ECE 669

Parallel Computer Architecture

Lecture 15

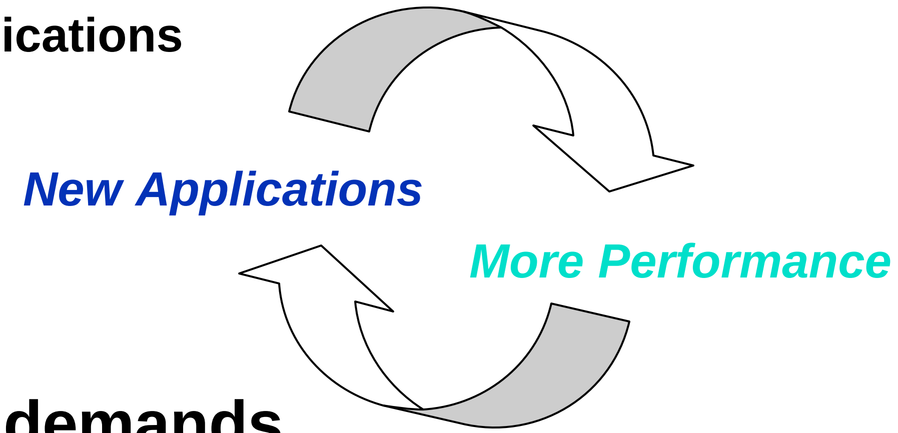
Mid-term Review

Is Parallel Computing Inevitable?

- Application demands: Our insatiable need for computing cycles
- Technology Trends
- Architecture Trends
- Economics
- Current trends:
    - Today’s microprocessors have multiprocessor support
    - Servers and workstations becoming MP: Sun, SGI, DEC, HP!...
    - Tomorrow’s microprocessors are multiprocessors

Application Trends

- Application demand for performance fuels advances in hardware, which enables new appl’ns, which...
    - Cycle drives exponential increase in microprocessor performance
    - Drives parallel architecture harder
        - most demanding applications



- Range of performance demands
    - Need range of system performance with progressively increasing cost

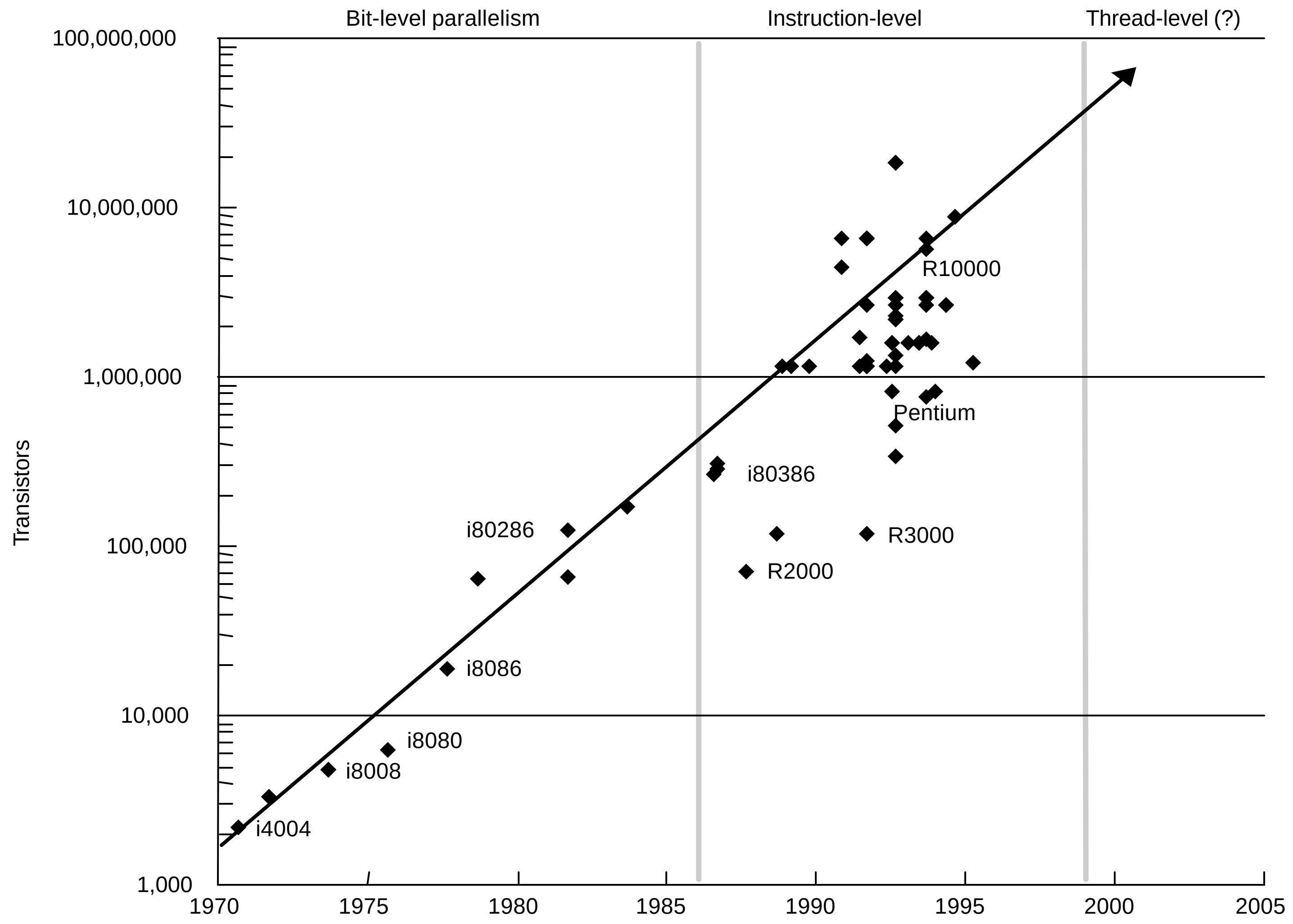
Architectural Trends

- Architecture translates technology's gifts into performance and capability
- Resolves the tradeoff between parallelism and locality
  - Current microprocessor: 1/3 compute, 1/3 cache, 1/3 off-chip connect
  - Tradeoffs may change with scale and technology advances
- Understanding microprocessor architectural trends

