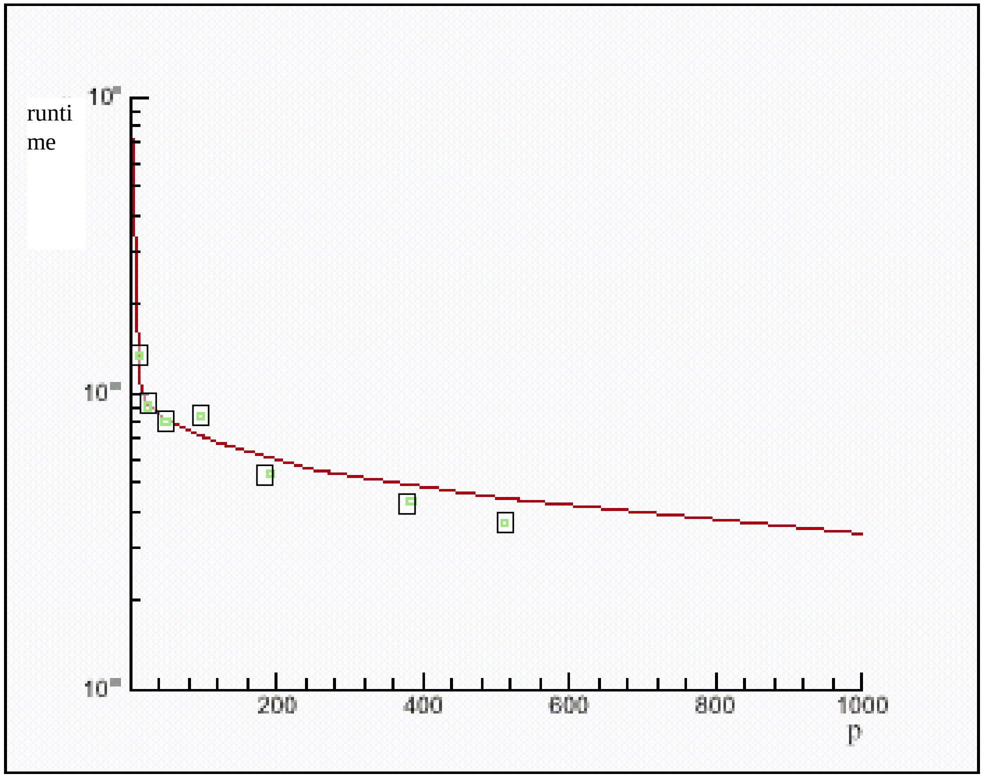
$$\alpha = 31.5\mu s$$

$$\beta = 9.52 \times 10^{-3}\,\mu s / byte$$

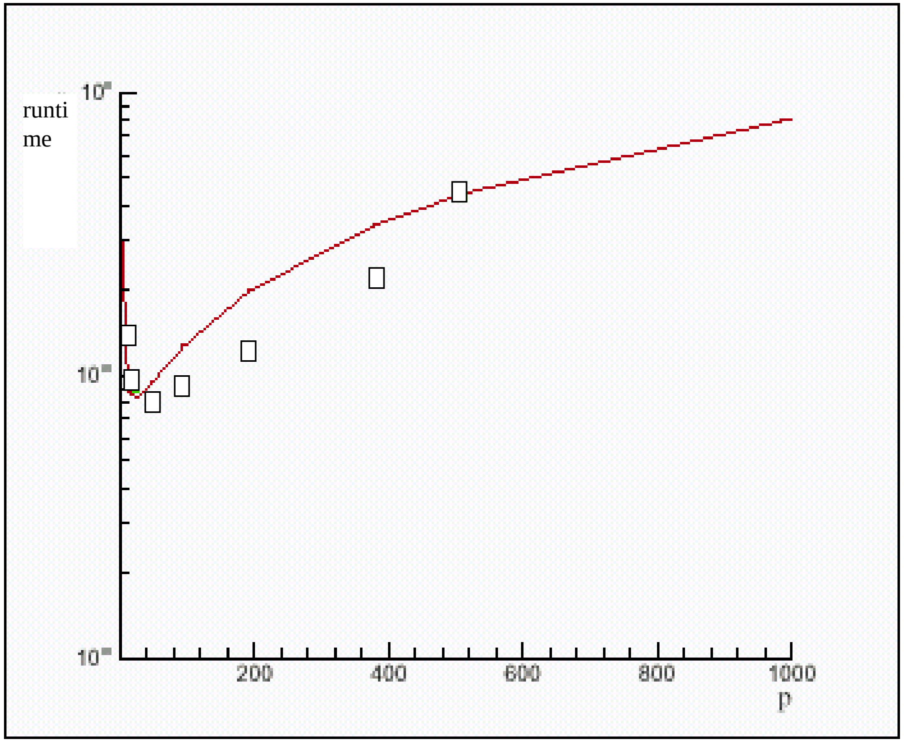
Computation and Comm. Count (multiple right sides)

