Review: From problem to parallel algorithm

- Mathematical formulations of “interesting” problems abound
- Poisson’s equation
  - Sources: Electrostatics, gravity, fluid flow, image processing (!)
  - Numerical solution: Discretize and solve $Ax = b$
  - Many methods, which serve as building blocks for other problems and algorithms
- Jacobi’s method: Easy to parallelize iterative method with slow convergence

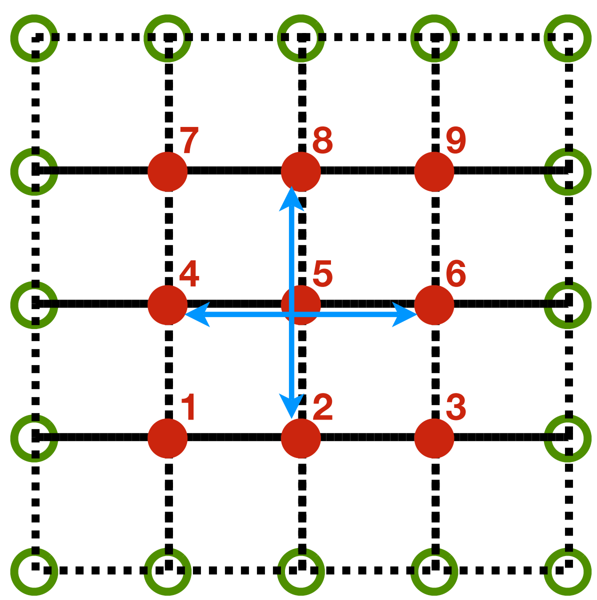
Recall: Algorithms for 2-D (3-D) Poisson, $N=n^2$ ($=n^3$)

Algorithm	Serial	PRAM	Memory	# procs
Dense LU	N^3	N	N^2	N^2
Band LU	N^2 ($N^{7/3}$)	N	$N^{3/2}$ ($N^{5/3}$)	N ($N^{4/3}$)
Jacobi	**N^2 ($N^{5/3}$)**	**N ($N^{2/3}$)**	**N**	**N**
Explicit inverse	N^2	$\log N$	N^2	N^2
Conj. grad.	$N^{3/2}$ ($N^{4/3}$)	$N^{1/2(1/3)} \log N$	N	N
RB SOR	$N^{3/2}$ ($N^{4/3}$)	$N^{1/2}$ ($N^{1/3}$)	N	N
Sparse LU	$N^{3/2}$ (N^2)	$N^{1/2}$	$N \log N$ ($N^{4/3}$)	N
FFT	$N \log N$	$\log N$	N	N
Multigrid	N	$\log^2 N$	N	N
Lower bound	**N**	**$\log N$**	**N**	

PRAM = idealized parallel model with zero communication cost.
Source: Demmel (1997)

Recall: 2-D Poisson Equation

Graph and stencil

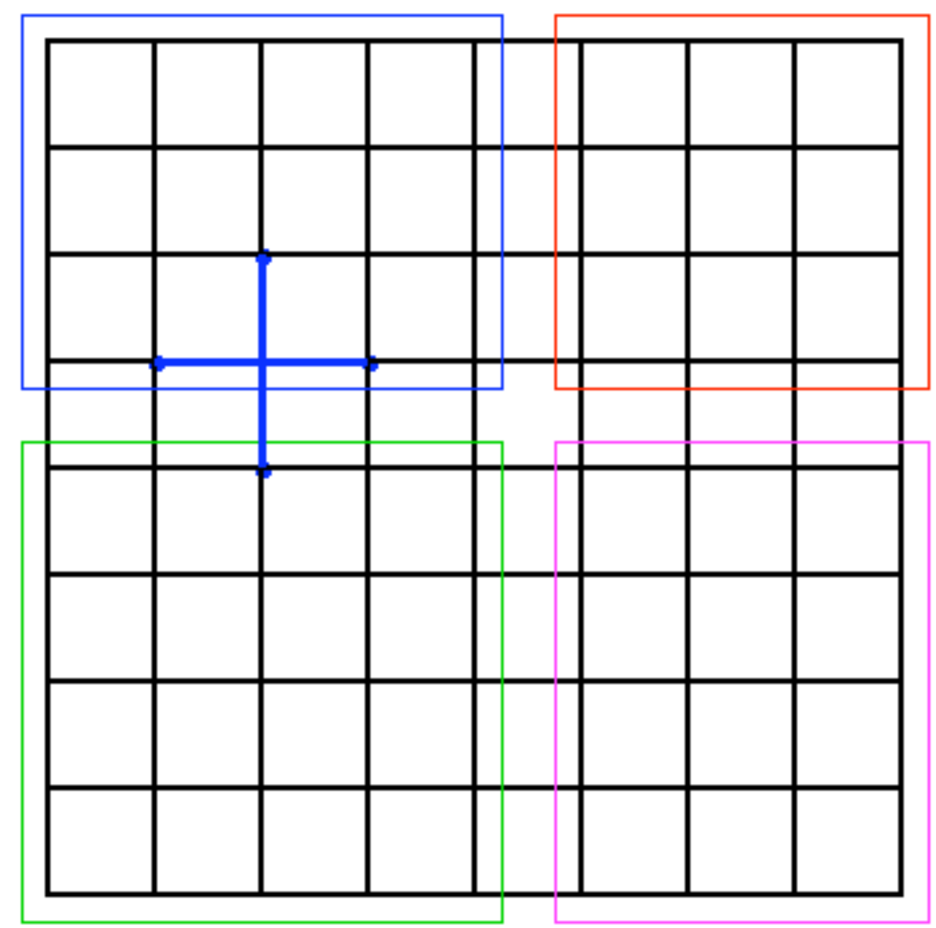


$$
T = \begin{array}{c}
\begin{array}{ccccccccc} & 2 & & 4 & 5 & 6 & & 8 & \end{array} \\
\left(\begin{array}{ccc|ccc|ccc}
4 & -1 & & -1 & & & & & \\
-1 & 4 & -1 & & -1 & & & & \\
 & -1 & 4 & & & -1 & & & \\
\hline
-1 & & & 4 & -1 & & -1 & & \\
 & -1 & & -1 & 4 & -1 & & -1 & \\
 & & -1 & & -1 & 4 & & & -1 \\
\hline
 & & & -1 & & & 4 & -1 & \\
 & & & & -1 & & -1 & 4 & -1 \\
 & & & & & -1 & & -1 & 4
\end{array}\right)
\end{array}
$$

Recall: Jacobi's method is easy to parallelize

- Parallelism: Update all points independently
- Partition domain into blocks
  - n^2/p elements / block
- Communicate at boundaries
  - n/p per neighbor
  - Small if $n >> p$

Block partition domain



I: More Poisson
II: Performance metrics and models

Prof. Richard Vuduc

Georgia Institute of Technology

CSE/CS 8803 PNA, Spring 2008

[L.04] Thursday, January 17, 2008

Sources for today's material

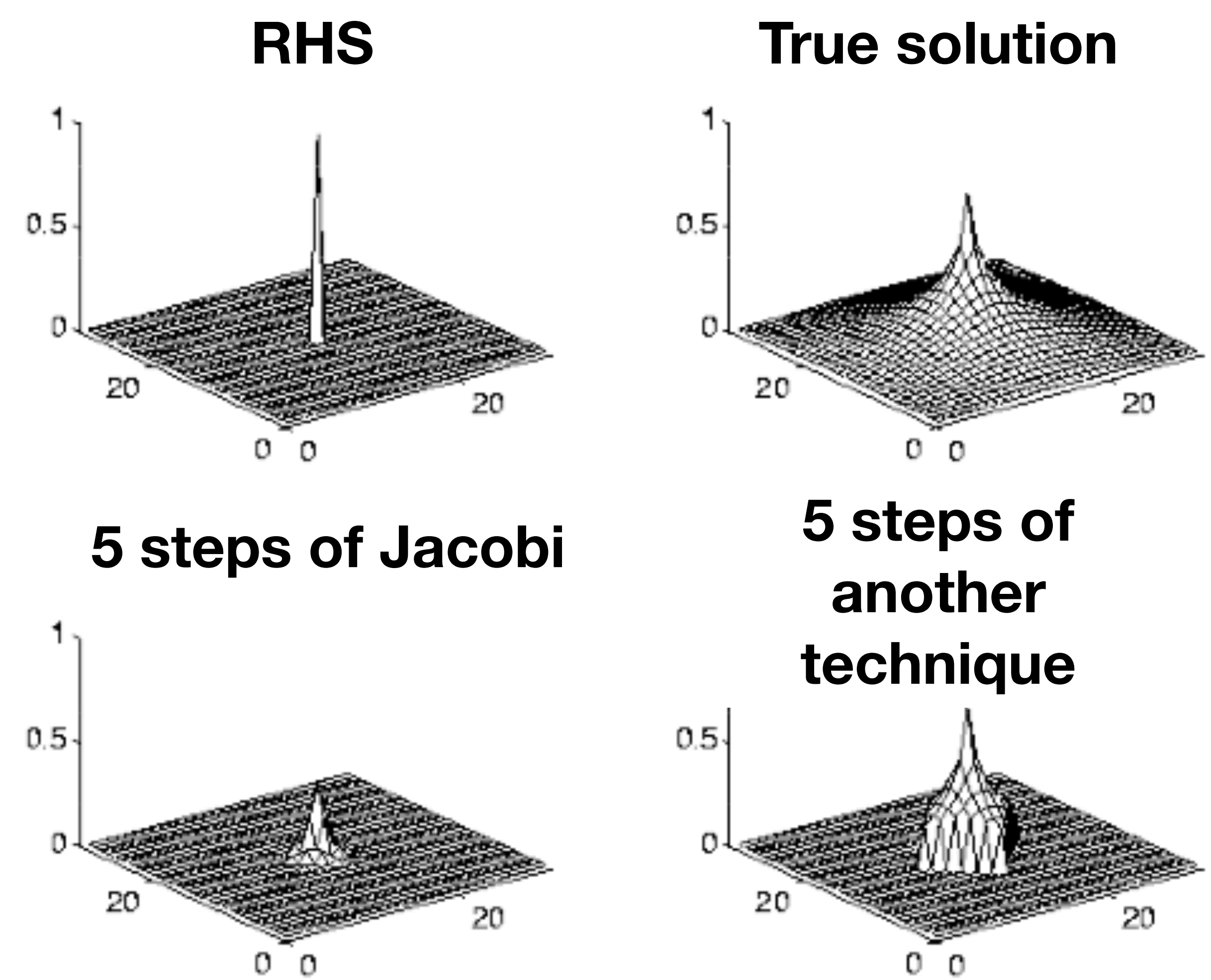
- CS 267 (Yelick & Demmel, UCB)
- "Sourcebook", eds. Dongarra, *et al*.
- Mike Heath at UIUC
- "*Intro to the CG method w/o the agonizing pain*," by Jonathan Shewchuk (UCB)