  => Helps build intuition about design issues or parallel machines

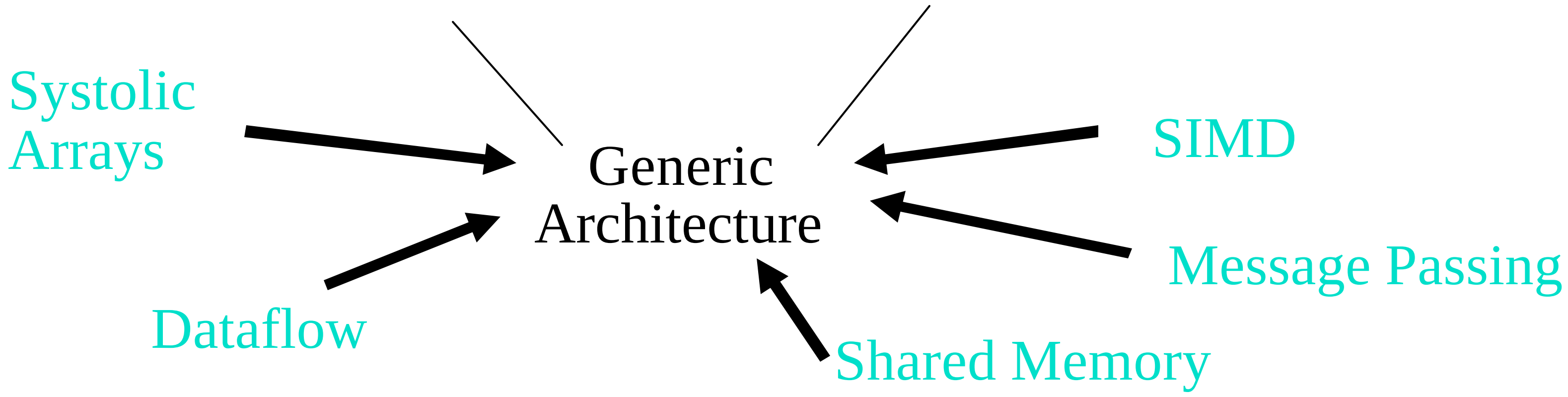
  => Shows fundamental role of parallelism even in "sequential" computers

Phases in “VLSI” Generation



Programming Model

- **Look at major programming models**
  - **Where did they come from?**
  - **What do they provide?**
  - **How have they converged?**
- **Extract general structure and fundamental issues**
- **Reexamine traditional camps from new perspective**



Programming Model

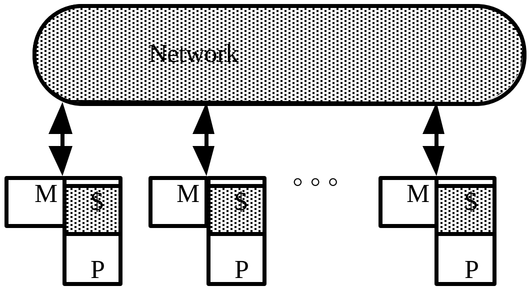
- *Conceptualization of the machine that programmer uses in coding applications*
  - How parts cooperate and coordinate their activities
  - Specifies communication and synchronization operations
- Multiprogramming
  - no communication or synch. at program level
- *Shared address space*
  - like bulletin board
- *Message passing*
  - like letters or phone calls, explicit point to point
- *Data parallel*:
  - more regimented, global actions on data
  - Implemented with shared address space or message passing

Shared Physical Memory

- Any processor can directly reference any memory location
- Any I/O controller - any memory
- Operating system can run on any processor, or all.
  - OS uses shared memory to coordinate
- Communication occurs implicitly as result of loads and stores
- What about application processes?

