Case study: adding n numbers

$\text{Cost} = pT_p \propto K T_s$ for cost-efficiency $\quad T_o = pT_p - T_s$

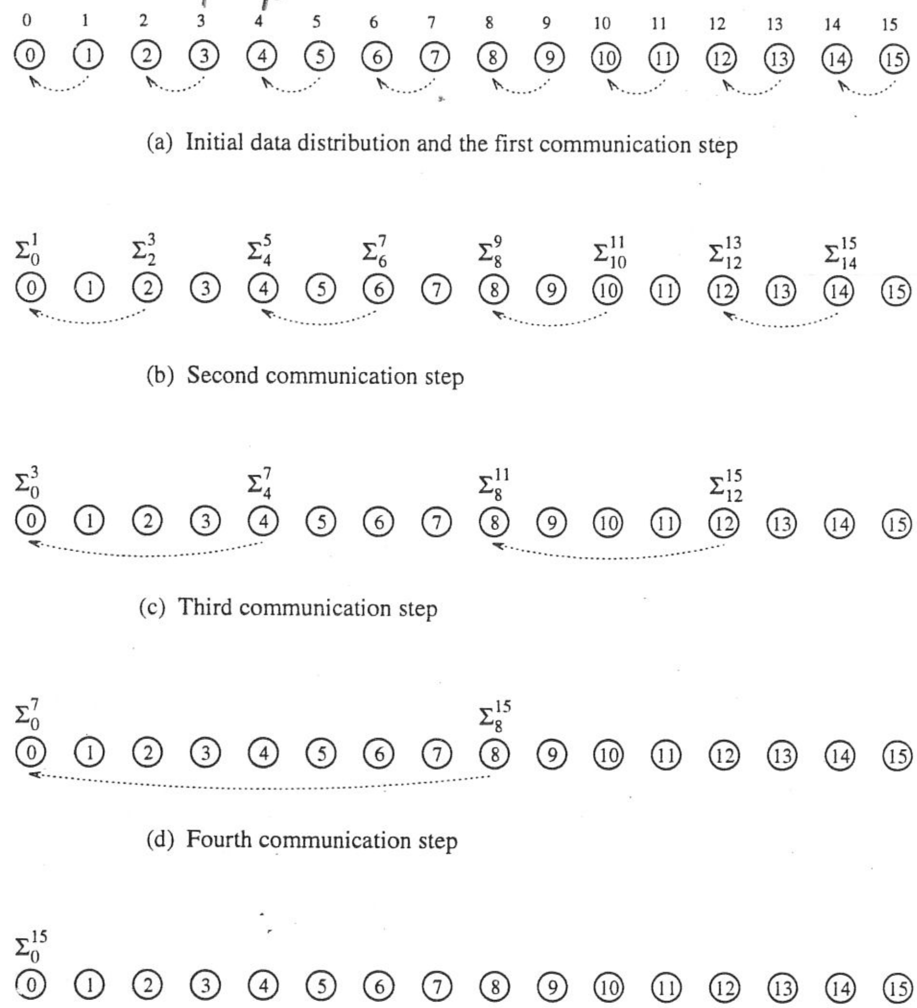
$p \downarrow$ Cases $\downarrow$	T_s	T_p	Cost pT_p	Speedup $\frac{T_s}{T_p}$	Efficiency $\frac{T_s}{p \cdot T_p}$
A) $p=1$	n	n	n	1	1
B) $p=n$	n	$\log n$	$n \log n$	$\frac{n}{\log n}$	$\frac{1}{\log n}$
C) $p<n$ blind map	n	$\frac{n}{p} \log n$	$n \log n$	$\frac{p}{\log n}$	$\frac{1}{\log n}$
D) $p<n$ $x \rightarrow x \bmod p$	n	$\frac{n}{p}\log p + \frac{n}{p}$ $= O\left(\frac{n}{p} \log p\right)$	$n \log p$	$\frac{p}{\log p}$	$\frac{1}{\log p}$
E) $p<n$ $x \rightarrow \left\lfloor \frac{x}{p} \right\rfloor$	n	$\log p + \frac{n}{p}$	$n + p\log p$	$\frac{np}{p \log p + n}$	$\frac{n}{p \log p + n}$