Algorithms for 2-D (3-D) Poisson, $N=n^2$ ($=n^3$)

Algorithm	Serial	PRAM	Memory	# procs
Dense LU	N^3	N	N^2	N^2
Band LU	N^2 ($N^{7/3}$)	N	$N^{3/2}$ ($N^{5/3}$)	N ($N^{4/3}$)
Jacobi	N^2 ($N^{5/3}$)	N ($N^{2/3}$)	N	N
Explicit inverse	N^2	$\log N$	N^2	N^2
Conj. grad.	$N^{3/2}$ ($N^{4/3}$)	$N^{1/2(1/3)} \log N$	N	N
RB SOR	**$N^{3/2}$ ($N^{4/3}$)**	**$N^{1/2}$ ($N^{1/3}$)**	**N**	**N**
Sparse LU	$N^{3/2}$ (N^2)	$N^{1/2}$	$N \log N$ ($N^{4/3}$)	N
FFT	$N \log N$	$\log N$	N	N
Multigrid	N	$\log^2 N$	N	N
Lower bound	**N**	**$\log N$**	**N**	

PRAM = idealized parallel model with zero communication cost.
Source: Demmel (1997)

Can we speed up the rate at which information propagates?



Can “speed up” info propagation?

- **Jacobi**:

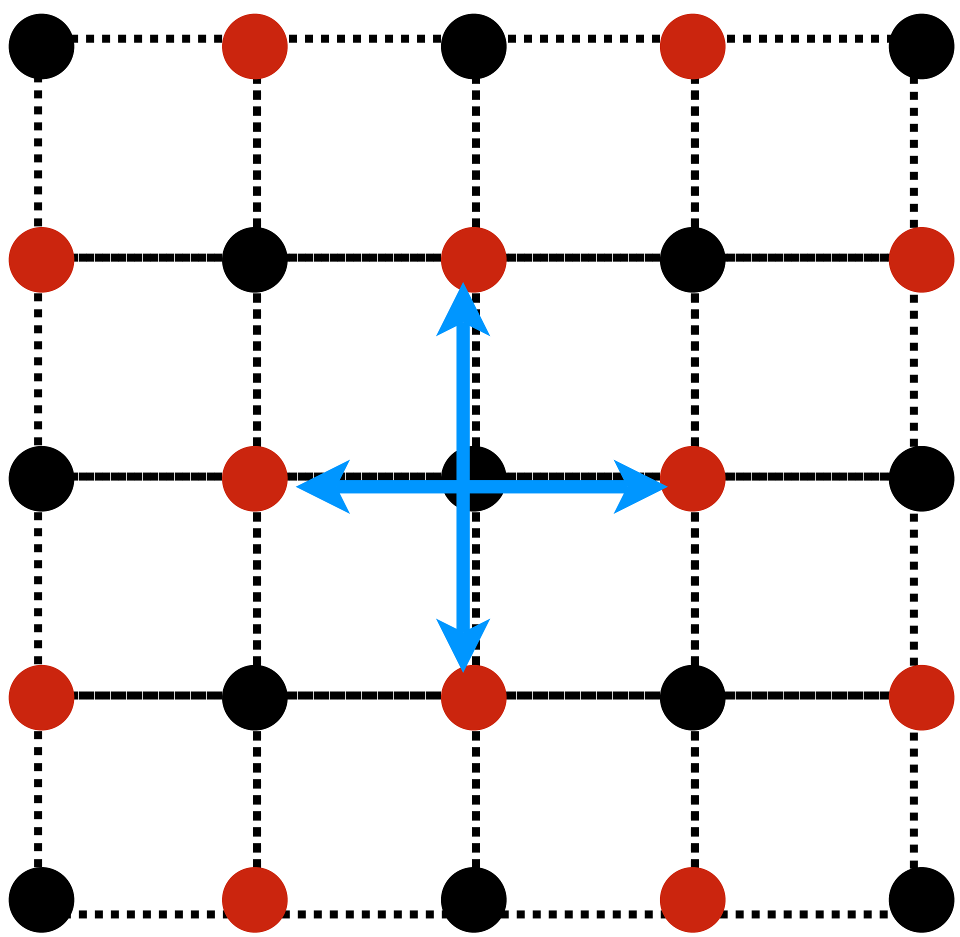
$$u_{i,j}^{t+1} = \frac{1}{4}\left(u_{i-1,j}^{t} + u_{i+1,j}^{t} + u_{i,j-1}^{t} + u_{i,j+1}^{t} + h^2 f_{i,j}\right)$$

- If processing in lexicographic order, can use “**most recent**” values

$$u_{i,j}^{t+1} = \frac{1}{4}\left(u_{i-1,j}^{t+1} + u_{i+1,j}^{t} + u_{i,j-1}^{t+1} + u_{i,j+1}^{t} + h^2 f_{i,j}\right)$$

- **“Gauss-Seidel” algorithm**

Red-black Gauss-Seidel



- Alternately update **R** & **B** subsets
- General graphs?

- **Not much improvement**
  - Convergence: (only) 2x faster
  - PRAM: 2x parallel steps

Successive overrelaxation (SOR)

- Rewrite Jacobi as "**original + correction**:"

$$u_{i,j}^{t+1} = u_{i,j}^{t} + \Delta_{i,j}$$

- If "correction" is a good direction, accelerate by **relaxation factor ω > 1**:

$$u_{i,j}^{t+1} = u_{i,j}^{t} + \omega \cdot \Delta_{i,j}$$

- **Red-black SOR**: Alternately apply the following to red, black subsets

$$u_{i,j}^{t+1} = (1-\omega)u_{i,j}^{t} + \frac{\omega}{4}\left(u_{i-1,j}^{t} + u_{i+1,j}^{t} + u_{i,j-1}^{t} + u_{i,j+1}^{t} + h^2 f_{i,j}\right)$$

Red-black SOR

- **Red-black SOR**: Alternately apply the following to red, black subsets

$$u_{i,j}^{t+1} = (1-\omega)u_{i,j}^{t} + \frac{\omega}{4}\left(u_{i-1,j}^{t} + u_{i+1,j}^{t} + u_{i,j-1}^{t} + u_{i,j+1}^{t} + h^2 f_{i,j}\right)$$

- Can show, for Poisson, that error minimized when: [Demmel (1997)]

$$1 < \quad \omega = \frac{2}{1+\sin\frac{\pi}{N+1}} \quad < 2$$

- Can also show no. of steps to converge is $O(n)$ vs. Jacobi's $O(n^2)$
- Serial complexity = $O(n^3 = N^{3/2})$ vs. Jacobi's $O(n^4 = N^2)$. [PRAM?]