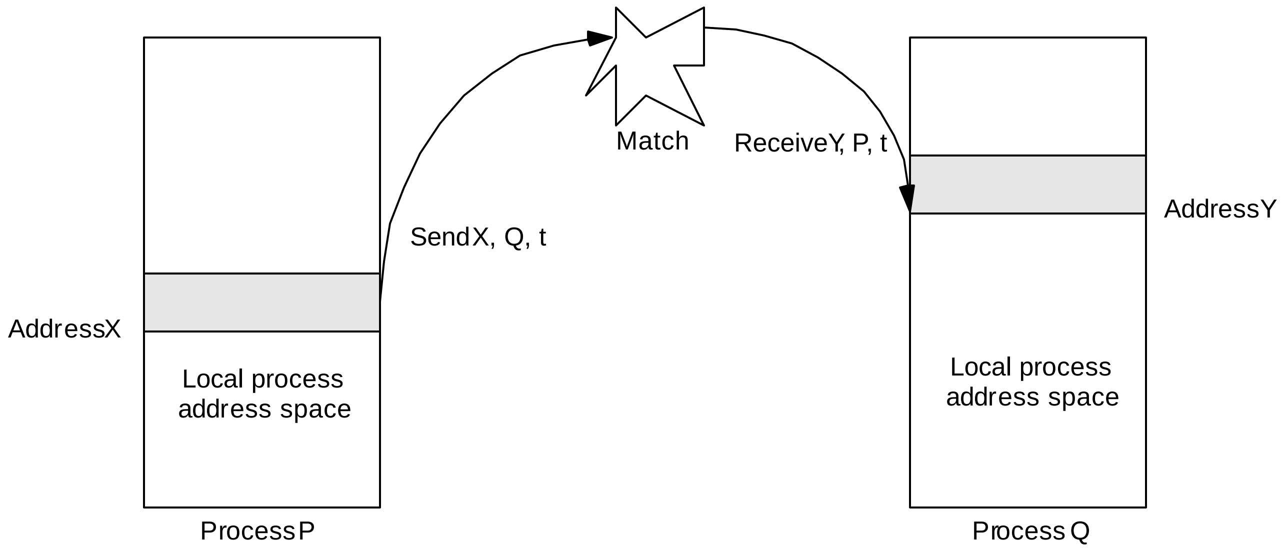
Message Passing Architectures

- Complete computer as building block, including I/O
  - Communication via explicit I/O operations
- Programming model
  - direct access only to private address space (local memory),
  - communication via explicit messages (send/receive)
- High-level block diagram
  - Communication integration?
    - Mem, I/O, LAN, Cluster
  - Easier to build and scale than SAS



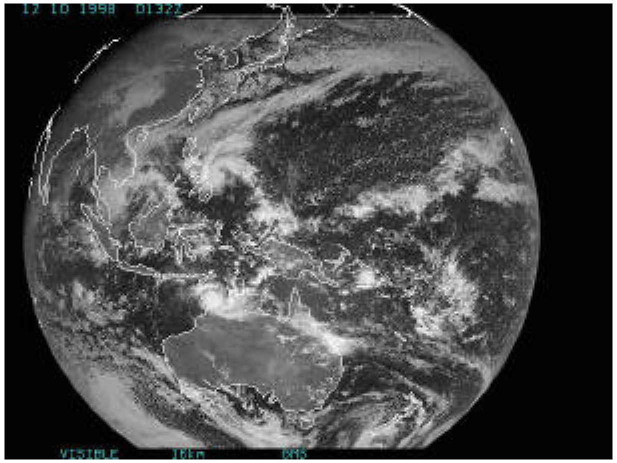
- Programming model more removed from basic hardware operations
  - Library or OS intervention

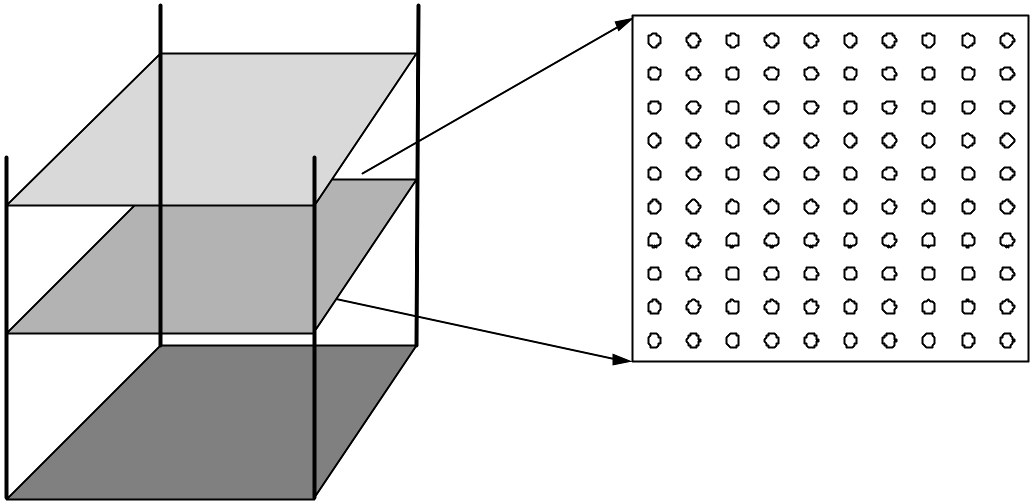
Message-Passing Abstraction



- **Send specifies buffer to be transmitted and receiving process**
- **Recv specifies sending process and application storage to receive into**
- **Memory to memory copy, but need to name processes**
- **Optional tag on send and matching rule on receive**
- **User process names local data and entities in process/tag space too**
- **In simplest form, the send/recv match achieves pairwise synch event**
  - **Other variants too**
- **Many overheads: copying, buffer management, protection**

