Parallel System Performance: Evaluation & Scalability

- **Factors affecting parallel system performance:**
  - **Algorithm-related, parallel program related, architecture/hardware-related.**
- **Workload-Driven Quantitative Architectural Evaluation:**
  - **Select applications or suite of benchmarks to evaluate architecture either on real or simulated machine.**
  - **From measured performance results compute performance metrics:**
    - **Speedup, System Efficiency, Redundancy, Utilization, Quality of Parallelism.**
  - **Resource-oriented Workload scaling models: How the speedup of an application is affected subject to specific constraints:**
    1. ***Problem constrained* (PC): Fixed-load Model.**
    2. ***Time constrained* (TC): Fixed-time Model.**
    3. ***Memory constrained* (MC): Fixed-Memory Model.**
- **Performance Scalability:**
  - **Definition.**
  - **Conditions of scalability.**
  - **Factors affecting scalability.**

For a given parallel system and parallel problem/algorithm:

Informally:
The ability of parallel system performance to increase with increased problem and system size.

Parallel Computer Architecture, Chapter 4
Parallel Programming, Chapter 1, handout

Parallel Program Performance

- **Parallel processing goal is to maximize speedup:**

$$\text{Speedup} = \frac{\text{Time(1)}}{\text{Time(p)}} \leq \frac{\text{Sequential Work}}{\textit{Max}\ (\text{Work} + \text{Synch Wait Time} + \text{Comm Cost} + \text{Extra Work})}$$

Max for any processor

Parallelizing Overheads

- **By:**
  - 1 – **Balancing computations/overheads (workload) on processors (every processor has the same amount of work/overheads).**
  - 2 – **Minimizing communication cost and other overheads associated with each step of parallel program creation and execution.**

Parallel Performance Scalability: For a given parallel system and parallel computation/problem/algorithm

Achieve a good speedup for the parallel application on the parallel architecture as problem size and machine size (number of processors) are increased.

Or

Continue to achieve good parallel performance "speedup"as the sizes of the system/problem are increased.

(More formal treatment of scalability later)

Factors affecting Parallel System Performance

- **Parallel Algorithm-related:**

i.e Inherent Parallelism

  - Available concurrency and profile, dependency graph, uniformity, patterns.
  - Complexity and predictability of computational requirements
  - Required communication/synchronization, uniformity and patterns.
  - Data size requirements.

- **Parallel program related:**
  - Partitioning: Decomposition and assignment to tasks
    - Parallel task grain size.
    - Communication to computation ratio.

C-to-C ratio (measure of inherent communication)

  - Programming model used.
  - Orchestration
    - Cost of communication/synchronization.
  - Resulting data/code memory requirements, locality and working set characteristics.
  - Mapping & Scheduling: Dynamic or static.

- **Hardware/Architecture related:**
  - Total CPU computational power available.
  - Parallel programming model support:
    - e.g support for Shared address space Vs. message passing support.
    - Architectural interactions, artifactual "extra" communication
  - Communication network characteristics: Scalability, topology ..
  - Memory hierarchy properties.

Refined from factors in Lecture # 1

Parallel Performance Metrics Revisited

- **Degree of Parallelism (DOP): For a given time period, reflects the number of processors in a specific parallel computer actually executing a particular parallel program.**
- **Average Parallelism, A:**
  - **Given maximum parallelism = *m***
  - ***n* homogeneous processors**
  - **Computing capacity of a single processor Δ** ← **Computations/sec**
  - **Total amount of work (instructions or computations):**

i.e DOP at a given time = Min (Software Parallelism, Hardware Parallelism)

$$W = \Delta \int_{t1}^{t2} DOP(t)dt$$ **or as a discrete summation** $$W = \Delta \sum_{i=1}^{m} i.t_i$$

Where t_i is the total time that DOP = *i* and $\sum_{i=1}^{m} t_i = t_2 - t_1$ ← **Execution Time**

The average parallelism A:

$$A = \frac{1}{t_2 - t_1} \int_{t1}^{t2} DOP(t)dt$$

Execution Time

In discrete form

$$A = \left(\sum_{i=1}^{m} i.t_i \right) \Big/ \left(\sum_{i=1}^{m} t_i \right)$$