$\hookrightarrow$ if $\Theta(n) \stackrel{?}{=} \Theta(n + p \log p)$

$\Theta(n) = \Theta(p \log p)$

PERFORMANCE & SCALABILITY

- parallel system ≡ algorithm + architecture
- Speedup $S = \dfrac{T_s \rightsquigarrow ?}{T_p \rightsquigarrow ?}$
- Superlinear speedup ?
- Efficiency $E = S/p = \dfrac{T_s}{p \cdot T_p}$
- $\text{Cost} = p \cdot T_p$
- Cost-optimal if $\text{Cost} \propto T_s$, $E = \Theta(1)$



(a) Initial data distribution and the first communication step

(b) Second communication step

(c) Third communication step

(d) Fourth communication step

(e) Accumulation of the sum at processor 0 after the final communication

Figure 4.1 Computing the sum of 16 numbers on a 16-processor hypercube. Σ_i^j denotes the sum of numbers with consecutive labels from i to j.

- $S = \Theta\left(\dfrac{n}{\log n}\right)$
- $E = \Theta\left(\dfrac{1}{\log n}\right)$
- $\text{Cost} = \Theta(n \log n)$
- not cost optimal

- Examine effect of granularity and data mapping:
  "if system with n procs is cost-optimal, using p procs $(p < n)$ to simulate n procs preserves cost-optimality"

$T_p = \#\text{transfers} \times \#\text{steps}$

- $\underbrace{\Theta\left(\frac{n}{p}\log p\right)}_{\text{step 1}} + \underbrace{\Theta\left(\frac{n}{p}\right)}_{\text{step 2}}^{\#\text{adds}} = \Theta\left(\frac{n}{p}\log p\right)$ ∴ not cost-optimal
- Overhead $T_o = \Theta(n \log p)$
- Problem size W to add n numbers is $\Theta(n)$

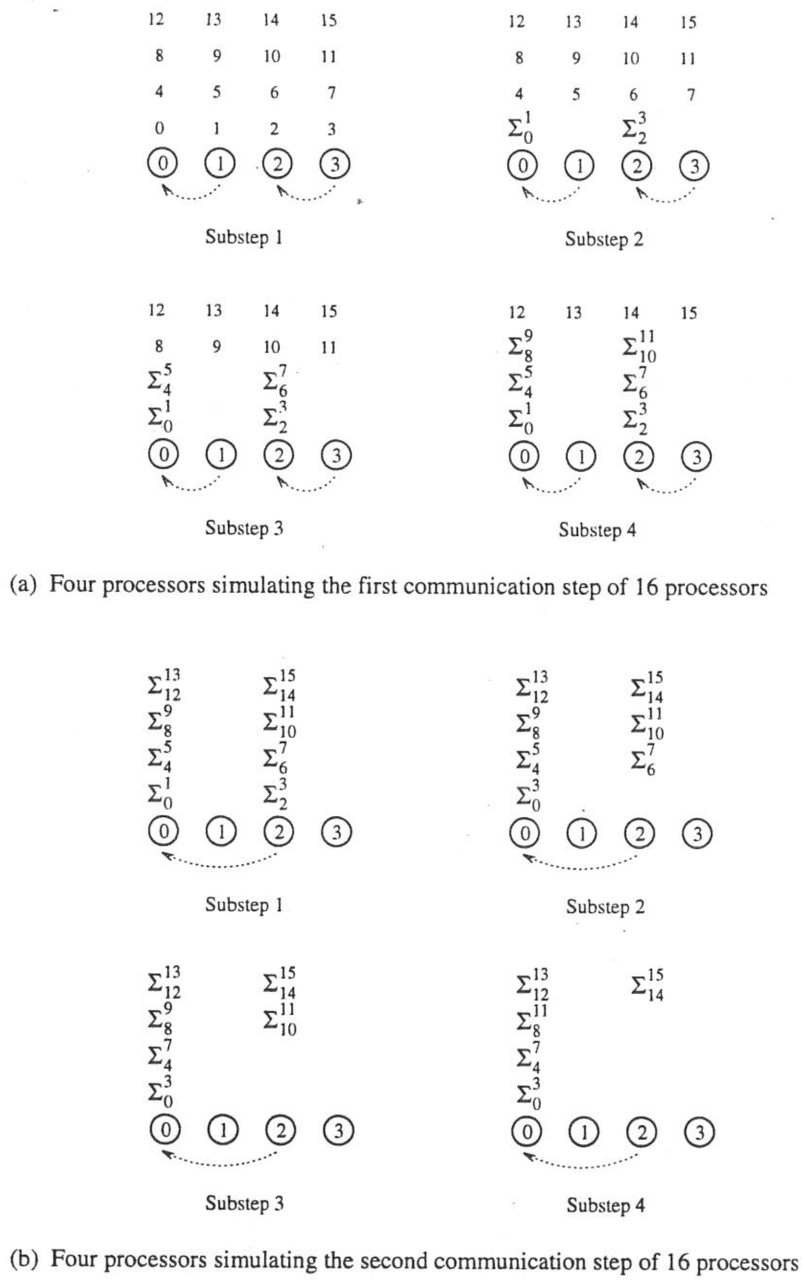
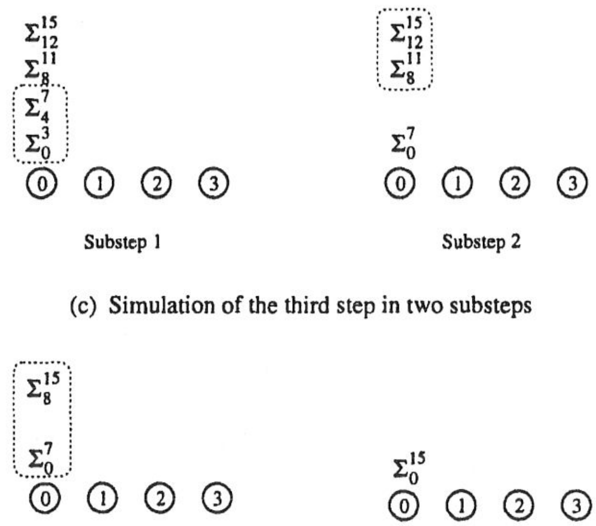