Simulating Ocean Currents



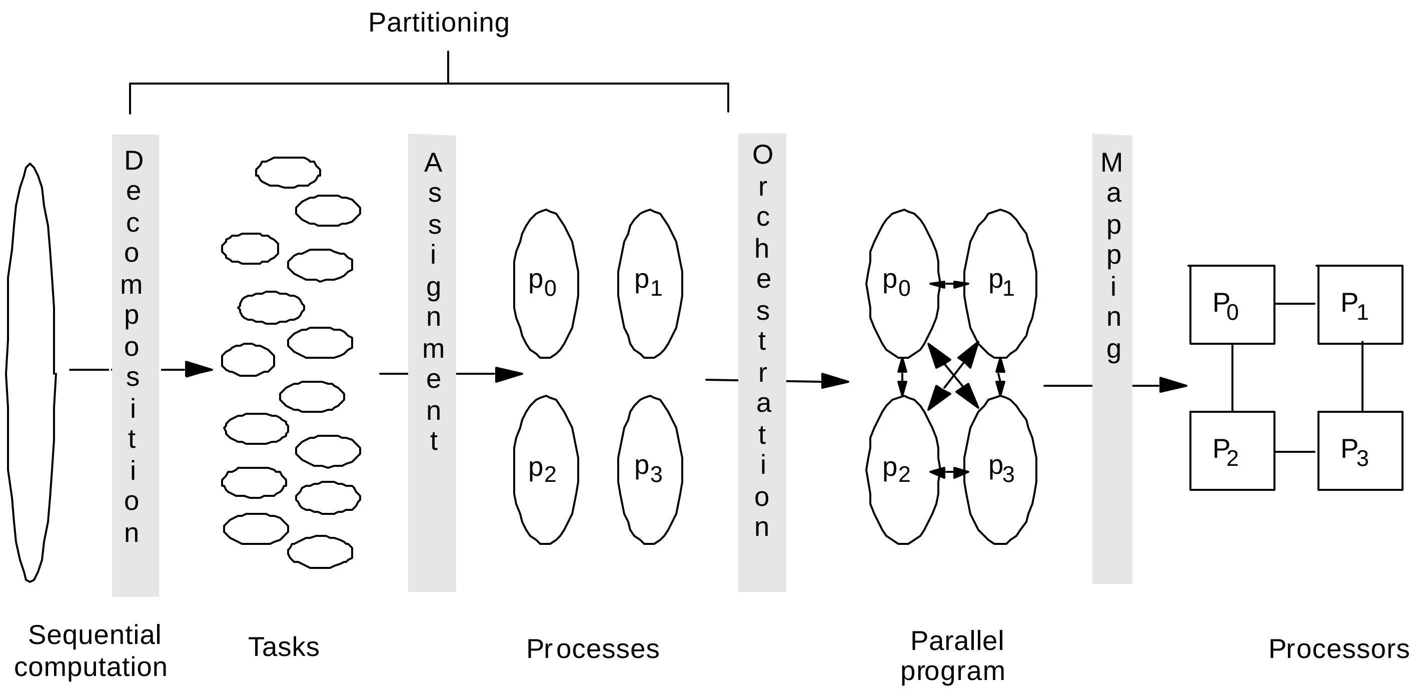


(a) Cross sections

(b) Spatial discretization of a cross section

- Model as two-dimensional grids
  - Discretize in space and time
  - finer spatial and temporal resolution => greater accuracy
- Many different computations per time step
  - set up and solve equations
  - Concurrency across and within grid computations
- Static and regular

4 Steps in Creating a Parallel Program



- **Decomposition of computation in tasks**
- **Assignment of tasks to processes**
- **Orchestration of data access, comm, synch.**
- **Mapping processes to processors**

Discretize

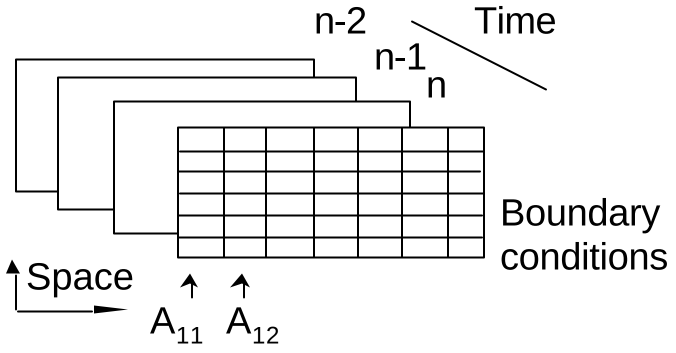
- **Time**

$$\frac{\P A}{\P T} = \frac{A^{n+1} - A^{n}}{\Delta t}$$

Forward difference

  - **Where**

$$\Delta t = \frac{1}{T \text{ steps}}$$



- **Space**
- **1st**

$$\frac{\P A}{\P x} = \frac{A_{i+1} - A_{i}}{\Delta x}$$

  - **Where**

$$\Delta x = \frac{1}{X \text{ grid points}}$$

- **2nd**

$$\frac{\P}{\P x}\left(\frac{\P A}{\P x}\right) = \frac{(A_{i+1} - A_{i}) - (A_{i} - A_{i-1})}{\Delta x^{2}}$$

$$\frac{\P^{2} A}{\P x^{2}} = \frac{A_{i+1} - 2A_{i} + A_{i-1}}{\Delta x^{2}}$$

  - **Can use other discretizations**
    - **Backward**
    - **Leap frog**

1D Case

$$\frac{\P A}{\P T} = \frac{\P^2 A}{\P x^2} + B$$

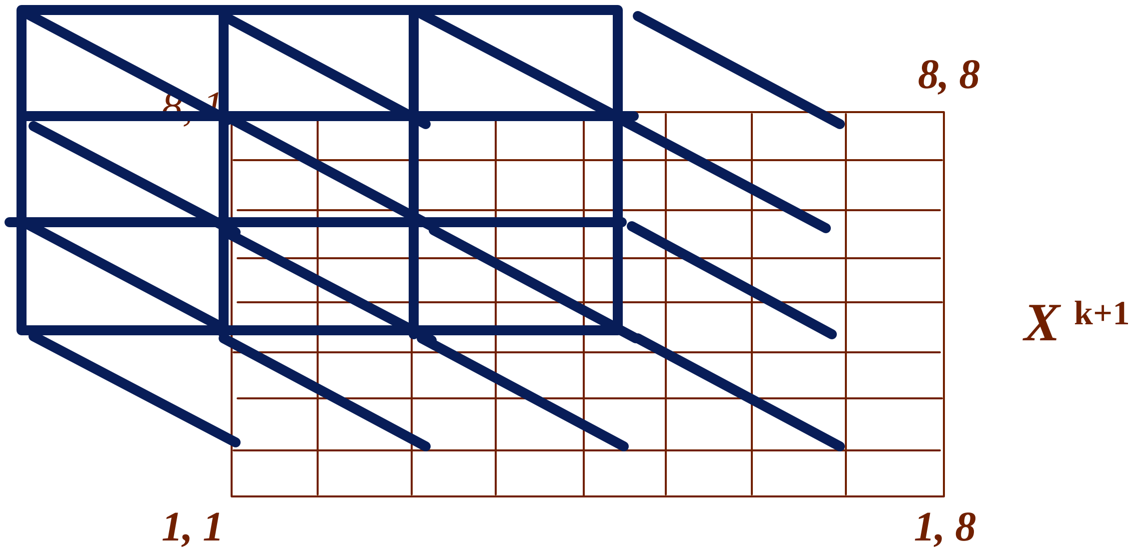
$$\frac{A_i^{n+1} - A_i^n}{\Delta t} = \frac{1}{\Delta x^2}\left[A_{i+1}^n - 2A_i^n + A_{i-1}^n\right] + B_i$$