DOP Area

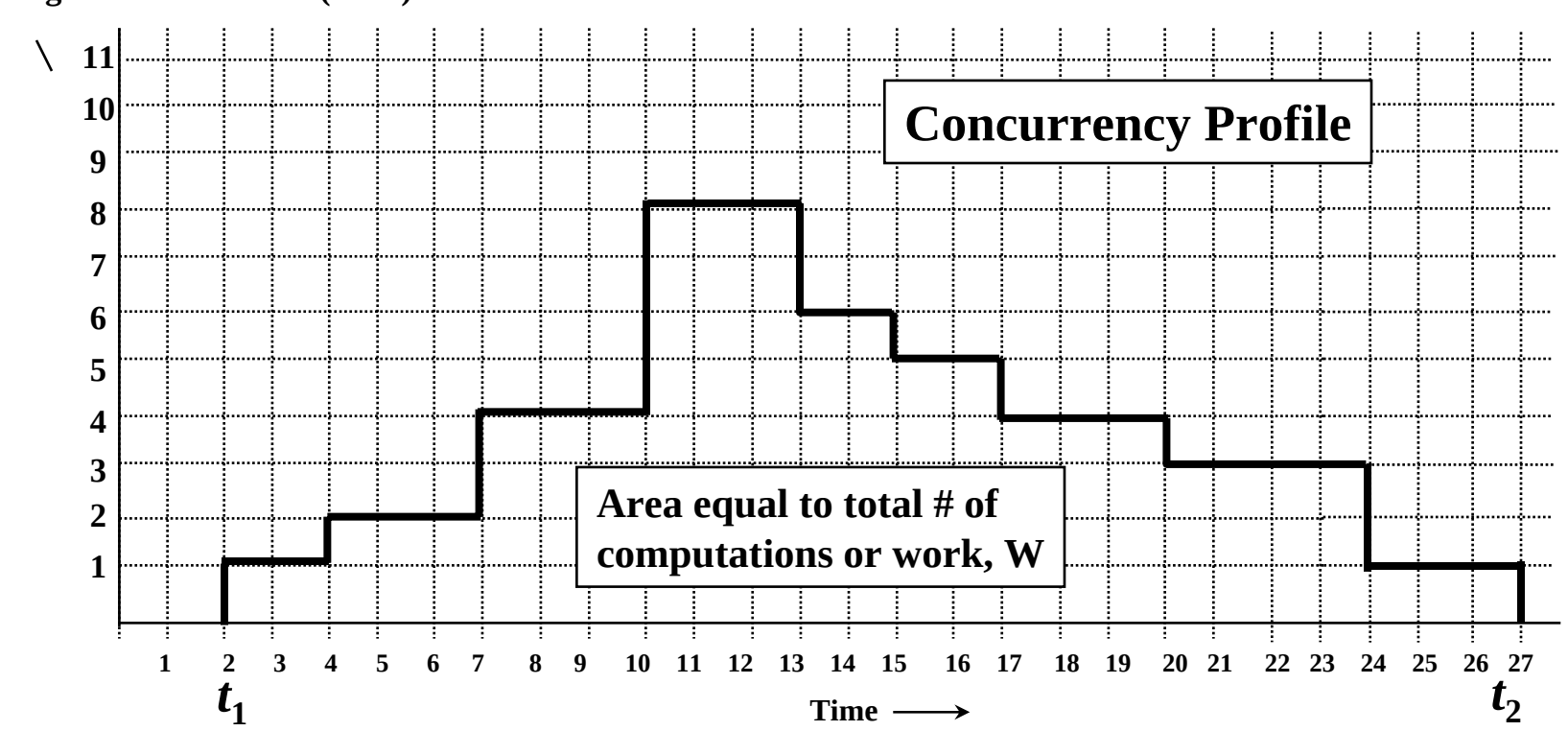
From Lecture # 3

Example: Concurrency Profile of A Divide-and-Conquer Algorithm

- Execution observed from $t_1 = 2$ to $t_2 = 27$
- Peak parallelism $m = 8$
- A = (1x5 + 2x3 + 3x4 + 4x6 + 5x2 + 6x2 + 8x3) / (5 + 3+4+6+2+2+3)
  = 93/25 = 3.72

$$A = \left(\sum_{i=1}^{m} i.t_i\right) \Big/ \left(\sum_{i=1}^{m} t_i\right)$$

Average Parallelism



From Lecture # 3

Parallel Performance Metrics Revisited

Asymptotic Speedup:

i.e. Hardware Parallelism > Software Parallelism

(more processors n than max software DOP, m)

Execution time with one processor

$$T(1) = \sum_{i=1}^{m} t_i(1) = \sum_{i=1}^{m} \frac{W_i}{\Delta}$$

Execution time with an infinite number of available processors (number of processors n = ∞ or n > m)

$$T(\infty) = \sum_{i=1}^{m} t_i(\infty) = \sum_{i=1}^{m} \frac{W_i}{i\Delta}$$

Asymptotic speedup S_∞

$$S_\infty = \frac{T(1)}{T(\infty)} = \frac{\sum_{i=1}^{m} W_i}{\sum_{i=1}^{m} W_i / i}$$

The above ignores all overheads.