Algorithms for 2-D (3-D) Poisson, $N=n^2$ ($=n^3$)

Algorithm	Serial	PRAM	Memory	# procs
Dense LU	N^3	N	N^2	N^2
Band LU	N^2 ($N^{7/3}$)	N	$N^{3/2}$ ($N^{5/3}$)	N ($N^{4/3}$)
Jacobi	N^2 ($N^{5/3}$)	N ($N^{2/3}$)	N	N
Explicit inverse	N^2	$\log N$	N^2	N^2
Conj. grad.	$N^{3/2}$ ($N^{4/3}$)	$N^{1/2(1/3)} \log N$	N	N
RB SOR	$N^{3/2}$ ($N^{4/3}$)	$N^{1/2}$ ($N^{1/3}$)	N	N
Sparse LU	$N^{3/2}$ (N^2)	$N^{1/2}$	$N \log N$ ($N^{4/3}$)	N
FFT	$N \log N$	$\log N$	N	N
Multigrid	N	$\log^2 N$	N	N
Lower bound	N	$\log N$	N	

PRAM = idealized parallel model with zero communication cost.
Source: Demmel (1997)

Ax=b solves a minimization problem

- If A is **symmetric positive definite**, then the quadratic form,

$$\phi(x) = \frac{1}{2}x^T A x - x^T b$$

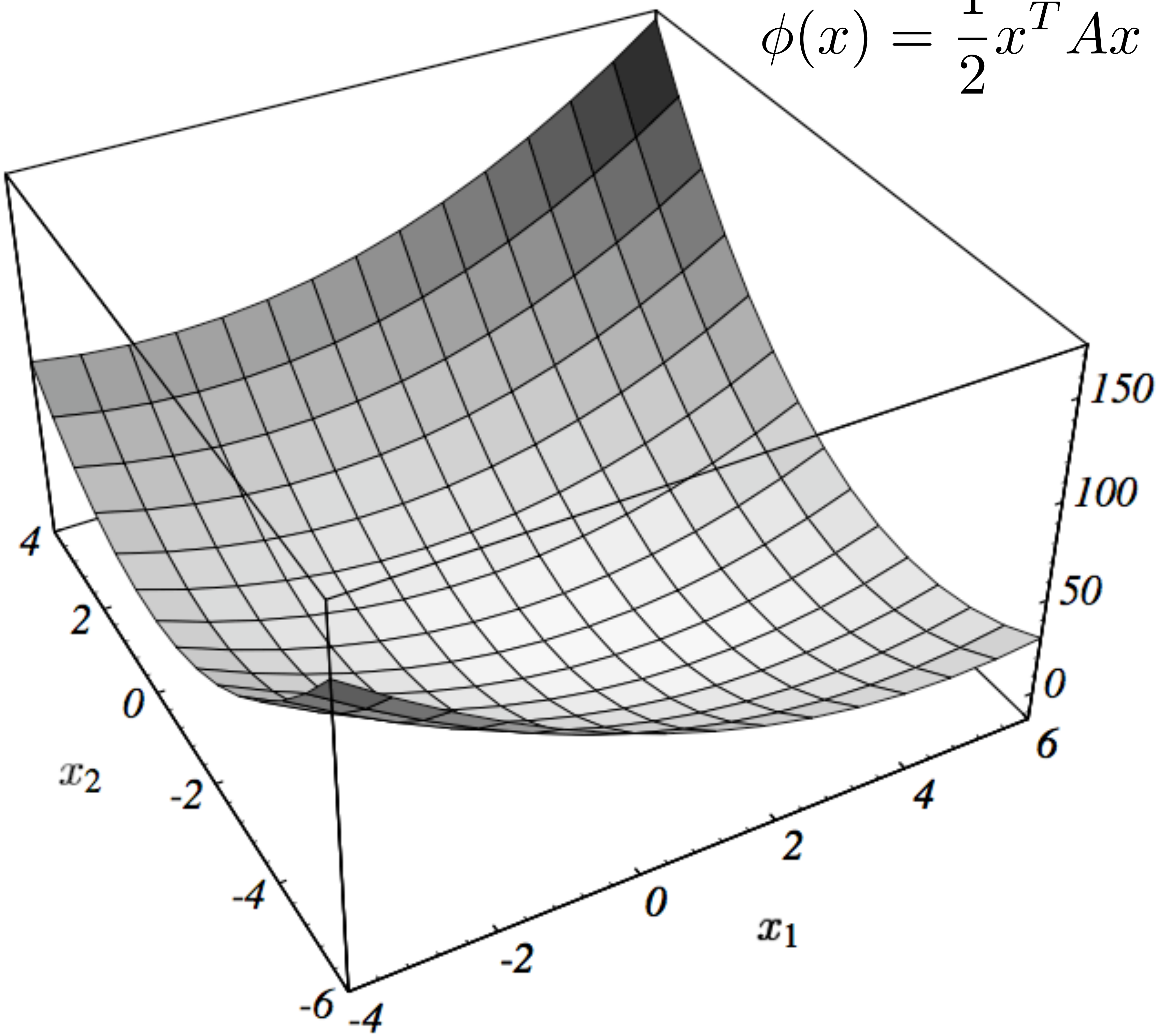
- is **minimized** when

$$Ax = b$$

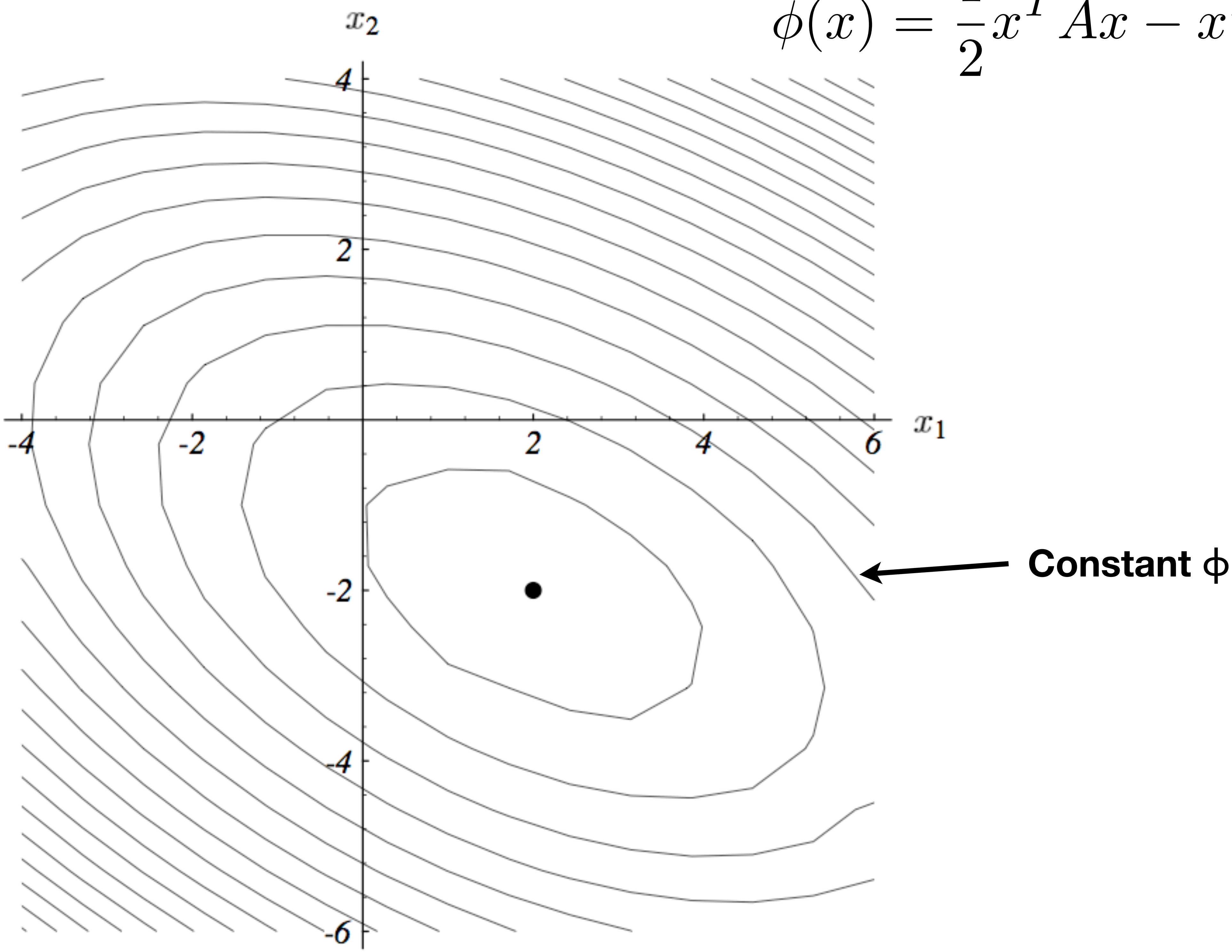
- Intuition? Consider

$$A = \begin{pmatrix} 3 & 2 \\ 2 & 6 \end{pmatrix} \qquad b = \begin{pmatrix} 2 \\ -8 \end{pmatrix}$$

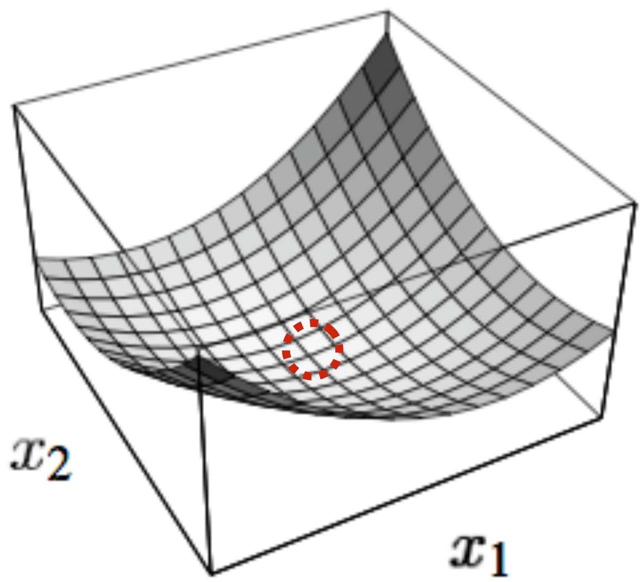
$$\phi(x) = \frac{1}{2}x^T A x - x^T b$$