- **Or** $$A_i^{n+1} = \frac{\Delta t}{\Delta x^2}\left[A_{i+1}^n - 2A_i^n + A_{i-1}^n\right] + B_i \Delta t + A_i^n$$

$$\begin{bmatrix} A_x^{n+1} \\ A_i^{n+1} \\ \\ A_2^{n+1} \\ A_i^{n+1} \end{bmatrix} = \begin{bmatrix} \ddots & \ddots & \ddots & & 0 \\ & & & & \\ & \frac{\Delta t}{\Delta x^2} & \frac{-2\Delta t}{\Delta x^2} + 1 & \frac{\Delta t}{\Delta x^2} & \\ 0 & & \ddots & \ddots & \ddots \\ & & & & \end{bmatrix} \begin{bmatrix} A_x^n \\ A_i \\ \\ A_2^n \\ A_i^n \end{bmatrix} + \begin{bmatrix} \\ \\ B \\ \\ \\ \end{bmatrix}$$

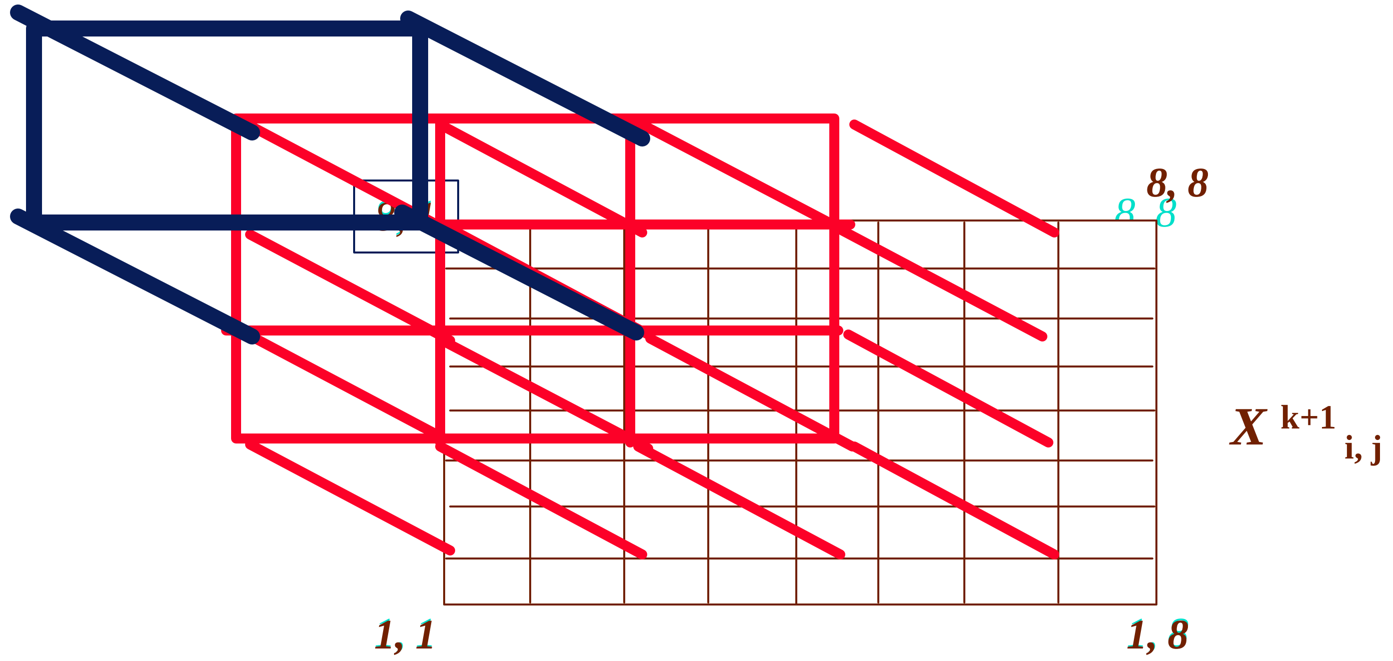
Multigrid

- Basic idea ---> Solve on coarse grid
  ---> then on fine grid



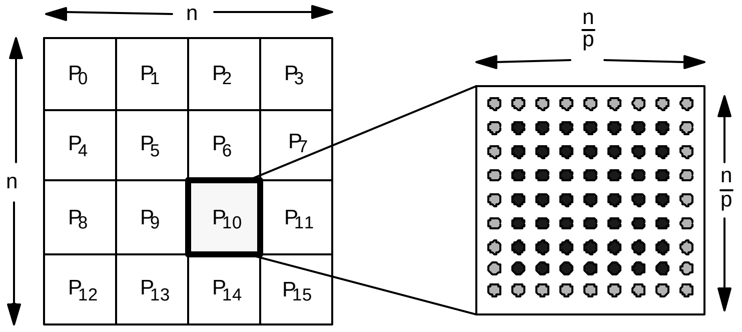
Multigrid

- Basic idea ---> Solve on coarse grid
  ---> then on fine grid



Domain Decomposition

- **Works well for scientific, engineering, graphics, ... applications**
- **Exploits local-biased nature of physical problems**
  - **Information requirements often short-range**
  - **Or long-range but fall off with distance**
- **Simple example: nearest-neighbor grid computation**