Δ = Computing capacity of a single processor
m = maximum degree of software parallelism
t_i = total time that DOP = *i*
W_i = total work with DOP = *i*

Keeping problem size fixed and ignoring parallelization overheads/extra work

i.e. Hardware parallelism n exceeds software parallelism m

Phase Parallel Model of An Application

- **Consider a sequential program of size s consisting of k computational phases C_1 …. C_k where each phase C_i has a degree of parallelism DOP = i**
- **Assume single processor execution time of phase C_i = $T_1(i)$**
- **Total single processor execution time =** $$T_1 = \sum_{i=1}^{i=k} T_1(i)$$ **k = max. DOP**
- **Ignoring overheads, n processor execution time:** $$T_n = \sum_{i=1}^{i=k} T_1(i) / \min(i, n)$$
- **If all overheads are grouped as interaction $T_{interact}$** = Synch Time + Comm Cost **and parallelism T_{par}** = Extra Work**, as h(s, n) = $T_{interact}$ + T_{par} then parallel execution time:**

Accounting for parallelization overheads

$$T_n = \sum_{i=1}^{i=k} T_1(i) / \min(i, n) + h(s, n)$$

Total overheads

n = number of processors

s = problem size

- **If k = n and f_i is the fraction of sequential execution time with DOP =i $\pi = \{f_i | i = 1, 2, \ldots, n\}$ and ignoring overheads (h(s, n) = 0) the speedup is given by:**

$$S(n) = S(\infty) = \frac{T_1}{T_n} = \frac{1}{\left(\sum_{i=1}^{n} f_i / i\right)}$$

$\pi = \{f_i | i = 1, 2, \ldots, n\}$ for max DOP = n
is parallelism degree probability distribution (DOP profile)

Harmonic Mean Speedup for *n* Execution Mode Multiprocessor system

Fig 3.2 page 111
See handout

Parallel Performance Metrics Revisited: Amdahl's Law

- **Harmonic Mean Speedup** (*i* number of processors used f_i is the fraction of sequential execution time with DOP =i):

$$S(n) = T_1 / T_n = \frac{1}{\left(\sum_{i=1}^{n} f_i / i\right)}$$

DOP =1 (sequential)

DOP =n

- In the case $\pi = \{f_i \text{ for } i = 1, 2, .., n\} = (\alpha, 0, 0, \ldots, 1-\alpha)$, the system is running sequential code with probability α and utilizing *n* processors with probability $(1-\alpha)$ with other processor modes not utilized.

Amdahl's Law:

$$S_n = \frac{1}{\alpha + (1-\alpha)/n}$$

Keeping problem size fixed and ignoring overheads (i.e h(s, n) = 0)

$S \rightarrow 1/\alpha$ as $n \rightarrow \infty$

$\Rightarrow$ Under these conditions the best speedup is upper-bounded by $1/\alpha$

Alpha = α = Sequential fraction with DOP = 1

Parallel Performance Metrics Revisited

Efficiency, Utilization, Redundancy, Quality of Parallelism

- **System Efficiency:** Let $O(n)$ (i.e total parallel work on n processors) be the total number of unit operations performed by an n-processor system and $T(n)$ be the execution time in unit time steps:
  - In general $T(n) << O(n)$ (more than one operation is performed by more than one processor in unit time).
  - Assume $T(1) = O(1)$
  - **Speedup factor:** $S(n) = T(1) / T(n)$
    - Ideal $T(n) = T(1)/n$ -> Ideal speedup = n
  - **System efficiency** E(n) for an n-processor system:

n = number of processors
Here O(1) = work on one processor
O(n) = total work on n processors

$$E(n) = S(n)/n = T(1)/[nT(n)]$$

Ideally:

Ideal speedup: S(n) = n

and thus ideal efficiency: E(n) = n /n = 1

Parallel Performance Metrics Revisited
Cost, Utilization, Redundancy, Quality of Parallelism

- **Cost:** The processor-time product or cost of a computation is defined as

Speedup = T(1)/T(n) Efficiency = S(n)/n

$$Cost(n) = n\,T(n) = n \times T(1) / S(n) = T(1) / E(n)$$

  - The cost of sequential computation on one processor n=1 is simply T(1)
  - A cost-optimal parallel computation on n processors has a cost proportional to T(1) when:

$$S(n) = n,\ E(n) = 1 \quad \text{---} > \quad Cost(n) = T(1)$$

- **Redundancy:** $R(n) = O(n)/O(1)$
  - Ideally with no overheads/extra work $O(n) = O(1)$ -> $R(n) = 1$
- **Utilization:** $U(n) = R(n)E(n) = O(n) /[nT(n)]$
  - ideally $R(n) = E(n) = U(n) = 1$

Perfect load balance?

Assuming: T(1) = O(1)

- **Quality of Parallelism:**

$$Q(n) = S(n)\,E(n) / R(n) = T^3(1) /[nT^2(n)O(n)]$$

  - Ideally $S(n) = n$, $E(n) = R(n) = 1$ ---> $Q(n) = n$

n = number of processors
here: O(1) = work on one processor O(n) = total work on n processors

A Parallel Performance measures Example

For a hypothetical workload with

- $O(1) = T(1) = n^3$ Work or time on one processor
- $O(n) = n^3 + n^2 \log_2 n$ Total parallel work on n processors $T(n) = 4n^3/(n+3)$ Parallel execution time on n processors
- $\text{Cost}(n) = 4n^4/(n+3) \sim 4n^3$