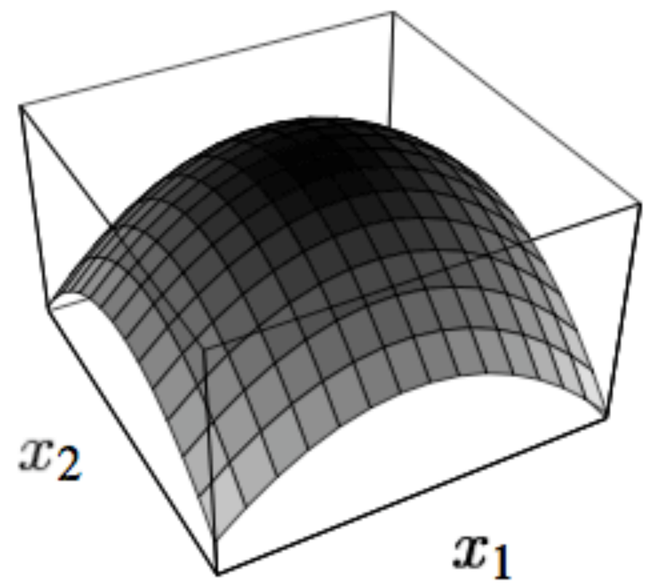
$$\phi(x) = \frac{1}{2}x^T A x - x^T b$$



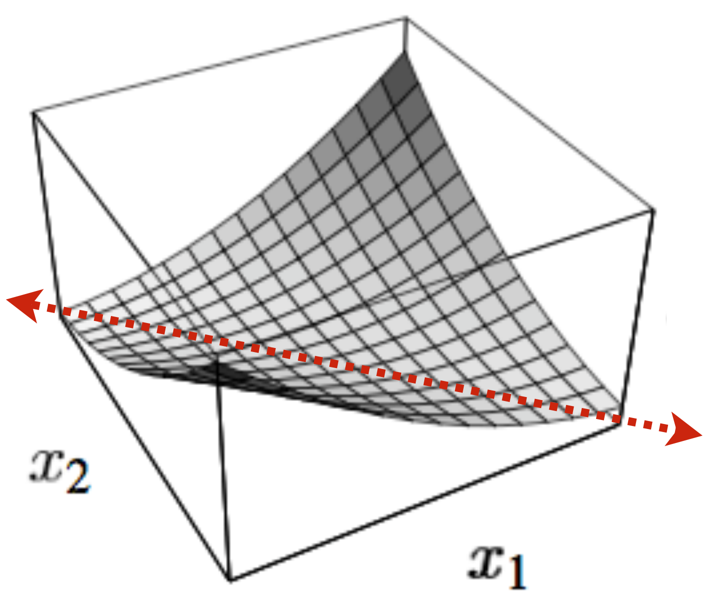
Positive-definite



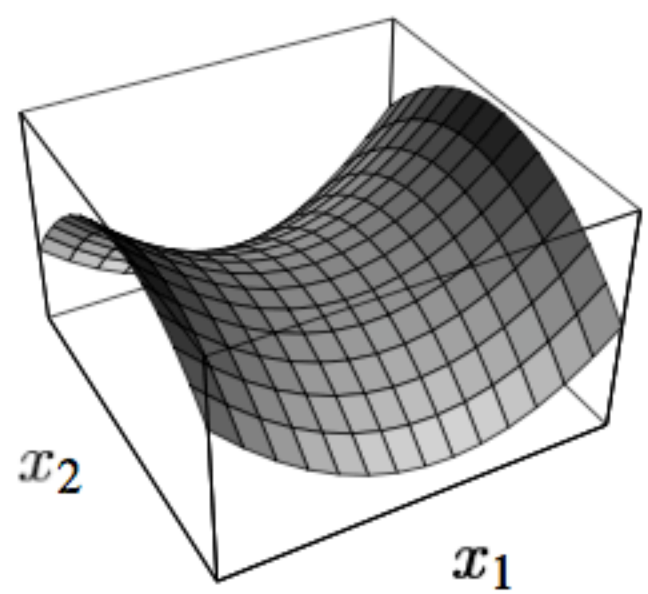
Negative-definite



Singular, positive-indefinite



Indefinite



Why the quadratic form?

$$\phi(x) = \frac{1}{2}x^T Ax - x^T b$$

- Consider error between approx. and true solution at step k of some method:

$$\begin{aligned} e &= x_{\text{approx}} - x_* \\ ||e||_A^2 &= e^T Ae \end{aligned}$$

Some additional observations about this minimization problem

- In general, an iterative numerical optimization method has the form

$$x_{k+1} = x_k + \alpha \cdot s_k$$

- Choose α to minimize $\phi(x_k + \alpha s_k)$
- Negative gradient is the residual vector

$$-\nabla \phi(x) = b - Ax \triangleq r$$

- Can show analytically that

$$\alpha = \frac{r_k^T s_k}{s_k^T A s_k}$$

Conjugate gradient algorithm for solving linear systems

$\boldsymbol{x}_0 =$ initial guess

$\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$

$\boldsymbol{s}_0 = \boldsymbol{r}_0$

for $k = 0, 1, 2, \ldots$

$\quad \alpha_k = \boldsymbol{r}_k^T \boldsymbol{r}_k / \boldsymbol{s}_k^T \boldsymbol{A} \boldsymbol{s}_k$

$\quad \boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{s}_k$

$\quad \boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k \boldsymbol{A} \boldsymbol{s}_k$ **(Sparse) matrix-vector multiply**

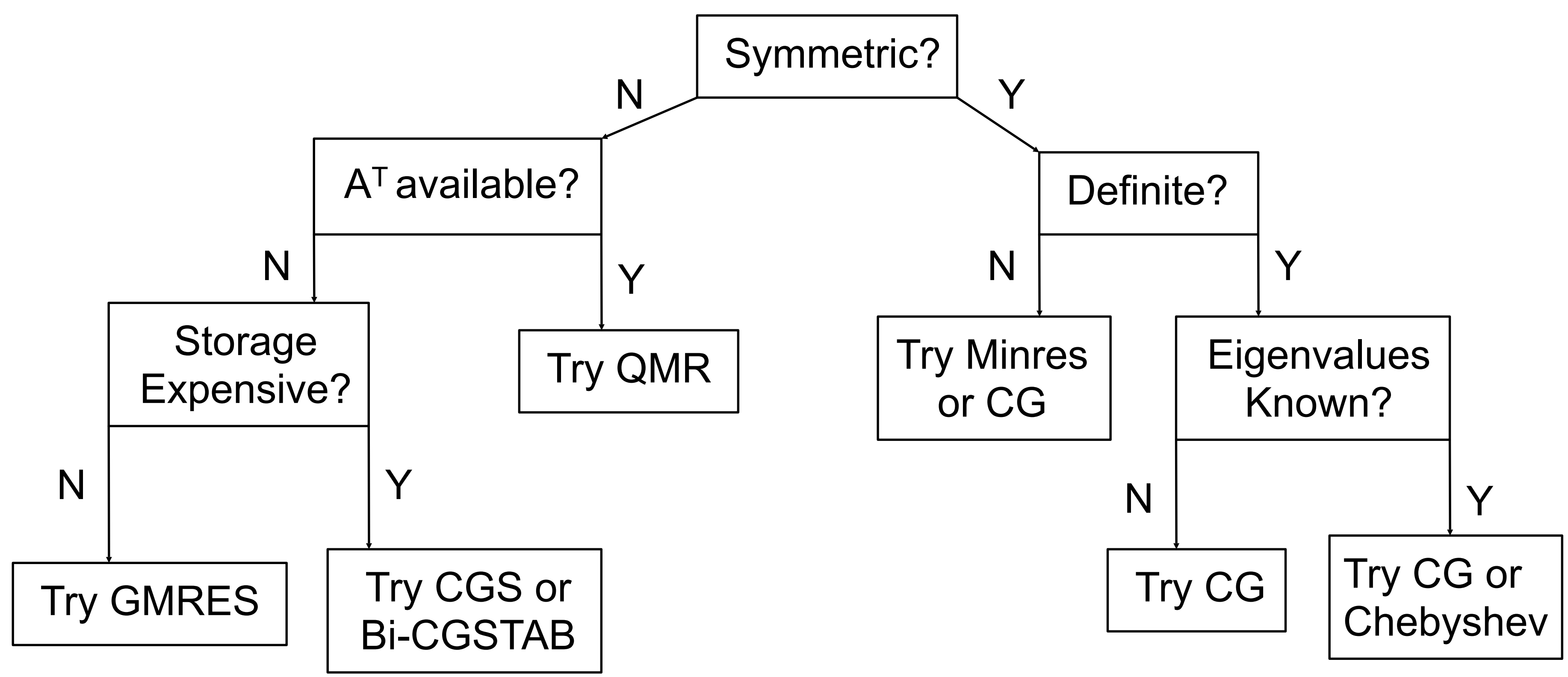
$\quad \beta_{k+1} = \boldsymbol{r}_{k+1}^T \boldsymbol{r}_{k+1} / \boldsymbol{r}_k^T \boldsymbol{r}_k$

