ECE-451/ECE-566 - Introduction to Parallel and Distributed Programming

Evaluating Parallel Performance

Department of Electrical & Computer Engineering
Rutgers University

Speedup Factor

$$S(p) = \frac{\text{Execution time using one processor (best sequential algorithm)}}{\text{Execution time using a multiprocessor with } p \text{ processors}} = \frac{t_s}{t_p}$$

where t_s is execution time on a single processor and t_p is execution time on a multiprocessor.

$S(p)$ gives increase in speed by using multiprocessor.

Use best sequential algorithm with single processor system. Underlying algorithm for parallel implementation might be (and is usually) different.

Speedup factor can also be cast in terms of computational steps:

$$S(p) = \frac{\text{Number of computational steps using one processor}}{\text{Number of parallel computational steps with } p \text{ processors}}$$

Can also extend time complexity to parallel computations.

Maximum Speedup

Maximum speedup is usually p with p processors (linear speedup).

Possible to get superlinear speedup (greater than p) but usually a specific reason such as:

- Extra memory in multiprocessor system
- Nondeterministic algorithm

Models of Speedup

- **Speedup**

$$S_p = \frac{\text{Uniprocessor Execution Time}}{\text{Parallel Execution Time}}$$

- **Scaled Speedup**
  - **Parallel processing gain over sequential processing, where problem size scales up with computing power (having sufficient workload/parallelism)**

5

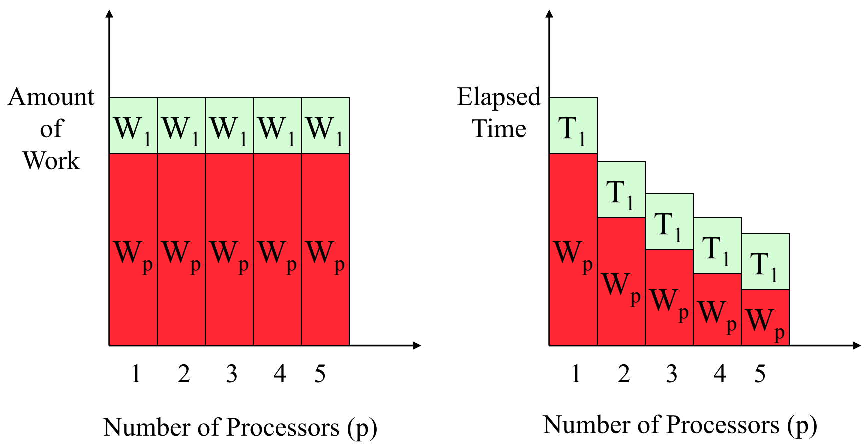
Fixed-Size Speedup (Amdahl Law)

- Fixed-Size Speedup (Amdahl)
  - Emphasis on turnaround time
  - Problem size, ***W***, is fixed

$$S_p = \frac{\text{Uniprocessor Time of Solving } W}{\text{Parallel Time of Solving } W}$$

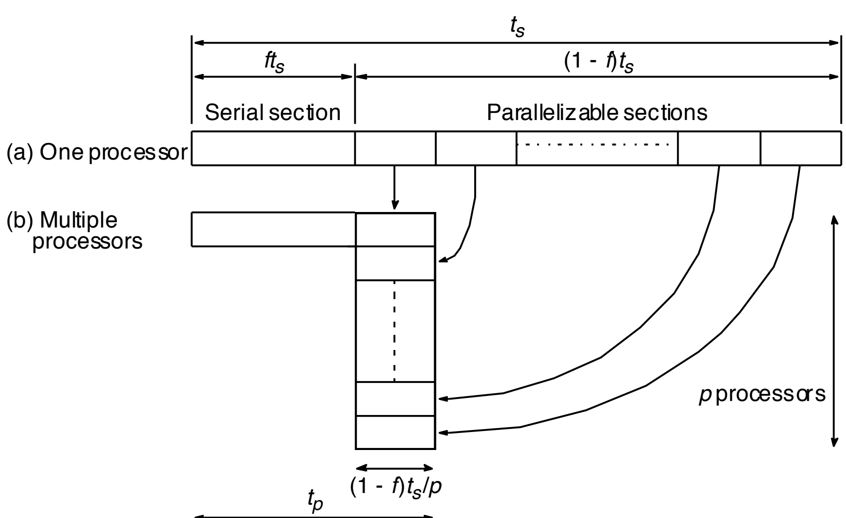
6

Fixed-Size Speedup (Amdahl Law)