Fig 3.4 page 114

Table 3.1 page 115
See handout

Application Scaling Models for Parallel Computing

- If work load *W* or problem size "*s*" is unchanged then:
  - The efficiency *E* may decrease as the machine size *n* increases if the overhead *h(s, n)* increases faster than the increase in machine size.
- The condition of a scalable parallel computer solving a scalable parallel problem exists when:
  - A desired level of efficiency is maintained by increasing the machine size "n" and problem size "s" proportionally. $E(n) = S(n)/n$
  - In the ideal case the workload curve is a linear function of *n: (Linear scalability in problem size).*
- Application Workload Scaling Models for Parallel Computing:

  Workload scales subject to a given constraint as the machine size is increased:

1. Problem constrained (PC): or Fixed-load Model. Corresponds to a constant workload or fixed problem size.
2. Time constrained (TC): or Fixed-time Model. Constant execution time.
3. Memory constrained (MC): or Fixed-memory Model: Scale problem so memory usage per processor stays fixed. Bound by memory of a single processor.

n = Number of processors s = Problem size

Problem Constrained (PC) Scaling : Fixed-Workload Speedup

Corresponds to "Normal" parallel speedup:
Keep problem size (workload) fixed as the size of the parallel machine (number of processors) is increased.

When DOP = $i > n$ (i = 1 …m) ***(n = number of processors)***

Execution time of W_i

$$t_i(n) = \frac{W_i}{i\Delta}\left\lceil \frac{i}{n} \right\rceil$$

Total execution time

$$T(n) = \sum_{i=1}^{m} \frac{W_i}{i\Delta}\left\lceil \frac{i}{n} \right\rceil$$

Ignoring parallelization overheads

If DOP = $i < n$, then $t_i(n) = t_i(\infty) = W_i / i\Delta$

Fixed-load speedup factor is defined as the ratio of *T(1)* to *T(n)*:

Ignoring overheads

$$S_n = \frac{T(1)}{T(n)} = \frac{\sum_{i=1}^{m} W_i}{\sum_{i=1}^{m} \frac{W_i}{i}\left\lceil \frac{i}{n} \right\rceil}$$

Let *h(s, n)* be the total system overheads on an *n*-processor system:

$$S_n = \frac{T(1)}{T(n) + h(s,n)} = \frac{\sum_{i=1}^{m} W_i}{\sum_{i=1}^{m} \frac{W_i}{i}\left\lceil \frac{i}{n} \right\rceil + h(s,n)}$$

Total parallelization overheads term

The overhead term *h(s,n)* is both application- and machine-dependent and usually difficult to obtain in closed form.

s = problem size n = number of processors

Amdahl's Law for Fixed-Load Speedup

- **For the special case where the system either operates in sequential mode (DOP = 1) or a perfect parallel mode (DOP = *n*), the Fixed-load speedup is simplified to:**

n = number of processors

$$S_n = \frac{W_1 + W_n}{W_1 + W_n / n}$$

i.e. ignoring parallelization overheads

We assume here that the overhead factor *h(s, n)= 0*

For the normalized case where:

$$W_1 + W_n = \alpha + (1 - \alpha) = 1 \text{ with } \alpha = W_1 \text{ and } 1 - \alpha = W_n$$

The equation is reduced to the previously seen form of Amdahl's Law:

$$S_n = \frac{1}{\alpha + (1 - \alpha)/n}$$

Alpha = α = Sequential fraction with DOP = 1

Time Constrained (TC) Workload Scaling

Fixed-Time Speedup

Both problem size (workload) and machine size are scaled (increased) so execution time remains constant.

- **To run the largest problem size possible on a larger machine with about the same execution time of the original problem on a single processor.**

Let m' be the maximum DOP for the scaled up problem,

W'_i be the scaled workload with DOP = i

In general, $W'_i > W_i$ for $2 \le i \le m'$ and $W'_1 = W_1$ assumption

Assuming that $T(1)=T'(n)$ we obtain :

i.e fixed execution time

$$T(1) = T'(n) = \sum_{i=1}^{m} W_i = \sum_{i=1}^{m'} \frac{W'_i}{i} \left\lceil \frac{i}{n} \right\rceil + h(s,n)$$

Speedup $S'_n = T'(1)/T'(n)$ **is given by:**

Time on one processor for scaled problem

$$S'_n = \frac{T'(1)}{T'(n)} = \frac{T'(1)}{T(1)} = \frac{\sum_{i=1}^{m'} W'_i}{\sum_{i=1}^{m} \frac{W'_i}{i} \left\lceil \frac{i}{n} \right\rceil + h(s,n)} = \frac{\sum_{i=1}^{m'} W'_i}{\sum_{i=1}^{m} W_i}$$

Original workload

Fixed-Time Speedup

Total parallelization overheads term

s = problem size n = number of processors

Gustafson's Fixed-Time Speedup

- **For the special fixed-time speedup case where DOP can either be 1 or *n* and assuming h(s,n) = 0** **i.e no overheads**