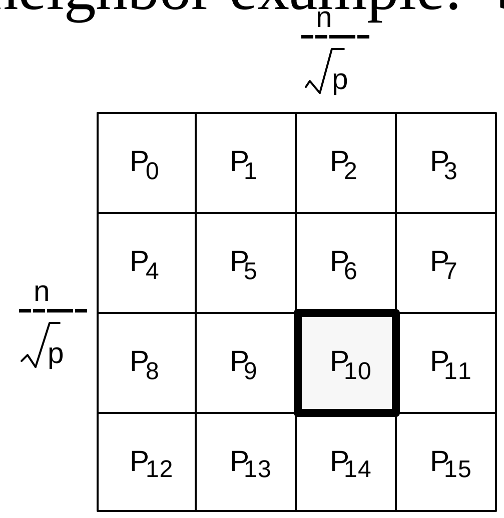


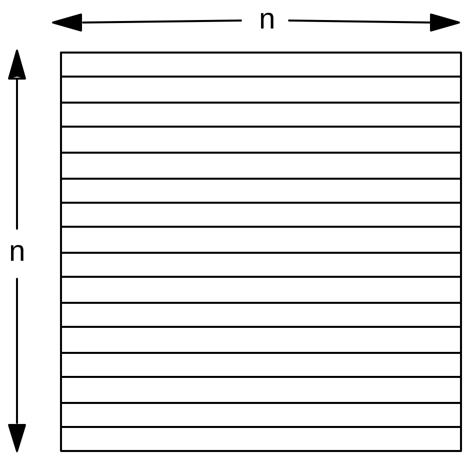
Perimeter to Area comm-to-comp ratio (area to volume in 3-d)

- Depends on *n*,*p*: decreases with *n*, increases with *p*

Domain Decomposition

Best domain decomposition depends on information requirements
Nearest neighbor example: block versus strip decomposition:

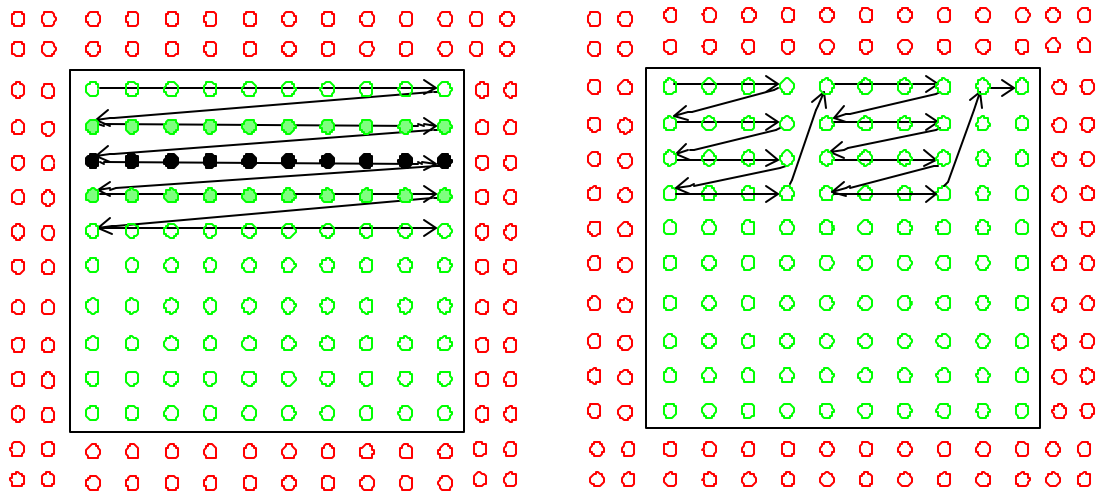




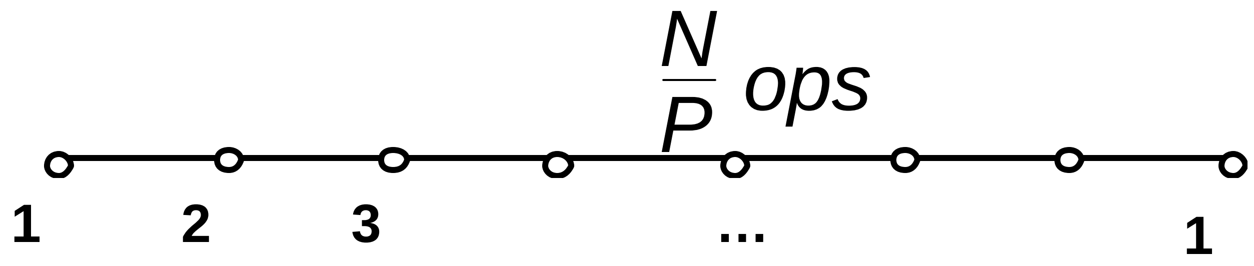
- Comm to comp: $\frac{4*p^{0.5}}{n}$ for block, $\frac{2*p}{n}$ for strip
- Application dependent: strip may be better in other cases