Figure 4.2 Four processors simulating 16 processors to compute the sum of 16 numbers (first two steps). Σ_i^j denotes the sum of numbers with consecutive labels from i to j.

- Isoefficiency function does not exist because $[W = KT_o(W, p)]$ cannot be satisfied for any K ∴ system not scalable



(c) Simulation of the third step in two substeps

(d) Simulation of the fourth step

(e) Final result

Figure 4.3 (cont.) Four processors simulating 16 processors to compute the sum of 16 numbers (last three steps).

- $\Theta(n/p) + \Theta(\log p) = T_p$ asymptotics
- $Cost = \Theta(n + p\log p)$ = cost-optimal as long as $n = \Omega(p \log p)$ constants

---

- $T_p = \frac{n}{p} - 1 + 2\log p$ • $S = \frac{n}{n/p + 2\log p} = \frac{np}{n + 2p\log p}$
- $E = \frac{n}{n + 2p\log p}$

- $T_o(W,p) = p\left(\frac{n}{p} + 2\log p\right) - n = 2p\log p$
- (isoefficiency func): $W = 2Kp\log p$ ie, $\Theta(p\log p)$
- $W \approx n$ & $T_o = \Theta(p\log p)$ & condn. for cost-optimality: $W = \Omega(p\log p)$

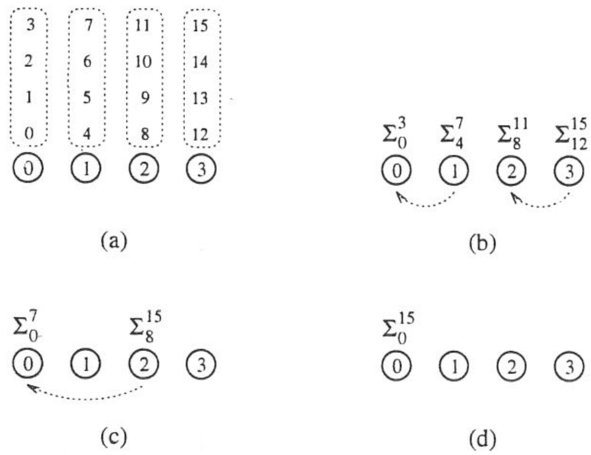


Figure 4.4 A cost-optimal way of computing the sum of 16 numbers on a four-processor hypercube. Copyright (r) 1994 Benjamin/Cummings Publishing Co.

$W \propto p \log p$

$W' \propto p' \log p'$

$\therefore$ Increasing p by factor $\frac{p'}{p}$ requires increasing workload by factor $\frac{p' \log p'}{p \log p}$ to increase speedup by factor p'/p

---

$E = \frac{T_s}{pT_p}$; $T_o = pT_p - T_s$ $\therefore E = \frac{1}{1 + \frac{T_o}{T_s}}$

T_o: $\uparrow$ fn of p. Pgm has serial component (T_{serial}), communication, idling etc.