Time for scaled up problem on one processor

fixed execution time

$$T(1) = T'(n)$$

Also assuming:

$$W'_1 = W_1$$

$$S'_n = \frac{T'(1)}{T'(n)} = \frac{T'(1)}{T(1)} = \frac{\sum_{i=1}^{m'} W'_i}{\sum_{i=1}^{m} W_i} = \frac{W'_1 + W'_n}{W_1 + W_n} = \frac{W_1 + nW_n}{W_1 + W_n}$$

assumption

Where $W'_n = nW_n$ and $W_1 + W_n = W'_1 + W'_n/n$

DOP = 1 DOP = n

Assuming $\alpha = W_1$ and $1-\alpha = W_n$ and $W_1 + W_n = 1$

(i.e normalize to 1)

$$S'_n = \frac{T(1)}{T'(n)} = \frac{\alpha + n(1-\alpha)}{\alpha + (1-\alpha)} = n - \alpha(n-1)$$

Alpha = α = Sequential fraction with DOP = 1

Memory Constrained (MC) Scaling

Fixed-Memory Speedup

Problem and machine size

Problem and machine sizes are scaled so memory usage per processor stays fixed.

- **Scale so memory usage per processor stays fixed**
- **Scaled Speedup: Time(1) / Time(n) for scaled up problem**
- **Let M be the memory requirement of a given problem**
- **Let W = g(M) or M = g^{-1}(W) where**

$W = \sum_{i=1}^{m} W_i$ workload for sequential execution $W^* = \sum_{i=1}^{m^*} W_i^*$ scaled workload on *n* nodes

The memory bound for an active node is $g^{-1}\left(\sum_{i=1}^{m} W_i\right)$

The fixed-memory speedup is defined by:

$$S_n^* = \frac{T^*(1)}{T^*(n)} = \frac{\sum_{i=1}^{m^*} W_i^*}{\sum_{i=1}^{m^*} \frac{W_i^*}{i} \left\lceil \frac{i}{n} \right\rceil + h(s,n)}$$

Assuming $W_n^* = g^*(nM) = G(n)g(M) = G(n)W_n$

and either sequential or perfect parallelim and h(s, n) = 0

DOP =1 DOP =n No overheads

$$S_n^* = \frac{W_1^* + W_n^*}{W_1^* + W_n^*/n} = \frac{W_1 + G(n)W_n}{W_1 + G(n)W_n/n}$$

4 cases for G(n)

Also assuming: $W_1^* = W_1$

1. **G(n) = 1 problem size fixed (Amdahl's)**
2. **G(n) = n workload increases n times as memory demands increase n times = Fixed Time**
3. **G(n) > n workload increases faster than memory requirements $S^*_n > S'_n$ — Fixed-Time Speedup**
4. **G(n) < n memory requirements increase faster than workload $S'_n > S^*_n$ — Fixed-Memory Speedup**

S^*_n Memory Constrained, MC (fixed memory) speedup
S'_n Time Constrained, TC (fixed time) speedup

Impact of Scaling Models: 2D Grid Solver

- For sequential n x n solver: memory requirements $O(n^2)$. Computational complexity $O(n^2)$ times number of iterations (minimum O(n)) thus W= $O(n^3)$

1 • **Problem constrained (PC) Scaling:** Fixed problem size

  – Grid size fixed = n x n Ideal Parallel Execution time = $O(n^3/p)$ — Parallelization overheads ignored

  – Memory requirements per processor = $O(n^2/p)$

2 • **Memory Constrained (MC) Scaling:**

  – **Memory requirements stay the same:** $O(n^2)$ per processor.

  – Scaled grid size = k x k = $n\sqrt{p}$ X $n\sqrt{p}$

  – Iterations to converge = $n\sqrt{p}$ = k (new grid size)

  – Workload = $O\left(\left(n\sqrt{p}\right)^3\right)$

  – Ideal parallel execution time = $O\left(\frac{\left(n\sqrt{p}\right)^3}{p}\right) = O\left(n^3\sqrt{p}\right)$

    • Grows by $\sqrt{p}$

$k = n\sqrt{p}$

Scaled Grid

$k = n\sqrt{p}$

Parallelization overheads ignored

Example: 1 hr on uniprocessor for original problem means 32 hr on 1024 processors for scaled up problem (new grid size 32 n x 32 n).

3 • **Time Constrained (TC) scaling:**

  – **Execution time remains the same** $O(n^3)$ as sequential case.

  – If scaled grid size is *k x k*, then $k^3/p = n^3$, so $k = n \times \sqrt[3]{p}$

  – Memory requirements per processor = $k^2/p = n^2/\sqrt[3]{p}$

    • Diminishes as cube root of number of processors

Workload = $O\left(\left(n\sqrt[3]{p}\right)^3\right)$

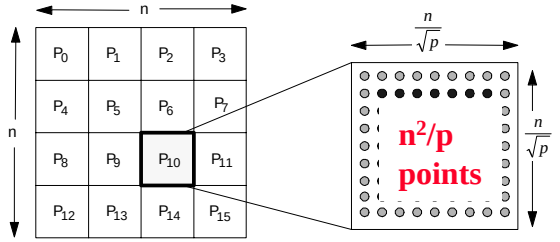
Grows slower than MC

p = number of processors n x n = original grid size

Number of iterations

Impact on Grid Solver Execution Characteristics

- **Concurrency: Total Number of Grid points**
  - **PC: fixed; n^2**
  - **MC: grows as p: p x n^2**
  - **TC: grows as $p^{0.67}$** $n^2 \times p^{\frac{2}{3}}$