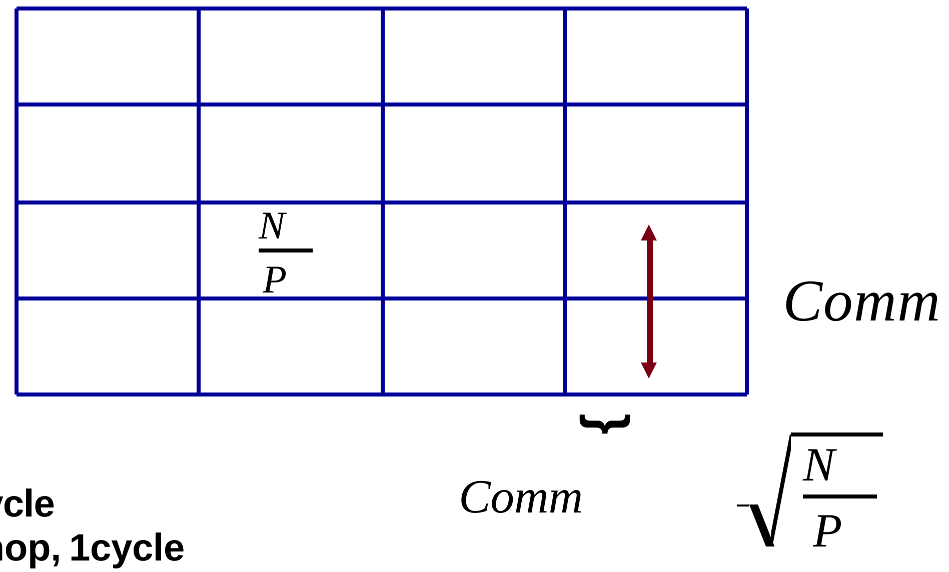
Exploiting Temporal Locality

- **Structure algorithm so working sets map well to hierarchy**
  - **often techniques to reduce inherent communication do well here**
  - **schedule tasks for data reuse once assigned**
- **Solver example: blocking**


(a) Unblocked access pattern in a sweep (b) Blocked access pattern with B = 4

1-D Array of nodes for Jacobi





Model: 1 op, 1cycle
1comm/hop, 1cycle

$$T = \frac{N}{P} + \sqrt{P}$$

Scalability

- $$S_I(N) = N$$
  $$S_R(N) = ?$$
- **Ideal speedup on any number of procs.**
- **<u>Find best *P*</u>**

$$T_{par} = \frac{N}{P} + \sqrt{P}$$
$$\frac{dT}{dP} = 0$$
$$P = N^{\frac{2}{3}} \ldots$$
$$T_{par} = q\left(N^{\frac{1}{3}}\right)$$
$$T_{seg} = N$$
$$S_R(N) = N^{\frac{2}{3}} = \frac{N}{N^{\frac{1}{3}}}$$

- **So,** $$y(N) = \frac{N^{\frac{2}{3}}}{N} = \frac{S_R(N)}{S_I(N)} = N^{-\frac{1}{3}}$$
- **So, 1-D array is** $\frac{1}{N^{\frac{1}{3}}}$ **scalable for Jacobi**

Detailed Example

$$\begin{aligned} p &= 10 \times 10^{6} \\ c &= 0.1 \times 10^{6} \\ m &= 0.1 \times 10^{6} \\ P &= 10 \end{aligned}$$

$$\frac{p}{c} = \frac{N}{P}$$

or $$\frac{10 \times 10^{6}}{0.1 \times 10^{6}} = \frac{N}{10}$$

or $N = 1000$ for balance

also
$$\begin{aligned} R_M &= m \\ \frac{N}{P} &= m \\ \frac{1000}{10} &= 100 = m \end{aligned}$$

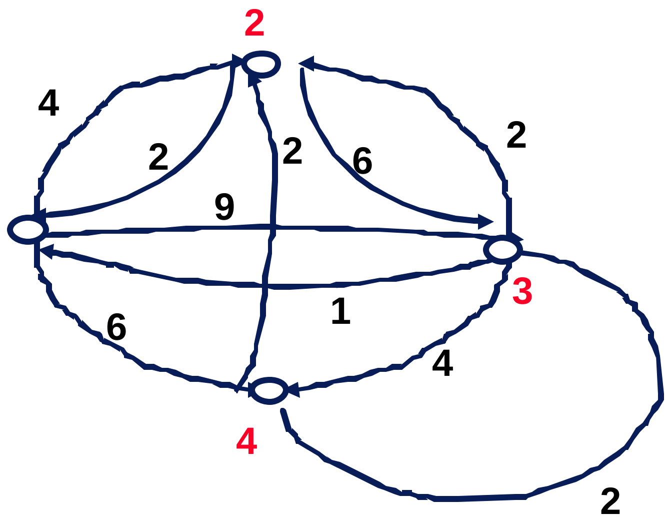
Memory size of m = 100 yields a balanced machine.

Better Algorithm, but basically Branch and Bound

- Little et al.
- Basic Ideas:

Cost matrix

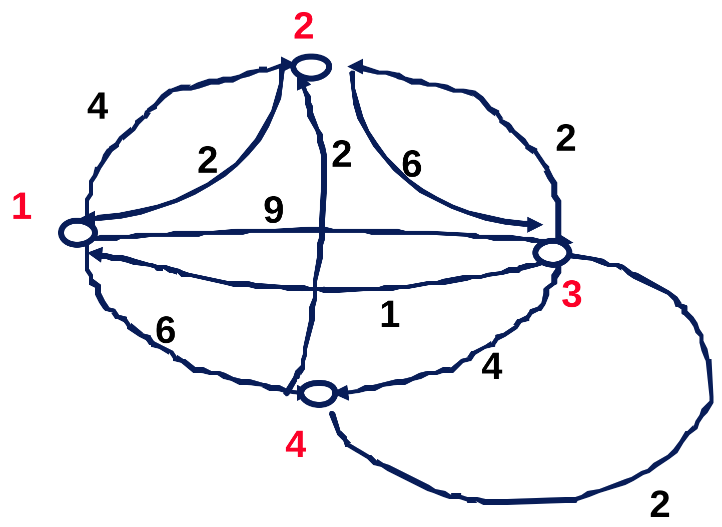
	1	2	3	4
1	•	4	9	6
2	2	•	6	•
3	1	2	•	4
4	2	•	2	•



Better Algorithm, but basically Branch and Bound

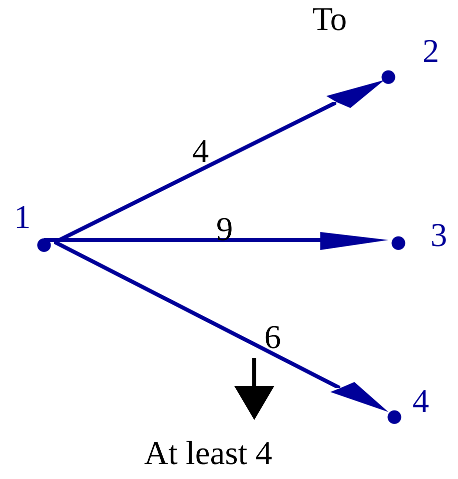
Little et al.