7

Maximum Speedup
Amdahl's law



8

Speedup factor (maximum speedup) is given by:

$$S(p) = \frac{t_s}{f \cdot t_s + (1-f) \cdot t_s / p} = \frac{p}{1 + (p-1) \cdot f}$$

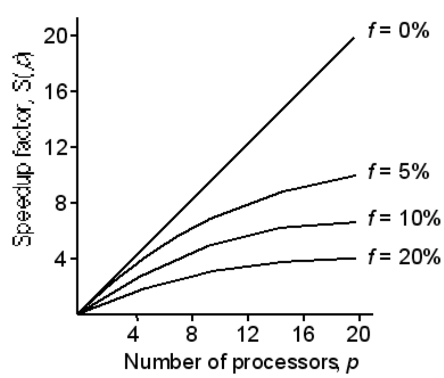
Where:

f is the fraction of this time that cannot be parallelized

p processors

This equation is known as Amdahl's law

Speedup against number of processors

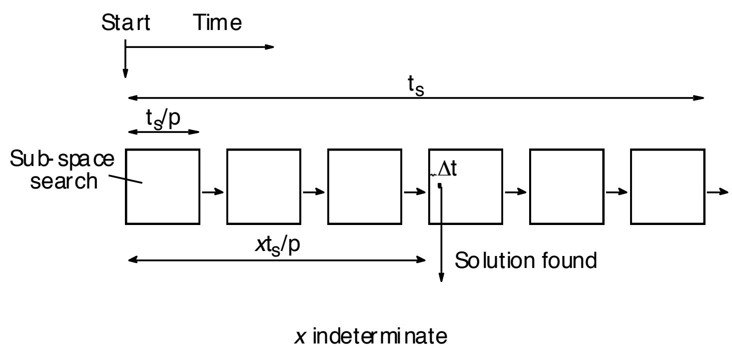


Even with infinite number of processors, maximum speedup limited to 1/f .

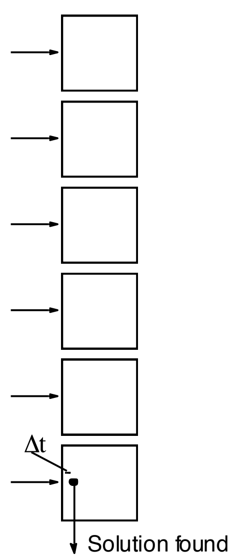
Example: With only 5% of computation being serial, maximum speedup is 20, irrespective of number of processors.

Superlinear Speedup example
- Searching

(a) Searching each sub-space sequentially



(b) Searching each sub-space in parallel



Speed-up then given by

$$S(p) = \frac{(x) \times \frac{t_s}{p} + \Delta t}{\Delta t}$$

Worst case for sequential search when solution found in last sub-space search. Then parallel version offers greatest benefit, i.e.

$$S(p) = \frac{\frac{p-1}{p} \times t_s + \Delta t}{\Delta t} \rightarrow \infty$$

as Δt tends to zero

Least advantage for parallel version when solution found in first sub-space search of the sequential search, i.e.

$$S(p) = \frac{\Delta t}{\Delta t} = 1$$

Actual speed-up depends upon which subspace holds solution but could be extremely large.

15

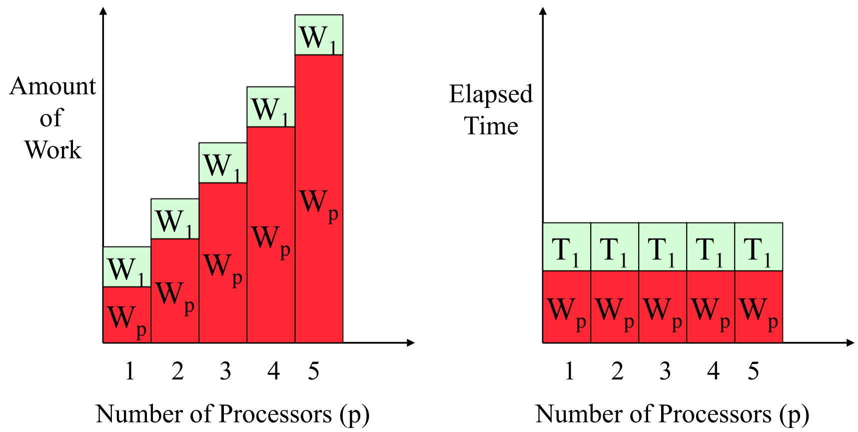
Fixed-Time Speedup (Gustafson)