- **Comm. to comp. Ratio: Assuming block decomposition**
  - **PC: grows as $\sqrt{p}$; Grid size n fixed** $Communication = \frac{4n}{\sqrt{p}}$ $Computation = \frac{n^2}{p}$
  - **MC: fixed; 4/n** **New grid size k = n $\sqrt{p}$**
  - **TC: grows as $\sqrt[6]{p}$** **New grid size k = $n \times \sqrt[3]{p}$**

$original \quad c-to-c = \frac{4 \times \sqrt{p}}{n}$

- **Working Set: (i.e. Memory requirements per processor)**

  PC: shrinks as *p* : n^2/p **MC: fixed = n^2**

  TC: shrinks as $\sqrt[3]{p}$: $n^2/\sqrt[3]{p}$

- **Expect speedups to be best under MC and worst under PC.**

PC= Problem constrained = Fixed-load or fixed problem size model
MC = Memory constrained = Fixed-memory Model
TC = Time constrained = Fixed-time Model

For a given parallel system and parallel computation/problem/algorithm

Scalability

… of Parallel Architecture/Algorithm Combination

- The study of scalability in parallel processing is concerned with determining the degree of matching between a parallel computer architecture and application/algorithm and whether this degree of matching continues to hold as problem and machine sizes are scaled up .
- Combined architecture/algorithmic scalability imply increased problem size can be processed with acceptable performance level with increased system size for a particular architecture and algorithm.
  - Continue to achieve good parallel performance "speedup"as the sizes of the system/problem are increased.
- Basic factors affecting the scalability of a parallel system for a given problem:

Machine Size *n*	Clock rate *f*
Problem Size *s*	CPU time T
I/O Demand *d*	Memory Capacity m
Communication/other overheads	h(*s*, *n*), where h(*s*, 1) =0
Computer Cost *c*	
Programming Overhead *p*	

For scalability, overhead term must grow slowly as problem/system sizes are increased

Parallel Architecture ⟷ Parallel Algorithm

Match?