$\quad \boldsymbol{s}_{k+1} = \boldsymbol{r}_{k+1} + \beta_{k+1} \boldsymbol{s}_k$

end

Refer to “Templates” book for broad survey of iterative linear solvers

- http://www.netlib.org/templates



Algorithms for 2-D (3-D) Poisson, $N=n^2$ ($=n^3$)

Algorithm	Serial	PRAM	Memory	# procs
Dense LU	N^3	N	N^2	N^2
Band LU	N^2 ($N^{7/3}$)	N	$N^{3/2}$ ($N^{5/3}$)	N ($N^{4/3}$)
Jacobi	N^2 ($N^{5/3}$)	N ($N^{2/3}$)	N	N
Explicit inverse	N^2	$\log N$	N^2	N^2
Conj. grad.	$N^{3/2}$ ($N^{4/3}$)	$N^{1/2(1/3)} \log N$	N	N
RB SOR	$N^{3/2}$ ($N^{4/3}$)	$N^{1/2}$ ($N^{1/3}$)	N	N
Sparse LU	$N^{3/2}$ (N^2)	$N^{1/2}$	$N \log N$ ($N^{4/3}$)	N
FFT	$N \log N$	$\log N$	N	N
Multigrid	N	$\log^2 N$	N	N
Lower bound	N	$\log N$	N	

PRAM = idealized parallel model with zero communication cost.
Source: Demmel (1997)

Administrivia

Administrative stuff

- Accounts: Apparently, you already have them or will soon (!)
  - Try logging into ‘warp1’ with your UNIX account password
  - If it doesn’t work, go see TSO Help Desk (and good luck!)
    - CCB 148 / M-F 7a-5p / 404.894.7065 / AIM:tsohlpdsk
- Summer internships at national and industrial research labs

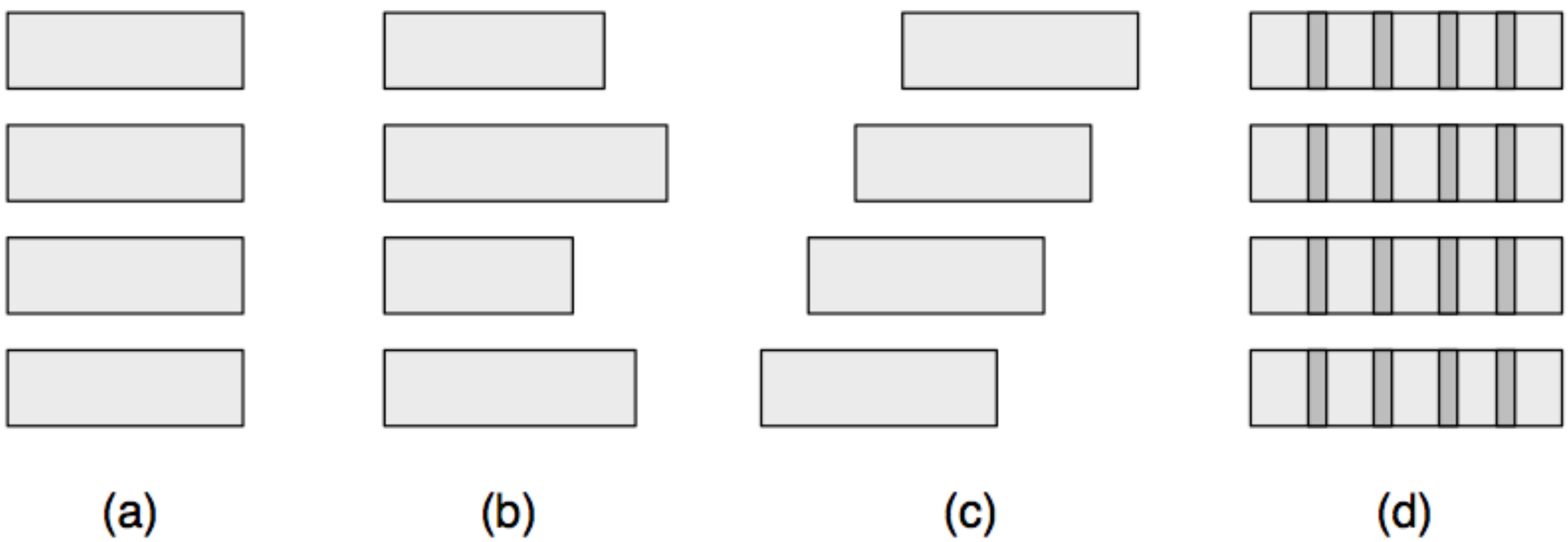
Metrics and models of efficiency and scalability

Outline

- **Parallel efficiency**: “Effectiveness” of parallel algorithm compared to serial
- **Scalability** - definitions, problem scaling, isoefficiency
- Simple models

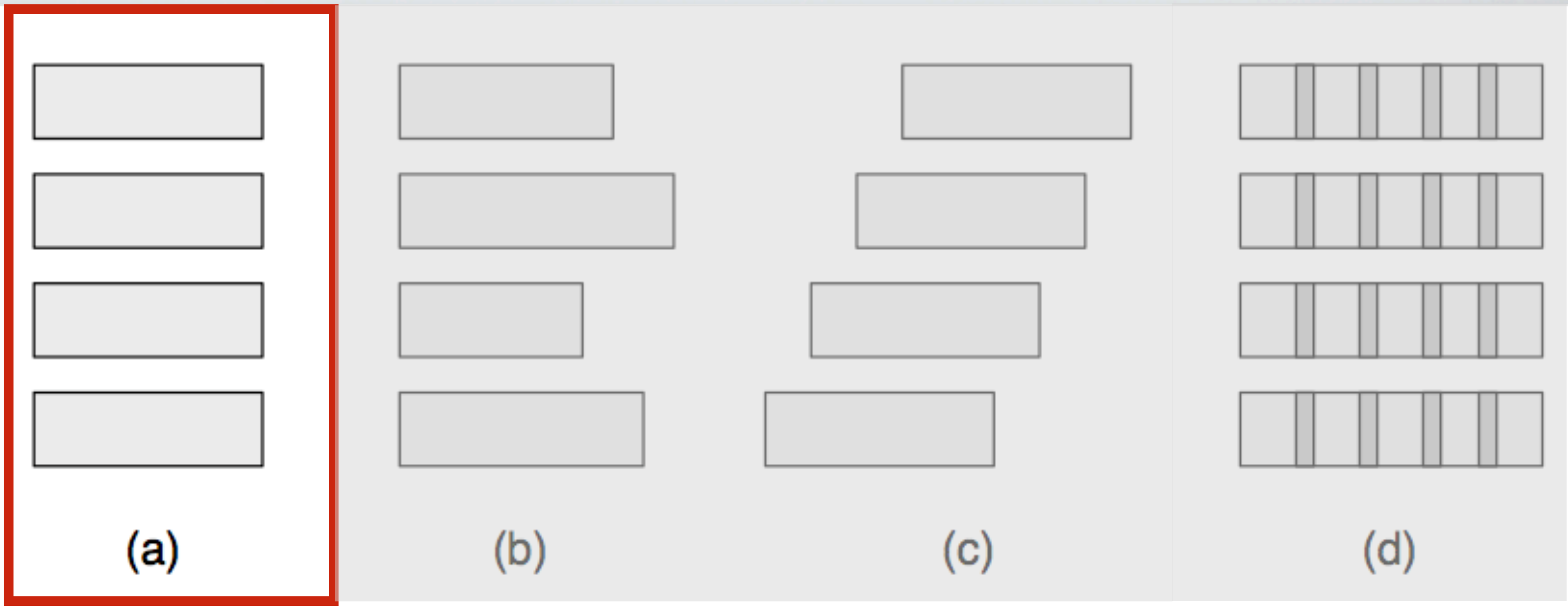
- Slides in this section taken from Heath (UIUC)