	1	2	3	4	
1	•	0 ~~4~~	5 ~~9~~	2 ~~6~~	→ 4
2	2	•	6	•	
3	1	2	•	4	
4	2	•	2	•	



Notion of reduction:

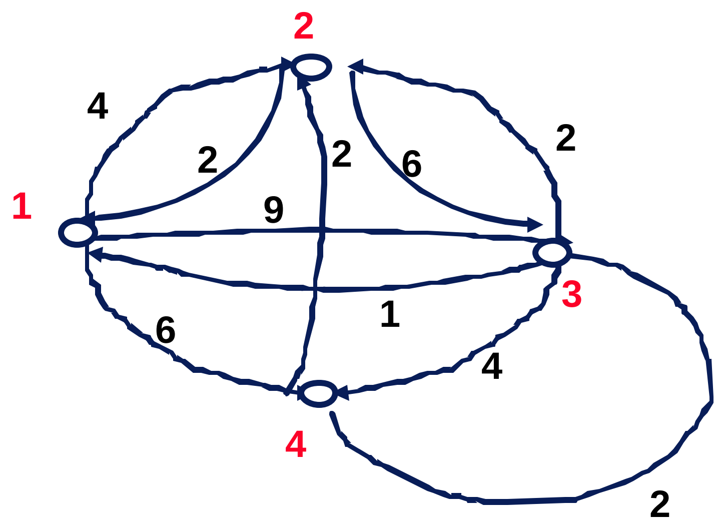
- Subtract same value from each row or column

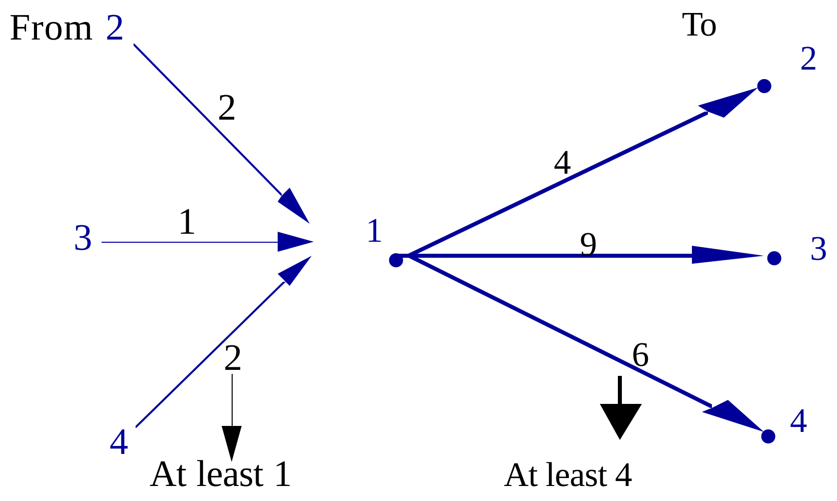


Better Algorithm, but basically Branch and Bound

Little et al.

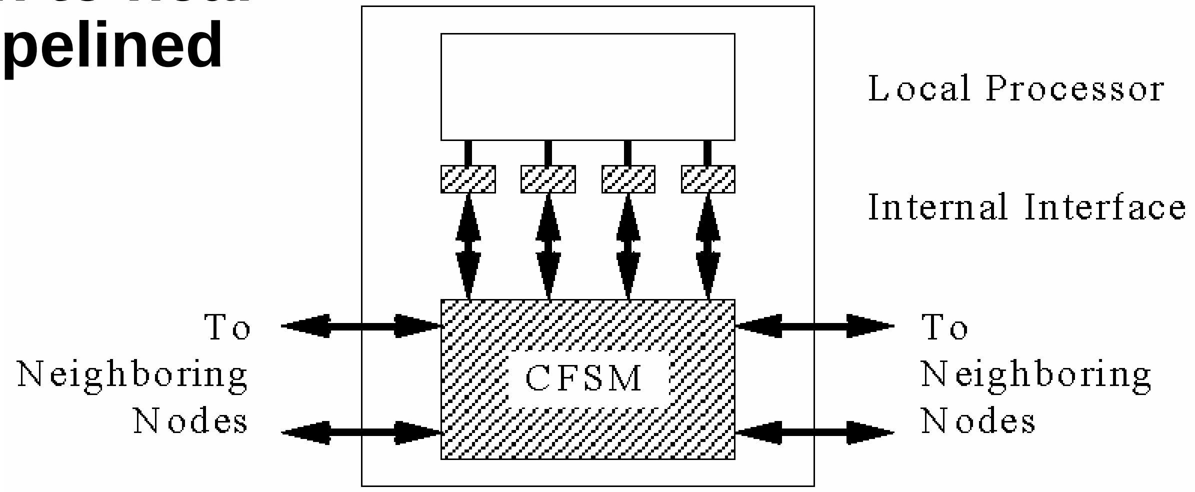
	1	2	3	4	
1	•	0 ~~4~~	5 ~~9~~	2 ~~6~~	→ 4
2	~~2~~ 1	•	6	•	
3	~~1~~ 0	2	•	4	
4	~~2~~ 1	•	2	•	
	↓ 1				





Communication Finite State Machine

- Each node has a processing part and a communications part
- Interface to local processor is a FIFO
- Communication to near-neighbors is pipelined



Statically Programmed Communication

- **Data transferred one node in one cycle**
- **Inter-processor path may require multiple cycles**
- **Heavy arrows represent local transfers**
- **Grey arrows represent non-local transfers**