As sizes increase

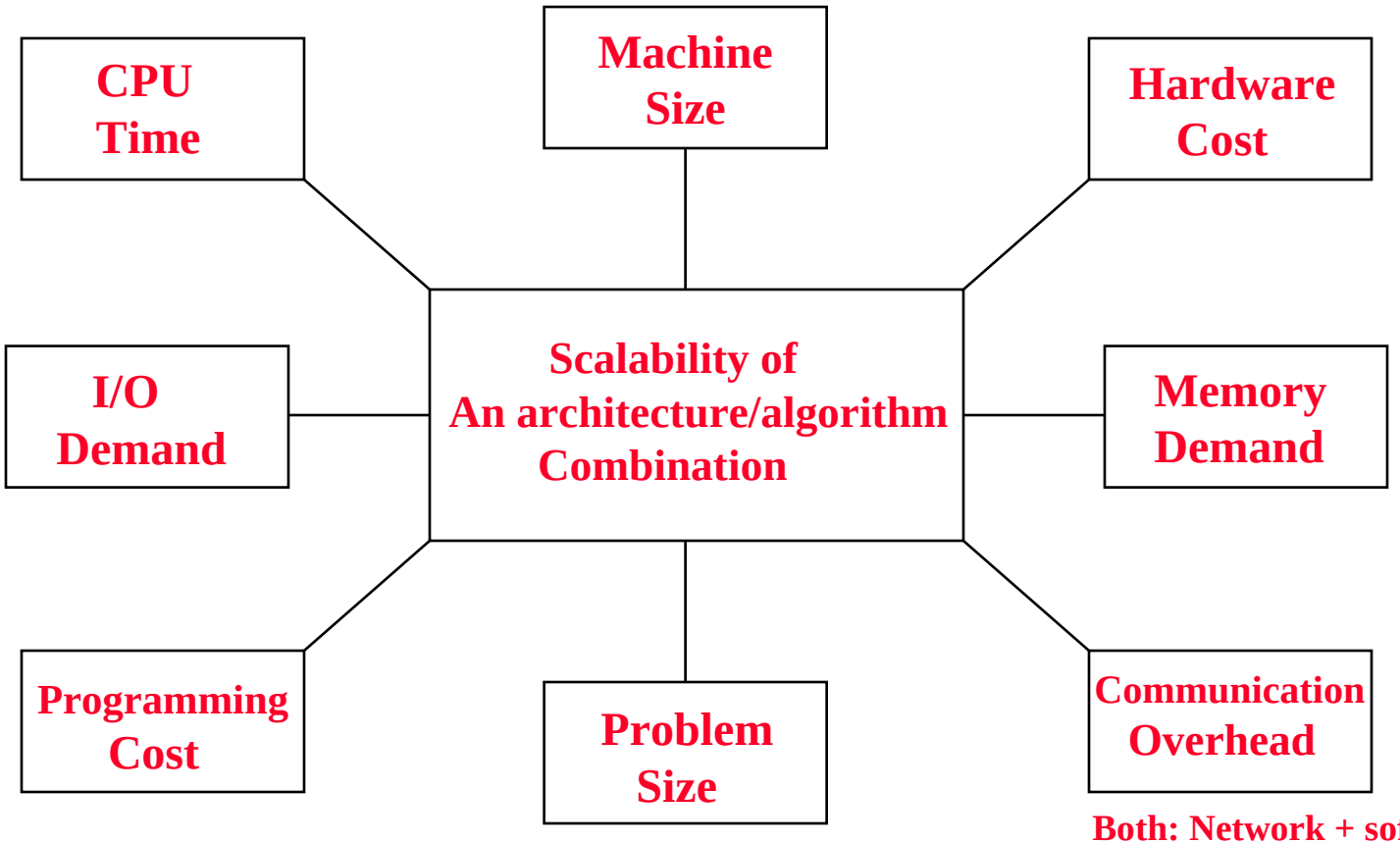
Parallel Scalability Factors

•The study of scalability in parallel processing is concerned with determining the degree of matching between a parallel computer architecture and application/algorithm and whether this degree of matching continues to hold as problem and machine sizes are scaled up .

•Combined architecture/algorithmic scalability imply increased problem size can be processed with acceptable performance level with increased system size for a particular architecture and algorithm.

–Continue to achieve good parallel performance "speedup"as the sizes of the system/problem are increased.

From last slide



Both: Network + software overheads

For a given parallel system and parallel computation/problem/algorithm

Revised Asymptotic Speedup, Efficiency

Vary both problem size S and number of processors n

- **Revised Asymptotic Speedup:**

Accounting for overheads

$$S(s,n) = \frac{T(s,1)}{T(s,n) + h(s,n)}$$

Condition for scalability

Problem/Architecture Scalable if h(s, n) grows slowly as s, n increase

Based on DOP profile

  - **s** problem size.
  - **n** number of processors
  - **T(s, 1)** minimal sequential execution time on a uniprocessor.
  - **T(s, n)** minimal parallel execution time on an n-processor system.
  - **h(s, n)** lump sum of all communication and other overheads.

- **Revised Asymptotic Efficiency:**

$$E(s,n) = \frac{S(s,n)}{n}$$

Parallel System Scalability

- **Scalability (very restrictive definition):**

  A system architecture is scalable if the system efficiency E(s, n) = 1 for all algorithms with any number of processors n and any size problem s

- **Another Scalability Definition (more formal, less restrictive):**

  The scalability Φ(s, n) of a machine for a given algorithm is defined as the ratio of the asymptotic speedup S(s,n) on the real machine to the asymptotic speedup $S_I(s, n)$ on the ideal realization of an EREW PRAM

"Ideal" PRAM Speedup

$$S_I(s,n) = \frac{T(s,1)}{T_I(s,n)}$$

For PRAM

$$\Phi(s,n) = \frac{S(s,n)}{S_I(s,n)} = \frac{T_I(s,n)}{T(s,n)}$$

For real parallel machine

Capital Phi

For PRAM

Ideal Φ ?

s = size of problem n = number of processors

Example: Scalability of Network Architectures for Parity Calculation

Table 3.7 page 142
see handout

Evaluating a Real Parallel Machine

- **Performance Isolation using Microbenchmarks**
- **Choosing Workloads**
- **Evaluating a Fixed-size Machine**
- **Varying Machine Size and Problem Size**
- **All these issues, plus more, relevant to evaluating a tradeoff via simulation**

Performance Isolation: Microbenchmarks

- **Microbenchmarks: Small, specially written programs to isolate performance characteristics**
  - **Processing.**
  - **Local memory.**
  - **Input/output.**
  - **Communication and remote access (read/write, send/receive)**
  - **Synchronization (locks, barriers).**
  - **Contention.**

Types of Workloads/Benchmarks

- ***Kernels:*** **matrix factorization, FFT, depth-first tree search**
- ***Complete Applications:*** **ocean simulation, ray trace, database.**
- ***Multiprogrammed Workloads.***

- Multiprog. ⟷ Appls ⟷ Kernels ⟷ Microbench.

Realistic	Easier to understand
Complex	Controlled
Higher level interactions	Repeatable
Are what really matters	Basic machine characteristics

Each has its place:

Use kernels and microbenchmarks to gain understanding, but full applications needed to evaluate realistic effectiveness and performance

Three Desirable Properties for Parallel Workloads

1. **Representative of application domains.**

2. **Coverage of behavioral properties.**

3. **Adequate concurrency.**

Desirable Properties of Workloads:

1 Representative of Application Domains

- **Should adequately represent domains of interest, e.g.:**
  - ***Scientific*: Physics, Chemistry, Biology, Weather ...**
  - ***Engineering*: CAD, Circuit Analysis ...**
  - ***Graphics*: Rendering, radiosity ...**
  - ***Information management*: Databases, transaction processing, decision support ...**
  - ***Optimization***
  - ***Artificial Intelligence*: Robotics, expert systems ...**
  - ***Multiprogrammed general-purpose workloads***
  - ***System software*: e.g. the operating system**

Etc….