Parallel efficiency: 4 scenarios

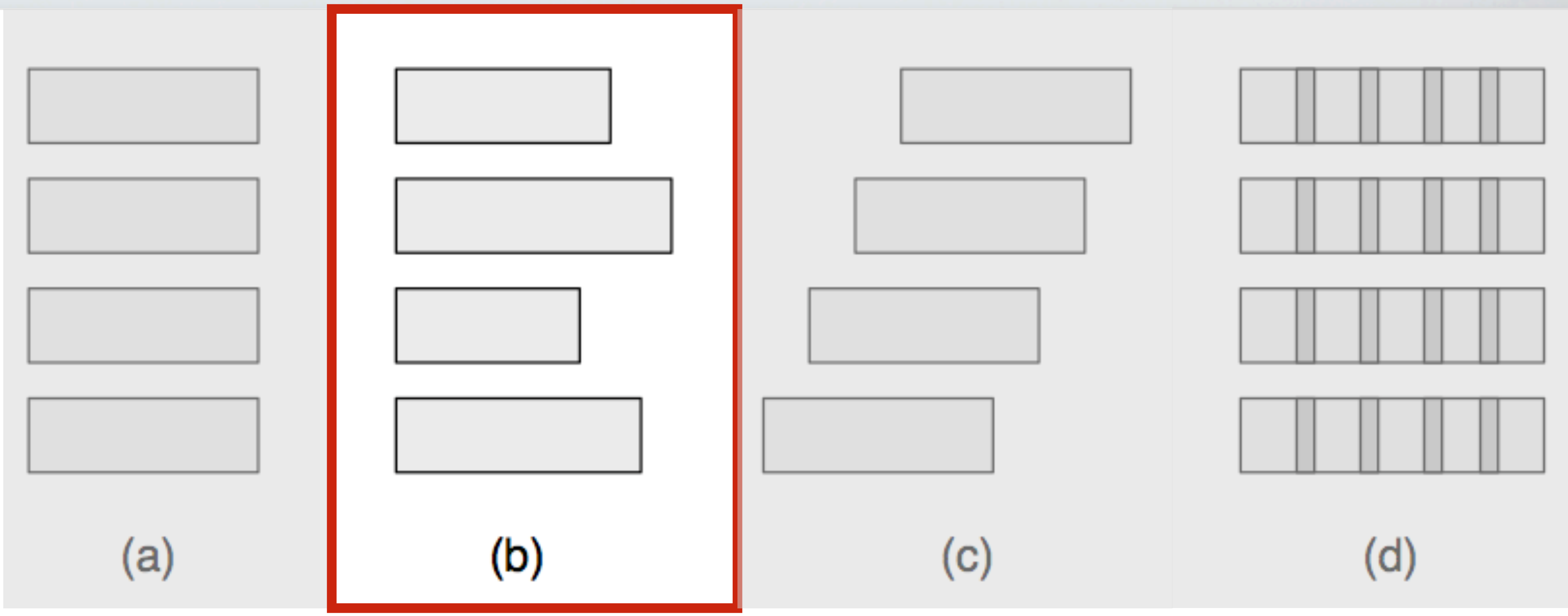


Consider **load balance**, **concurrency**, and **overhead**

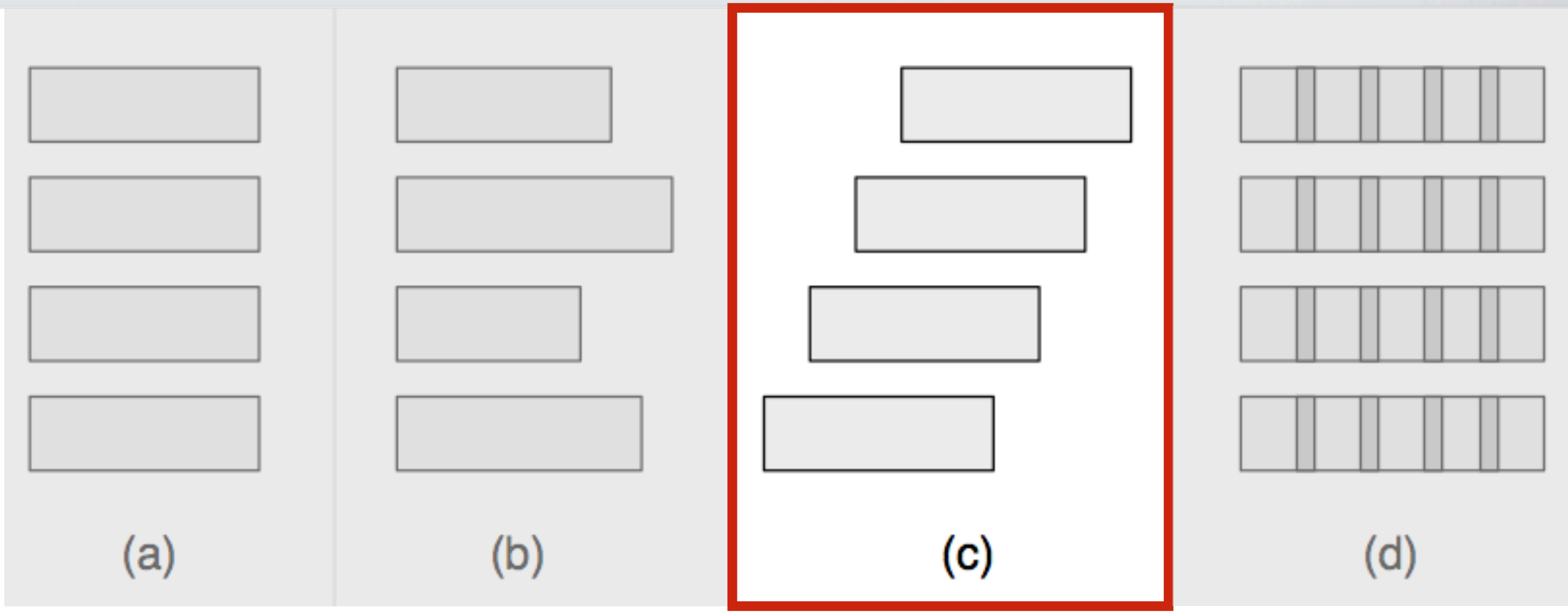
(a) Perfect load balance and concurrency



(b) Good initial concurrency but poor load balance



(c) Good load balance but poor concurrency



(d) Good load balance and concurrency, but with overheads

Basic definitions

M	**Memory complexity**	Storage for given problem (e.g., words)
W	**Computational complexity**	Amount of work for given problem (*e.g.*, flops)
V	**Processor speed**	Ops / time (*e.g.*, flop/s)
T	**Execution time**	Elapsed wallclock (*e.g.*, secs)
C	**Computational cost**	(No. procs) * (exec. time) [*e.g.*, processor-hours]

Basic definitions

M	**Memory complexity**	Storage for given problem (e.g., words)
W	**Computational complexity**	Amount of work for given problem (*e.g.*, flops)
V	**Processor speed**	Ops / time (*e.g.*, flop/s)
T	**Execution time**	Elapsed wallclock (*e.g.*, secs)
C	**Computational cost**	(No. procs) * (exec. time) [*e.g.*, processor-hours]

Subscripts denote processors used, *e.g.*, T_1 = serial time, W_p = work for p procs.

Basic definitions

M	**Memory complexity**	Storage for given problem (e.g., words)
W	**Computational complexity**	Amount of work for given problem (*e.g.*, flops)
V	**Processor speed**	Ops / time (*e.g.*, flop/s)
T	**Execution time**	Elapsed wallclock (*e.g.*, secs)
C	**Computational cost**	(No. procs) * (exec. time) [*e.g.*, processor-hours]

Assumptions: $M_p \geq M_1, W_p \geq W_1$

Quantities may be functions of one another

- Consider: W(M) to indicate that work depends on memory complexity.
- Example: Multiplying two n x n matrices

$$M = O(n^2), W = O(n^3) \implies W = O(M^{\frac{3}{2}})$$

Comments on processor speed, V

- Processor speed will depend on M due to memory hierarchies

$$V(M) \stackrel{?}{=} V(N) \quad ; \quad V\left(\frac{M}{p}\right) \stackrel{?}{\geq} V(M)$$

- Homogeneous vs. heterogeneous processors

$$V_p(M) \stackrel{?}{=} V_1(M)$$

- Aggregate speed

$$p \cdot V\left(\frac{M}{p}\right)$$

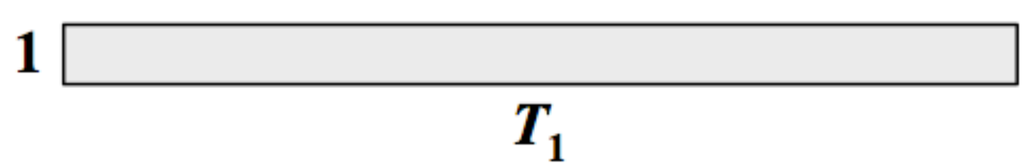
Execution time vs. cost

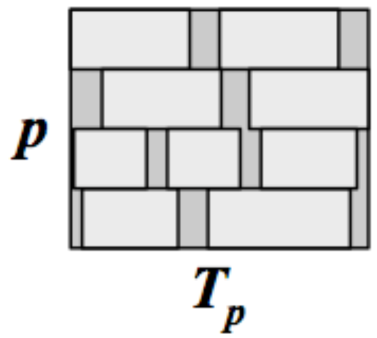
- Serial and parallel execution **time**: Work / Speed

$$T_1 = \frac{W_1}{V(M)} \qquad T_p = \frac{W_p}{p \cdot V\left(\frac{M}{p}\right)}$$

- **Cost** = (no. procs) * (execution time)

$$C_1 \equiv T_1 \qquad C_p \equiv p \cdot T_p = \frac{W_p}{V\left(\frac{M}{p}\right)}$$





