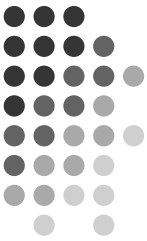
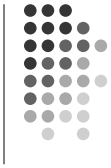
Concurrency 8A

Distributed and Parallel Computing

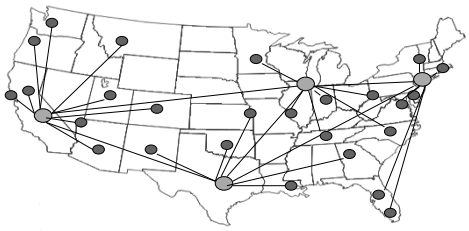


Distributed Computing



- Coordination of physically distributed computers to access shared resources.
  - The Internet itself (e.g. World Wide Web)
  - SETI - Search for Extraterrestrial Intelligence
  - Airline reservation systems (e.g. SABRE, Travelocity)
  - Google web indexing

Airline Reservation System



- Challenges:
  - Reserving seat for plane to only one customer if multiple requests come in at the same time
  - Dealing with down computers

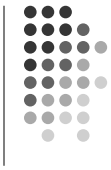
Google

- Runs on a distributed network of thousands of low-cost computers
- Main software components of Google:
  - Googlebot
  - Indexer
  - The Query Processor
    - uses PageRank

from www.googleguide.com

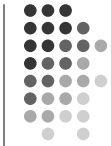
`http://www.google.com/technology/`

Parallelism

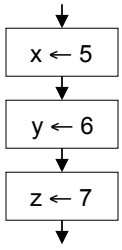


- What if a computer contained more than one processor?
  - Each processor can be used to solve some part of a problem.
  - The solutions to the subproblems could then be merged using one or more processors to yield the result.
  - This is the idea behind a parallel computer.

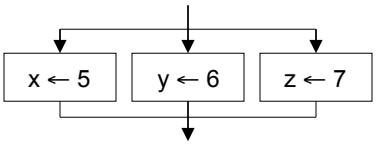
Parallel Computation



Sequential processors)



Parallel (using 3

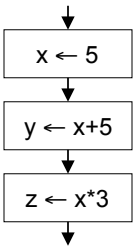


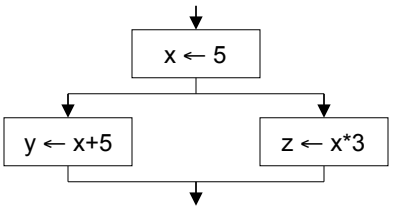
- Work is done in 1/3 the time.

Parallel Computation

- CAREFUL! Not all sequential computations can be parallelized easily:

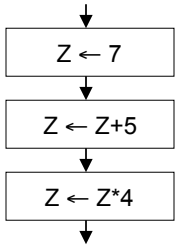
  Sequential Parallel (using 2 processors)



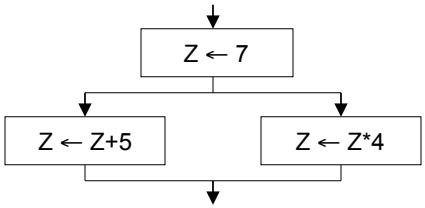


Parallel Computation

- Can we compute the last two instructions in parallel?



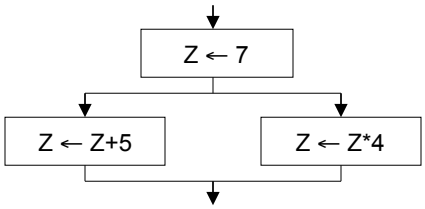
Parallel Computation



1. Set Z to 7.
2. Processor 1 gets the value in Z.
3. Processor 2 gets the value in Z.
4. Processor 1 adds 5 to its value of Z and stores the result in Z.
5. Processor 2 multiplies its value of Z by 4 and stores the result in Z, overwriting the result that processor 1 just wrote.

THE FINAL VALUE OF Z IS _____

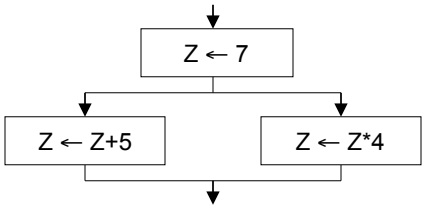
Parallel Computation



1. Set Z to 7.
2. Processor 1 gets the value in Z.
3. Processor 2 gets the value in Z.
4. Processor 2 multiplies its value of Z by 4 and stores the result in Z.
5. Processor 1 adds 5 to its value of Z and stores the result in Z, overwriting the result that processor 2 just wrote.

THE FINAL VALUE OF Z IS _____

Parallel Computation

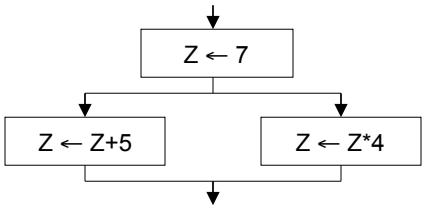


atomic operations

1. Set Z to 7.
2. Processor 1 gets the value in Z.
3. Processor 1 adds 5 to its value of Z and stores the result in Z.
4. Processor 2 gets the value in Z.
5. Processor 2 multiplies its value of Z by 4 and stores the result in Z, overwriting the result that processor 1 wrote.