$T_o \propto (p-1)T_{serial}$

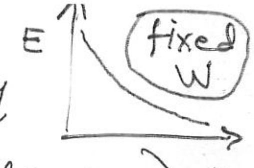
$\therefore$ For fixed problem size (ie, fixed T_s), as $p\uparrow$, $T_o\uparrow$ and $E\downarrow$ —— (A)

$\uparrow n$, keeping p constant: In many cases, T_o grows sublinearly w.r.t. T_s — (B)

$\Rightarrow E\uparrow$

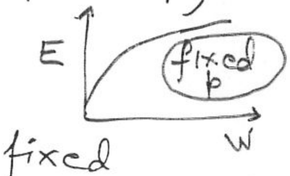
From (A) & (B), possible to keep efficiency fixed, by $\uparrow n$ & $\uparrow p$ --- ∴ "scalable"

Q] Rate at which problem size $n\uparrow$ w.r.t. p, to keep E fixed?



- ① Speedup does not ↑ linearly as # procs increases ∴ E ↓
- ② larger instance of same problem yields ↑ S & E (for fixed p)

→ Very common trends



- Scalable parallel system ≡ (↑ p & problem size) to keep E fixed
- Scalability ≡ measure of capacity to ↑ S in proportion to p
- $E = \Theta(1)$ for cost-optimal system
- Scalability & cost-optimality are related: scalable system can always be made cost-optimal if p and (size of computation) chosen correctly.

eg below: system stays cost-optimal at $E = 0.8$ if n is increased as $p \log p$

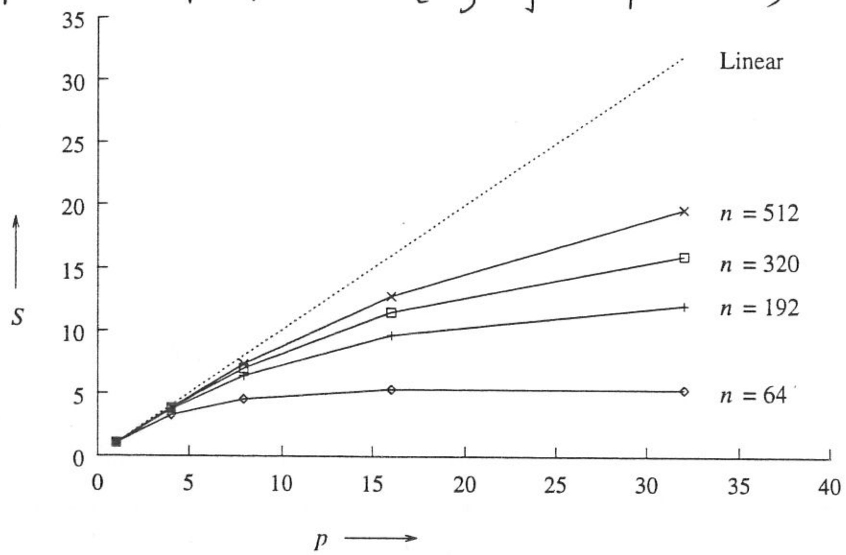


Figure 4.5 Speedup versus the number of processors for adding a list of numbers on a hypercube.

n	p=1	p=4	p=8	p=16	p=32
64	1	.8	.57	.33	.17
192	1	.92	.8	.6	.38
320	1	.95	.87	.71	.5
512	1	.97	.91	.8	.62

[Efficiency as a function of n & p for Fig 4.4]

- Cost-optimal when $n = \Omega(p \log p)$. For $\begin{bmatrix} n=64 \\ p=4 \end{bmatrix}$, $n = 8\, p \log p$

$64 = 8 \times [4 \log 4]$ $\quad 512 = 8\,[16 \log 16]$

$192 = 8 \times [8 \log 8]$

- performance of scaled down algo may be different for different assignments of (virtual → real) processors

eg $\begin{bmatrix} n \times n \end{bmatrix} \times \begin{bmatrix} n \times 1 \end{bmatrix}$ on p-proc HC → p square blocks / → p stripes of n/p cols/rows

ISOEFFICIENCY METRIC OF SCALABILITY

- useful to determine rate at which problem size must ↑ w.r.t. p to keep the efficiency fixed

Defn:

- Problem size W = # computation steps in best seq. algo. on 1 proc.
- Overhead $T_o(W, p) = pT_p - W$ // $W = T_s$

// T_o is ↑ing function of p

$$T_p = (T_o + W)/p$$

$$S = W/T_p = \frac{Wp}{T_o + W}$$

$$E = \frac{S}{p} = \frac{W}{W + T_o} = \frac{1}{1 + T_o(W,p)/W}$$

. Examine effect on E!