Algorithm	Computation	Communication
Pipelined Algorithm	$(m-1+p)\dfrac{8n-7}{p}m_1$	$(m-1+p)(3\alpha+12m_1\beta)$
PPT non-pivoting	$\left(17\dfrac{n}{p}+16p-45\right)n_1$	$((\log(p))\alpha+16(p-1)n_1\beta)$
PDD	$\left(17\dfrac{n}{p}-14\right)n_1$	$(2\alpha+12n_1\beta)$
PPD PDD/Pipeline	$(m-1+k)\dfrac{13n}{p}m_1+\left(\dfrac{4n}{p}+7\right)n_1$	$(m-1+k)(3\alpha+24m_1\beta)+$ $(2+\log(k))\alpha+(8\log(k)+12)n_1\beta$
PPT/ Pipeline	$(m-1+k)\dfrac{13n}{p}m_1+\left(\dfrac{4n}{p}+16\dfrac{p}{k}-23\right)n_1$	$(m-1+k)(3\alpha+24m_1\beta)+$ $(\log(p))\alpha+\left[16\left(\dfrac{p}{k}-1\right)+8\log(k)\right]n_1\beta$
PDD/PPT	$\left(30\dfrac{n}{p}+21k-41\right)n_1$	$(2+2\log(k))\alpha+$ $(16(k-1)+8\log(k)+20)n_1\beta$

PPD: The predicted (line) and numerical (square) runtime



Pipelining: The predicted (line) and numerical (square) runtime



Significance

- Advances in massively parallelism, grid computing, and hierarchical data access make performance sensitive to system and problem size
- Scalability is becoming increasingly important
- Poisson solver is a kernel solver used in many naval applications.
- The PPD algorithm provides a scalable solution for Poisson solver
- We also have proposed the general PTH method

Reference

– X.-H. Sun, H. Zhang, and L. Ni, "Efficient Tridiagonal Solvers on Multicomputers," *IEEE Trans. on Computers*, Vol. 41, No. 3, pp.286-296, March 1992.

– X.-H. Sun, "Application and Accuracy of the Parallel Diagonal Dominant Algorithm" *Parallel Computing*, August, 1995.

– X.-H. Sun and W. Zhang, "A Parallel Two-Level Hybrid Method for Tridiagonal Systems, and its Application to Fast Poisson Solvers," *IEEE Trans. on Parallel and Distributed Systems*, Vol. 15, No. 2, pp: 97-106, 2004.

– X.-H. Sun, and S. Moitra, "Performance Comparison of a Set of Periodic and Non-Periodic Tridiagonal Solvers on SP2 and Paragon Parallel Computers," *Concurrency: Practice and Experience*, pp.1-21, Vol.8(10), 1997.

– X.H. Sun, and D. Joslin, "A Simple Parallel Prefix Algorithm for Almost Toeplitz Tridiagonal Systems," *High Speed Computing*, Vol.7, No.4, pp. 547-576, Dec. 1995.

– Y. Zhuang, and X.H. Sun, "A High Order Fast Direct Solver for Singular Poisson Equations," *Journal of Computational Physics*, Vol. 171, pp. 79-94 (2001).

– Y. Zhuang, and X.H. Sun, "A High Order ADI Method For Separable Generalized Helmholtz Equations," *International Journal on Advances in Engineering Software*, Vol. 31, pp. 585-592, August 2000.

All the references can be found at

http://www.cs.iit.edu/~scs/research/scientific-computing.html