THE FINAL VALUE OF Z IS _____

Parallel Computation

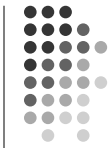


atomic operations

1. Set Z to 7.
2. Processor 2 gets the value in Z.
3. Processor 2 multiplies its value of Z by 4 and stores the result in Z.
4. Processor 1 gets the value in Z.
5. Processor 1 adds 5 to its value of Z and stores the result in Z, overwriting the result that processor 2 wrote.

THE FINAL VALUE OF Z IS _____

Example

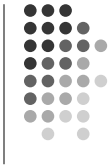


Three processors run a sequence of instructions, including the following:

Processor 1	Processor 2	Processor 3
:	:	:
❶ X = X + 5	❸ X = X + 6	❺ X = X + 7
:	:	:
❷ X = X * 2	❹ X = X * 3	❻ X = X * 4
:	:	:

(Assume X starts off initially at 0, each instruction is an atomic operation and that the processors do not necessarily run at the same pace nor at a constant pace.)

Example (cont'd)



Processor 1	Processor 2	Processor 3
:	:	:
❶ X = X + 5	❸ X = X + 6	❺ X = X + 7
:	:	:
❷ X = X * 2	❹ X = X * 3	❻ X = X * 4
:	:	:

One possible ordering of instructions is:
❶❸❺❷❹❻ yielding a final value of x = _____
Another possible ordering of instructions is:
❸❶❷❺❻❹ yielding a final value of x = _____
The following ordering is not possible:
❺❸❷❻❶❹ Why?

Example

Processor 1	Processor 2	Processor 3
❶ X = X + 5	❸ X = X + 6	❺ X = X + 7
❷ X = X * 2	❹ X = X * 3	❻ X = X * 4

How many possible orderings are there?

If we start with ❶, there are only 30 possible orderings:

❶❷❸❹❺❻	❶❸❷❹❺❻	❶❸❺❷❹❻	❶❺❷❸❹❻	❶❺❸❹❷❻
❶❷❸❺❹❻	❶❸❷❺❹❻	❶❸❺❷❻❹	❶❺❷❸❻❹	❶❺❸❹❻❷
❶❷❸❺❻❹	❶❸❷❺❻❹	❶❸❺❹❷❻	❶❺❷❻❸❹	❶❺❸❻❷❹
❶❷❺❸❹❻	❶❸❹❷❺❻	❶❸❺❹❻❷	❶❺❷❻❹❸	❶❺❸❻❹❷
❶❷❺❸❻❹	❶❸❹❺❷❻	❶❸❺❻❷❹	❶❺❹❷❸❻	❶❺❹❸❷❻
❶❷❺❻❸❹	❶❸❹❺❻❷	❶❸❺❻❹❷	❶❺❹❷❻❸	❶❺❹❸❻❷

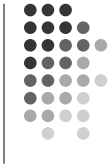
If we start with ❸, there are 30 more orderings.

If we start with ❺, there are 30 more orderings. Total=90

Maximum Speedup

- Assume an algorithm runs sequentially (on one processor) in time t.
- If we are given p processors, and we can split the algorithm up evenly between all p processors, then the best running time for our algorithm will be t/p.
- Example:
  - Let S be a sequential algorithm that runs in 2 mins.
  - If P is a parallel version that splits the work in the sequential algorithm evenly between 10 processors, then the running time of P would be 0.2 min = 12 sec.
    (assuming maximum speedup)

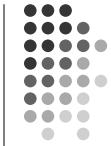
Problems



- The maximum speedup is not typically obtainable because:
  - Some parts of the algorithm may not be parallelizable.
  - Some parts of the algorithm may not require use of all the processors.
    (See sorting example in textbook.)
  - Parallel algorithms also require overhead in terms of communication between processors and protection of shared memory.

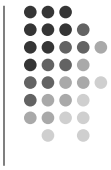
Amdahl's Law

Named after Gene Amdahl, founder of Amdahl Corporation, competitor to IBM in the 1970s in computer mainframe hardware.



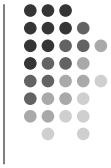
- Let *f* be the fraction of operations in a computation that must be performed sequentially, where $0 \leq f \leq 1$.
- The maximum speedup *s* achievable by a parallel computer with *p* processors is
  $s = 1/ (f + (1-f)/p)$
  - Example: If half of the operations must be performed sequentially and we have 2 processors (f = 0.5, p = 2), then the maximum speedup is s = 1/(0.5 + (1-0.5)/2) = 4/3.

Computability



- If we use a parallel computer or distributed computers, can we solve previously **unsolvable** problems?
- NO
  - Parallel computations can be simulated with sequential computers (like the Turing Machine), so parallel computations give us no more power than we had before computationally.

Computability



- If we use a parallel computer or distributed computers, can we solve previously **intractable** problems?
- YES
  - We can take a problem with an exponential number of potential solutions, and check each one in polynomial time using a separate processor.
  - BUT, we would need an exponential number of processors and a very complex communication network/algorithm.