Efficiency and speedup

- **Efficiency**

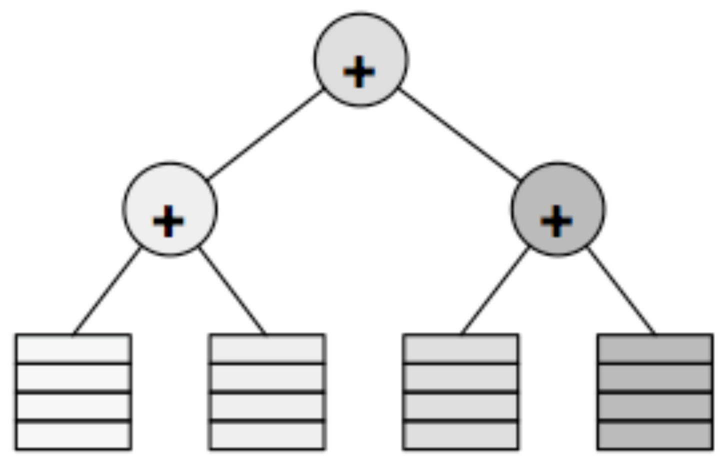
$$E_p \equiv \frac{C_1}{C_p} = \frac{T_1}{p \cdot T_p} = \frac{W_1}{W_p} \cdot \frac{V(M/p)}{V(M)}$$

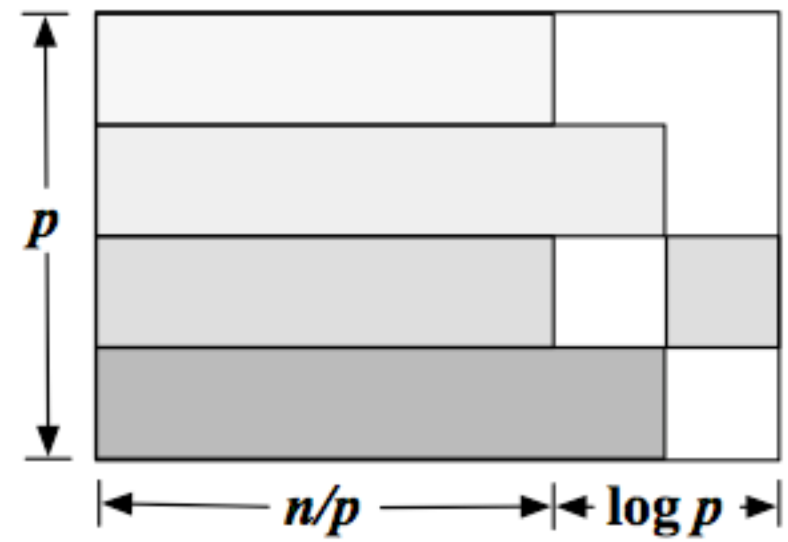
- **Speedup**

$$S_p \equiv \frac{T_1}{T_p} = p \cdot E_p$$

- **Question:** When might superlinear speedup occur?