- Fixed-Time Speedup (Gustafson)
  - Emphasis on work finished in a fixed time
  - Problem size is scaled from W to W'
  - W': Work finished within the fixed time with parallel processing

$$S'_p = \frac{\text{Uniprocessor Time of Solving } W'}{\text{Parallel Time of Solving } W'}$$
$$= \frac{\text{Uniprocessor Time of Solving } W'}{\text{Uniprocessor Time of Solving } W}$$
$$= \frac{W'}{W}$$

16

Fixed-Time Speedup (Gustafson)



17

Steps in Writing Parallel Programs

Creating a Parallel Program

- Identify work that can be done in parallel.
- Partition work and perhaps data among logical processes (threads).
- Manage the data access, communication, synchronization.
- Goal: maximize speedup due to parallelism

$$\text{Speedup}_{prob}(\text{P procs}) = \frac{\text{Time to solve prob with "best" sequential solution}}{\text{Time to solve prob in parallel on P processors}}$$

$$\le P \quad (?)$$

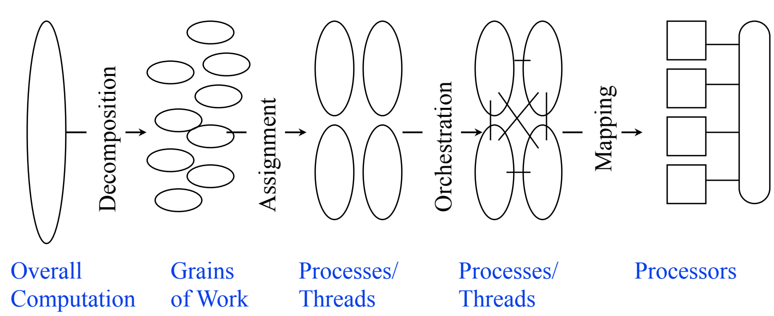
$$\text{Efficiency(P)} = \text{Speedup(P)} / P$$

$$\le 1$$

- Key question is when you can solve each piece:
  - statically, if information is known in advance.
  - dynamically, otherwise.

22

Steps in the Process



- Task: arbitrarily defined piece of work that forms the basic unit of concurrency.
- Process/Thread: abstract entity that performs tasks
  - tasks are assigned to threads via an assignment mechanism.
  - threads must coordinate to accomplish their collective tasks.
- Processor: physical entity that executes a thread.

23

Decomposition

- Break the overall computation into individual grains of work (tasks).
  - Identify concurrency and decide at what level to exploit it.
  - Concurrency may be statically identifiable or may vary dynamically.
  - It may depend only on problem size, or it may depend on the particular input data.
- Goal: identify enough tasks to keep the target range of processors busy, but not too many.
  - Establishes upper limit on number of useful processors (i.e., scaling).
- Tradeoff: sufficient concurrency vs. task control overhead.

24

Assignment

- Determine mechanism to divide work among threads
  - Functional partitioning:
    - Assign logically distinct aspects of work to different thread, e.g. pipelining.
  - Structural mechanisms:
    - Assign iterations of "parallel loop" according to a simple rule, e.g. proc j gets iterates j*n/p through (j+1)*n/p-1.
    - Throw tasks in a bowl (task queue) and let threads feed.
  - Data/domain decomposition:
    - Data describing the problem has a natural decomposition.
    - Break up the data and assign work associated with regions, e.g. parts of physical system being simulated.
- Goals:
  - Balance the workload to keep everyone busy (all the time).
  - Allow efficient orchestration.

25

Orchestration

° **Provide a means of**
- **Naming and accessing shared data.**
- **Communication and coordination among threads of control.**

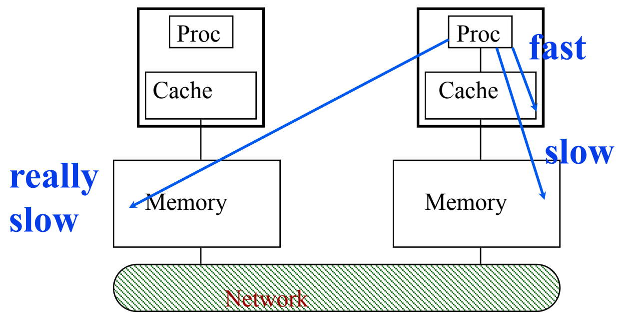
° **Goals:**
- **Correctness of parallel solution -- respect the inherent dependencies within the algorithm.**
- **Avoid serialization.**
- **Reduce cost of communication, synchronization, and management.**
- **Preserve locality of data reference.**