Desirable Properties of Workloads:

2 Coverage: Stressing Features

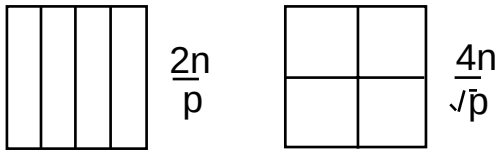
- Some features of interest to be covered by workload:
  - Compute v. memory v. communication v. I/O bound
  - Working set size and spatial locality
  - Local memory and communication bandwidth needs
  - Importance of communication latency
  - Fine-grained or coarse-grained
    - Data access, communication, task size
  - Synchronization patterns and granularity
  - Contention
  - Communication patterns

→ • Choose workloads that cover a range of properties

Coverage: Levels of Optimization

Example Grid Problem

- Many ways in which an application can be suboptimal
  - *Algorithmic*, e.g. assignment, blocking



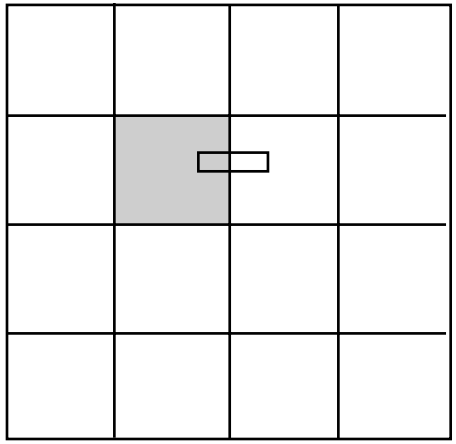
  - *Data structuring*, e.g. 2-d or 4-d arrays for SAS grid problem
  - *Data layout, distribution and alignment*, even if properly structured
  - *Orchestration*
    - contention
    - long versus short messages
    - synchronization frequency and cost, ...
  - Also, random problems with "unimportant" data structures
- Optimizing applications takes work
  - Many practical applications may not be very well optimized
- May examine selected different levels to test robustness of system

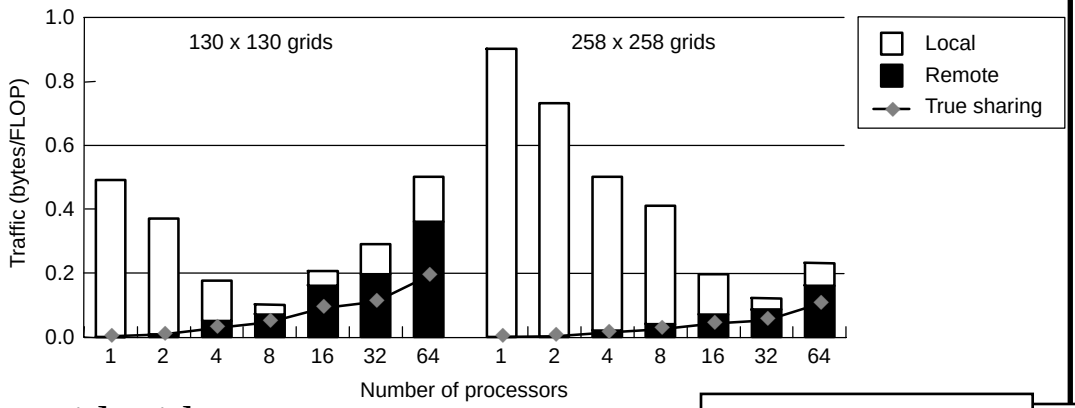
Desirable Properties of Workloads:

3 Concurrency

- **Should have enough to utilize the processors**
  - **If load imbalance dominates, may not be much machine can do**
  - **(Still, useful to know what kinds of workloads/configurations don't have enough concurrency)**
- ***Algorithmic speedup*: useful measure of concurrency/imbalance**
  - **Speedup (under scaling model) assuming all memory/communication operations take zero time**
  - **Ignores memory system, measures imbalance and extra work**
  - **Uses PRAM machine model (Parallel Random Access Machine)**
    - **Unrealistic, but widely used for theoretical algorithm development**
- **At least, should isolate performance limitations due to program characteristics that a machine cannot do much about (concurrency) from those that it can.**

Effect of Problem Size Example: Ocean





n-by-n grid with *p* processors
(computation like grid solver)

For block decomposition:

$$Communication = \frac{4n}{\sqrt{p}}$$

$$Computation = \frac{n^2}{p}$$

$$c-to-c = \frac{4 \times \sqrt{p}}{n}$$

n/p is large $\Rightarrow$

- Low communication to computation ratio
- Good spatial locality with large cache lines
- Data distribution and false sharing not problems even with 2-d array
- Working set doesn't fit in cache; high local capacity miss rate.

n/p is small $\Rightarrow$

- High communication to computation ratio
- Spatial locality may be poor; false-sharing may be a problem
- Working set fits in cache; low capacity miss rate.

e.g. Shouldn't make conclusions about spatial locality based only on small problems, particularly if these are not very representative.