Example: Summation using a tree algorithm

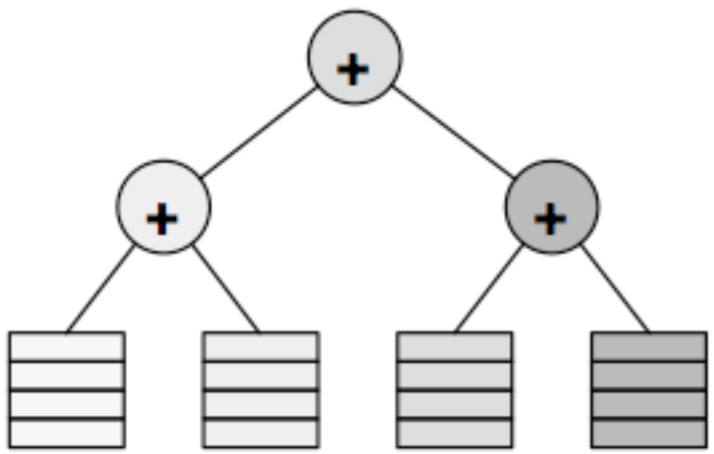


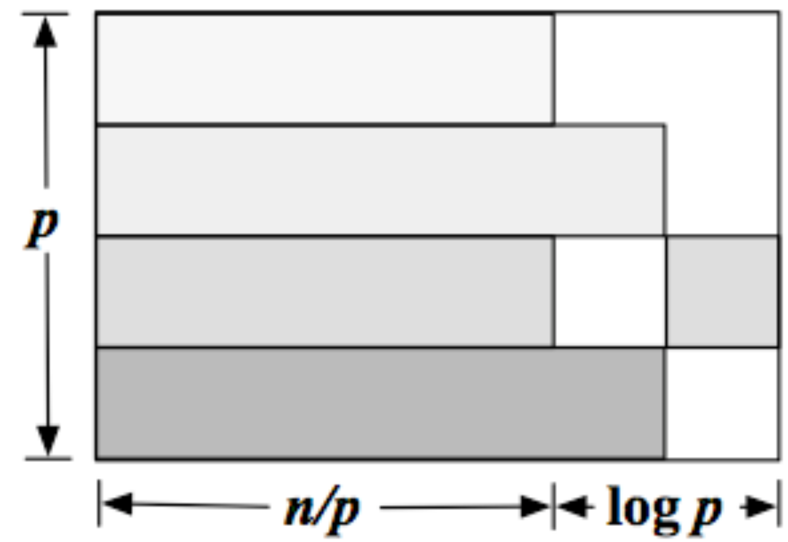


- **Memory** usage is the same

$$M_1 = M_p = n$$

Example: Summation using a tree algorithm

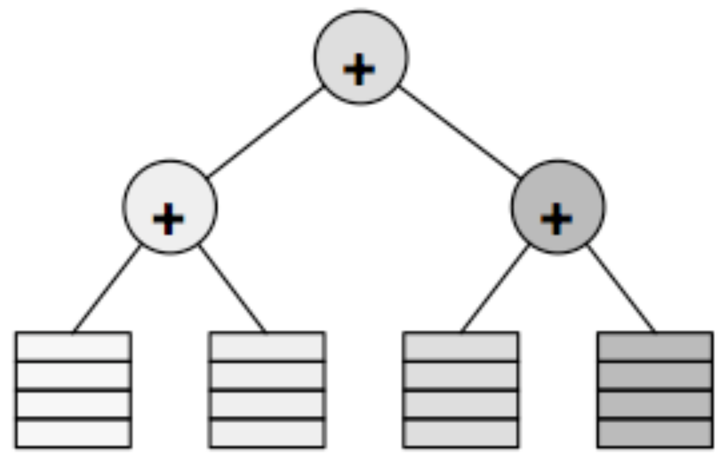


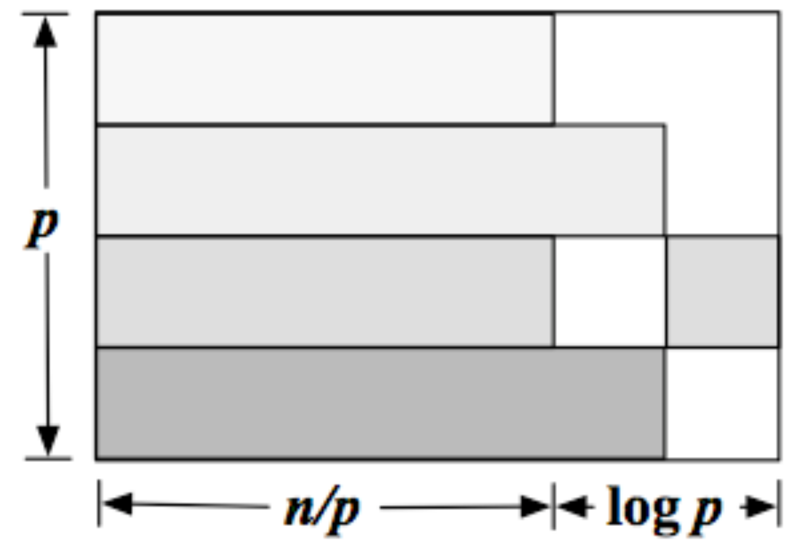


- **Work**: parallel case does more for intermediate sums

$$W_1 \approx n \qquad W_p \approx n + p \log p$$

Example: Summation using a tree algorithm

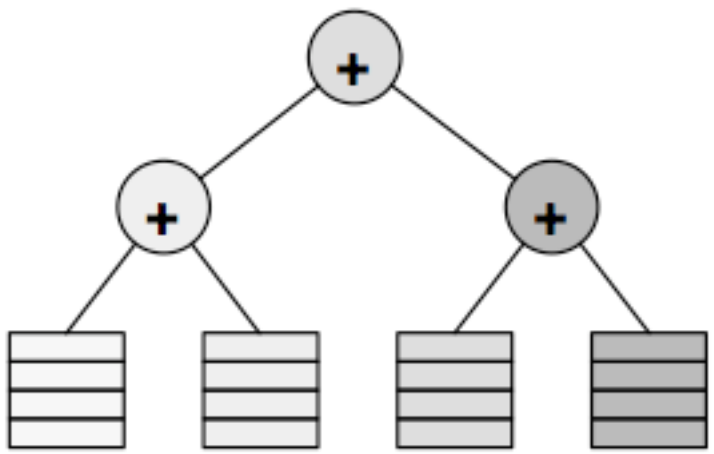


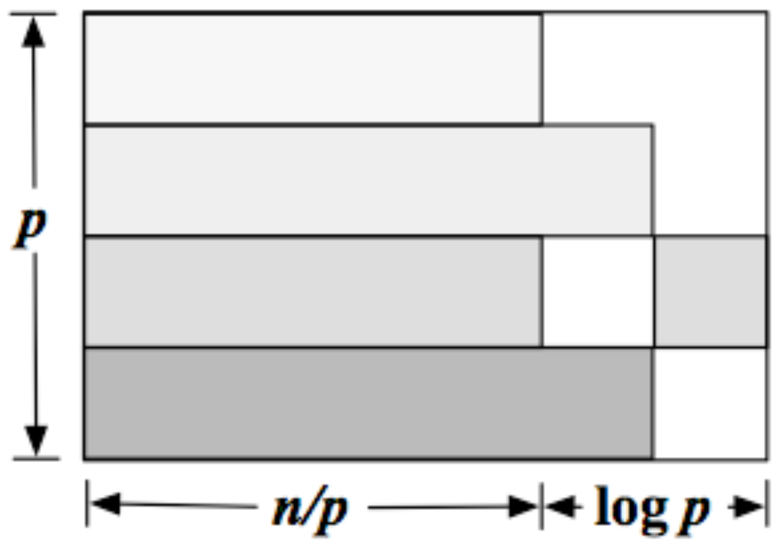


- **Time**: Assume perfect load balance & concurrency

$$T_1 \approx n \qquad T_p \approx \frac{n}{p} + \log p$$

Example: Summation using a tree algorithm

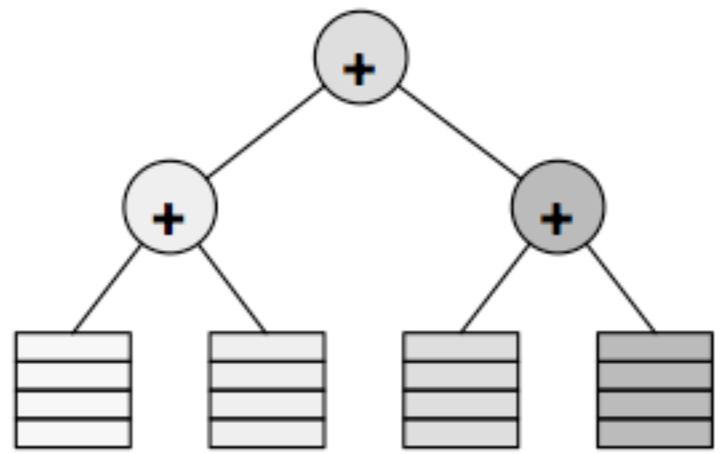


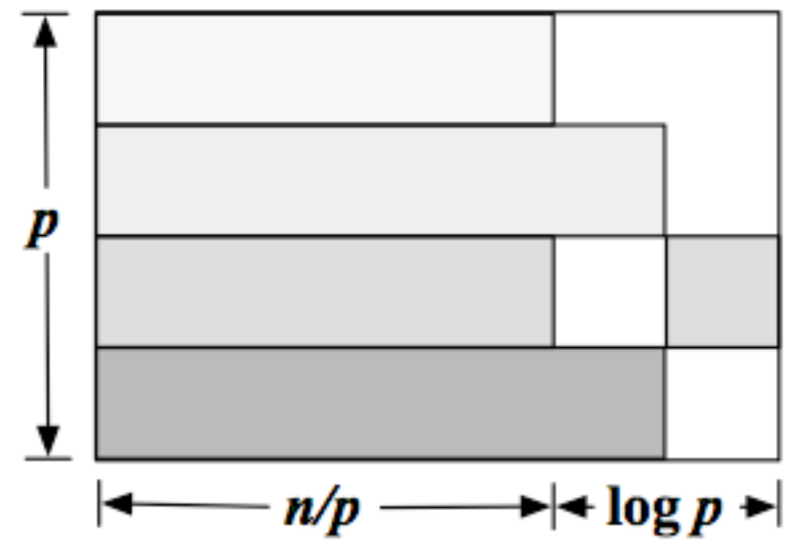


- **Cost**: (no. procs) * (time)

$$C_1 \approx n \qquad C_p \approx n + p\log p$$

Example: Summation using a tree algorithm

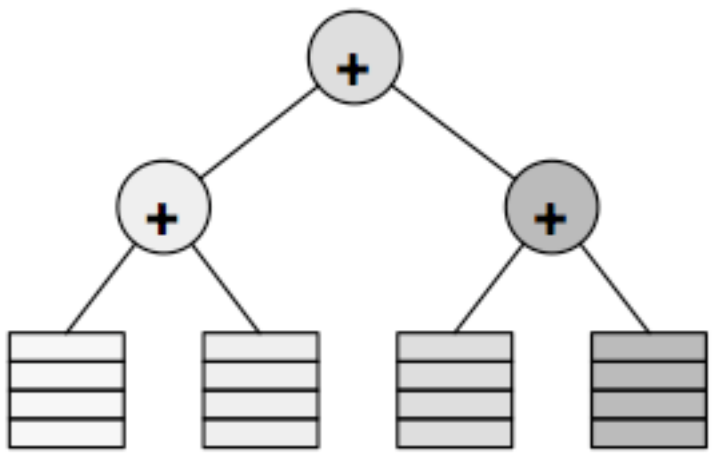


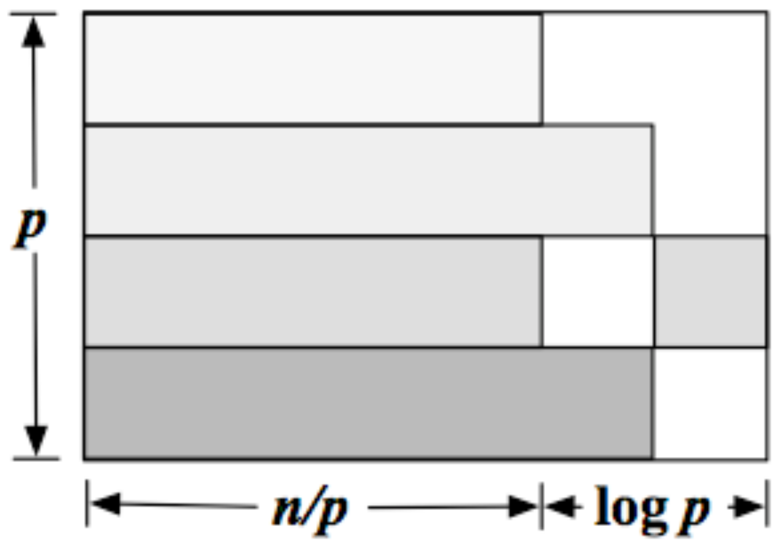


- **Efficiency**

$$E_p \equiv \frac{C_1}{C_p} \approx \frac{n}{n + p\log p} = \frac{1}{1 + \frac{p}{n}\log p}$$

Example: Summation using a tree algorithm





- **Speedup**

$$S_p \equiv \frac{T_1}{T_p} \approx \frac{n}{\frac{n}{p} + \log p} = \frac{p}{1 + \frac{p}{n}\log p}$$

Parallel scalability

- Algorithm is **scalable** if

$$E_p = \Theta(1) \text{ as } p \to \infty$$

- Why use more processors?
  - Solve fixed problem in less time
  - Solve larger problem in same time (or any time)
  - Obtain sufficient aggregate memory
  - Tolerate latency and/or use all available bandwidth (Little's Law)

Problem scaling: Fixed serial work

- More processors eventually hits diminishing returns
- Summation algorithm is **not** scalable in fixed work case

$$E_p \equiv \frac{C_1}{C_p} \approx \frac{n}{n + p \log p} = \frac{1}{1 + \frac{p}{n} \log p}$$

Problem scaling: Fixed execution time

- Applies when a strict time limit applies

$$T_1 = \frac{W_1}{V(M)} \qquad T_p = \frac{W_p}{p \cdot V\left(\frac{M}{p}\right)}$$

- Algorithm scales only if work scales linearly with p
- Summation algorithm does not scale in this scenario

$$T_1 \approx n \qquad T_p \approx \frac{n}{p} + \log p$$

Problem scaling: Scaled speedup

- **Fixed work per processor**

$$T_1 = \frac{W_1}{V(M)} \qquad T_p = \frac{W_p}{p \cdot V\left(\frac{M}{p}\right)}$$

- Summation algorithm does not scale in this scenario

$$E_p \propto \frac{W_1}{W_p} \approx \frac{pn}{pn + p\log p} = \frac{1}{1 + \frac{\log p}{n}} \longrightarrow 0$$

Problem scaling

- Fixed memory per processor
- Fixed accuracy
- **Fixed efficiency (isoefficiency) ⇒ next time**

“In conclusion…”