Typically, T_o grows slower than $\Theta(W)$ for a fixed p // observation

$$W = \left(\frac{E}{1-E}\right) T_o(W,p) = K T_o(W,p)$$ -------- isoefficiency func.

- func determines ease with which parallel system can maintain const. efficiency & hence achieve speedups increasing in proportion to p
- small ⇒ small ↑ in W sufficient to efficiently utilize ↑# procs
- large ⇒ poorly scalable
- T_o may have terms of different orders of magnitude
  component that requires W to grow at the highest rate w.r.t. p determines overall asymptotic isoefficiency function

eg $T_o = p^{3/2} + p^{3/4} W^{3/4}$ — $W = K p^{3/2}$

$W = K p^{3/4} W^{3/4} \Rightarrow W = K^4 p^3$

∴ $\Theta(p^{3/2})$ 1st term & $\Theta(p^3)$ 2nd term

- Parallel system is cost-optimal iff $p \cdot T_p = \Theta(W)$

  ie, $W + T_o(W, p) = \Theta(W)$ // $T_o = pT_p - W$

  $$\left.\begin{array}{l} T_o(W,p) = O(W) \\ W = \Omega(T_o(W,p)) \end{array}\right\}$$ ie iff T_o does not asymptotically exceed W

- If $W = KT_o(W,p)$ gives isoefficiency function $f(p)$, then $W = \Omega(f(p))$ must hold to ensure cost-optimality with scaling

- LOWER BOUND ON ISOEFFICIENCY FUNCTION

→ Asymptotically, W must increase at least as fast as $\Theta(p)$ to maintain fixed efficiency

∴ $\Omega(p)$ is asymptotic lower bound on isoefficiency fn.

Ideally, $\Theta(p)$.

- DEGREE OF CONCURRENCY & ISOEFFICIENCY FUNCTION

→ measure of # ops in parallel as a function of W

→ eg solve n eqns in n variables using Gaussian elimination

Computation = $\Theta(n^3)$ totally, but

n vars are sequentially eliminated, each needs $\Theta(n^2)$ computations

$W = \Theta(n^3)$, $C(W) = \Theta(W^{2/3})$

Given p processors, problem size $W \geq \Omega(p^{3/2})$ to use all procs

→ Isoefficiency func (due to concurrency) is optimal, ie $\Theta(p)$ only if $C(W) = \Theta(W)$

If $C(W) < \Theta(W)$, isoefficiency fn (due to concurrency) $> \Theta(p)$

- SOURCES OF PARALLEL OVHD T_o: ① Interprocessor communication

② Load imbalance → cannot predict / need to sync/seq. components

③ Extra computation → fastest seq. algo may to hard to parallelize

↳ (serial:) reuse results, (parallel:) cannot reuse, as generated by different processors ∴ extra computation eg. FFT

- performance of scaled down algo. may be different for different assignments of (virtual → real) processors

  eg $[n \times n][\ ]$ on p-proc HC → p square blocks / → p stripes of n/p rows

ISOEFFICIENCY METRIC OF SCALABILITY

- useful to determine rate at which problem size must increase w.r.t. #p to keep the efficiency fixed

- Problem size (W) ≡ # computation steps in best seq. algo on 1 proc.

- Overhead $T_o(W, p) = pT_p - W$

$$T_o = pT_p - T_s$$

- Problem size : regardless of problem, doubling size ⇒ 2 times the computation.

$$\text{For matrix mult, } W = \theta(n^3)$$
$$\text{matrix addn, } W = \theta(n^2)$$

$\therefore W = T_s$

- Isoefficiency fn:
  - in 1 expr, captures characteristics of || algo & || architecture
  - test performance of || pgm on a few processors; predict performance on larger # processors] not viable
  - characterizes amt of parallelism inherent in a parallel algo
  - study behavior of || system w.r.t. changes in HW parameters eg speed of CPUs, channels
  - useful for || algos for which we cannot derive a value of || runtime