Mapping

° **Binding processes to physical processors.**

° **Time to reach processor across network does not depend on which processor (roughly).**
- **lots of old literature on "network topology", no longer so important.**

° **Basic issue is how many remote accesses.**



Example

- $s = f(A[1]) + \ldots + f(A[n])$
- **Decomposition**
  - computing each f(A[j])
  - n-fold parallelism, where n may be >> p
  - computing sum s
- **Assignment**
  - thread k sums $s_k = f(A[k*n/p]) + \ldots + f(A[(k+1)*n/p-1])$
  - thread 1 sums $s = s_1 + \ldots + s_p$ (for simplicity of this example)
  - thread 1 communicates s to other threads
- **Orchestration**
  - starting up threads
  - communicating, synchronizing with thread 1
- **Mapping**
  - processor j runs thread j

28

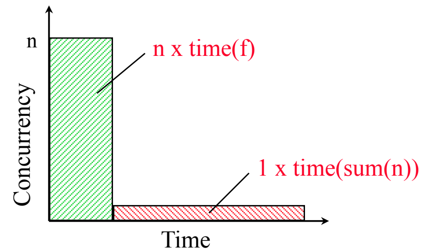
Cost Modeling and Performance Tradeoffs

Identifying enough Concurrency

- **Parallelism profile**
  - **area is total work done**

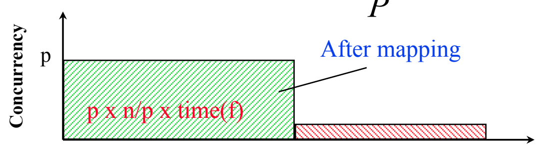
Simple Decomposition:
f (A[i]) is the parallel task

sum is sequential



- **Amdahl's law**
  - **let s be the fraction of total work done sequentially**

$$Speedup(P) \le \frac{1}{s + \frac{1-s}{P}} \le \frac{1}{s}$$

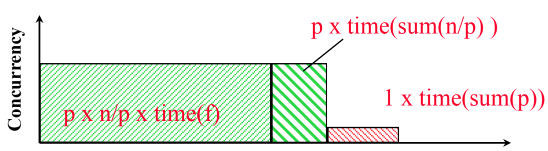


P	S=0%	S=1%	S=5%	S=10%
1	1	1	1	1
2	2	1.980198	1.904762	1.818182
5	5	4.807692	4.166667	3.571429
10	10	9.174312	6.896552	5.263158
20	20	16.80672	10.25641	6.896552
30	30	23.25581	12.2449	7.692308
40	40	28.77698	13.55932	8.163265
50	50	33.55705	14.49275	8.474576
60	60	37.73585	15.18987	8.695652
70	70	41.42012	15.73034	8.860759
80	80	44.69274	16.16162	8.988764
90	90	47.61905	16.51376	9.090909
100	100	50.25126	16.80672	9.174312

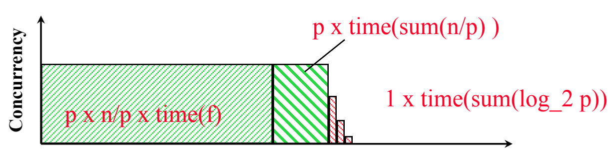
30

Algorithmic Trade-offs

- **Parallelize partial sum of the f's**
  - **what fraction of the computation is "sequential"**



- **Parallelize the final summation (tree sum)**



31

Problem Size is Critical

- Suppose Total work= n + P
- Serial work: P
- Parallel work: n
- s = serial fraction
  = P/ (n+P)

Amdahl's Law Bounds

n

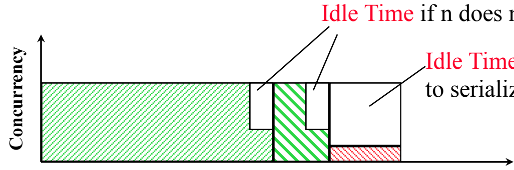
P	1000	10000	1000000
1	1	1	1
2	1.996016	1.9996	1.999996
5	4.902439	4.990025	4.9999
10	9.181818	9.910891	9.9991
20	14.57143	19.26923	19.9924
30	16.26316	27.6055	29.97392
40	16	34.62069	39.9377
50	15	40.2	49.87781
60	13.82609	44.38235	59.78836
70	12.69492	47.30872	69.66355
80	11.67568	49.17073	79.49762
90	10.78022	50.17127	89.28489
100	10	50.5	99.0198

In general, seek to exploit a large fraction of the peak parallelism in the problem.

32

Load Balancing Issues

- Insufficient concurrency will appear as **load imbalance**.



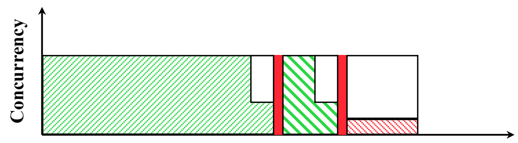
- Use of coarser grain tends to increase load imbalance.
- Poor assignment of tasks can cause load imbalance.
- Synchronization waits are instantaneous load imbalance

$$Speedup(P) \le \frac{Work(1)}{\max_p(Work(p) + idle)}$$

33

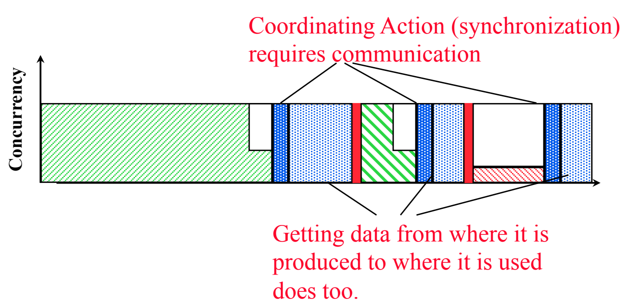
Extra Work

- There is always some amount of extra work to manage parallelism -- e.g. deciding who is to do what.



$$Speedup\,(P) \le \frac{Work\,(1)}{\operatorname{Max}_p(Work\,(p) + idle + extra\,)}$$

Communication and Synchronization

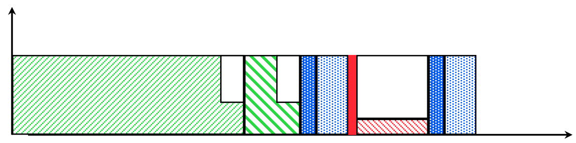


$$Speedup(P) \le \frac{Work(1)}{\max(Work(P) + idle + extra + comm)}$$

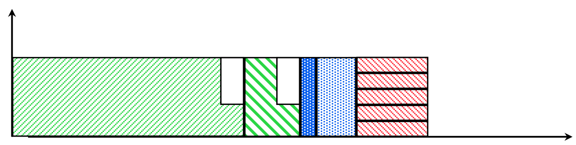
- There are many ways to reduce communication costs.

Reducing Communication Costs

- Coordinating placement of work and data to eliminate unnecessary communication.



- Replicating data.
- Redundant work.



- Performing required communication efficiently.
  - e.g., transfer size, contention, machine specific optimizations

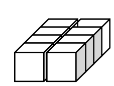
36

The Tension

$$Speedup(P) \leq \frac{Work(1)}{\max(Work(P) + idle + comm + extraWork)}$$

Minimizing one tends to increase the others

- Fine grain decomposition and flexible assignment tends to minimize load imbalance at the cost of increased communication
  - In many problems communication goes like the surface-to-volume ratio
  - Larger grain => larger transfers, fewer synchronization events
- Simple static assignment reduces extra work, but may yield load imbalance



37

The Good News

- The basic work component in the parallel program may be more efficient than in the sequential case.
  - Only a small fraction of the problem fits in cache.
  - Need to chop problem up into pieces and concentrate on them to get good cache performance.
  - Similar to the parallel case.
  - Indeed, the best sequential program may emulate the parallel one.
- Communication can be hidden behind computation.
  - May lead to better algorithms for memory hierarchies.
- Parallel algorithms may lead to better serial ones.
  - Parallel search may explore space more effectively.

38

Parallel Performance Metrics

Parallel Performance Metrics

(Run-time is the dominant metric)

- Run-Time (Execution Time)
- Speed: mflops, mips
- Speedup
- Efficiency:

$$E = \frac{\text{Speedup}}{\text{Number of Processors}}$$

- Throughput: X / Time
  - Where X can be jobs/iterations/problems, etc.
- Scalability

40

Use of Parallelism

- Capability computing
  - Address “big” problems
    - Not just time (CPUs)
    - Also memory, I/O, etc.
- Capacity computing
  - Throughput. Many problems
- Other indicators of performance
  - Quality, Price, etc.
    - MFLOPS/$
    - MFLOP/Watt
    - MFLOP/m^2

41