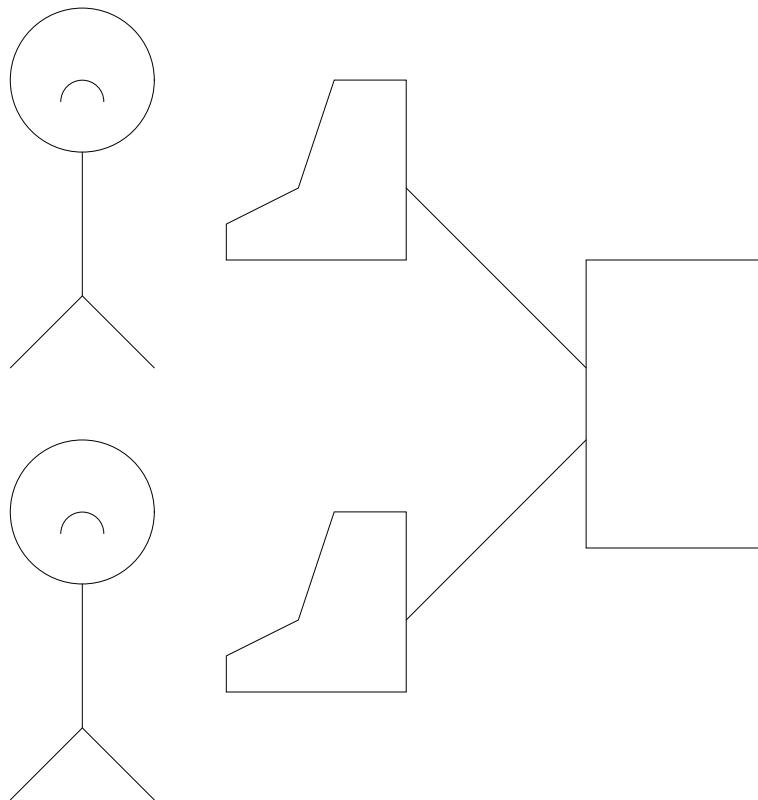


Experiences in Parallel and Distributed Simulation

Bruno R. Preiss, Ph.D., P.Eng.
Associate Professor
Department of Electrical and Computer
Engineering
University of Waterloo
Waterloo, ON N2L 3G1
Canada



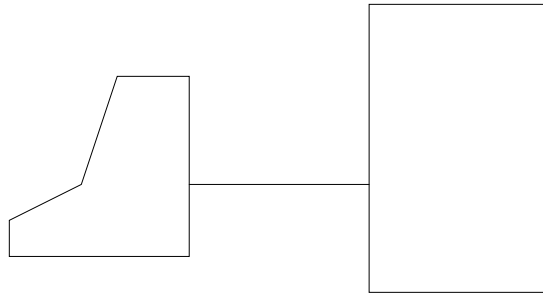
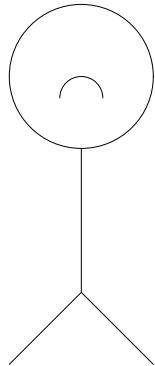
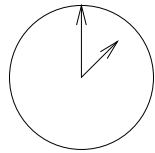
Why Parallel Processing?



~> More Jobs



Why Parallel Processing?



~> Less Time



What are some of the Issues in Using a Multiprocessor?

- speedup
- granularity
- regularity



What is Speedup?

- given workload X
- and execution time T

\leadsto throughput X/T

- given time to execute on 1 processor T_1
- given time to execute on n processors T_n

$$\begin{aligned}\text{speedup} &= \frac{\text{throughput}_n}{\text{throughput}_1} \\ &= \frac{X/T_n}{X/T_1} \\ &= T_1/T_n\end{aligned}$$



What is Ideal Speedup?

- recall speedup = $\frac{T_1}{T_n}$

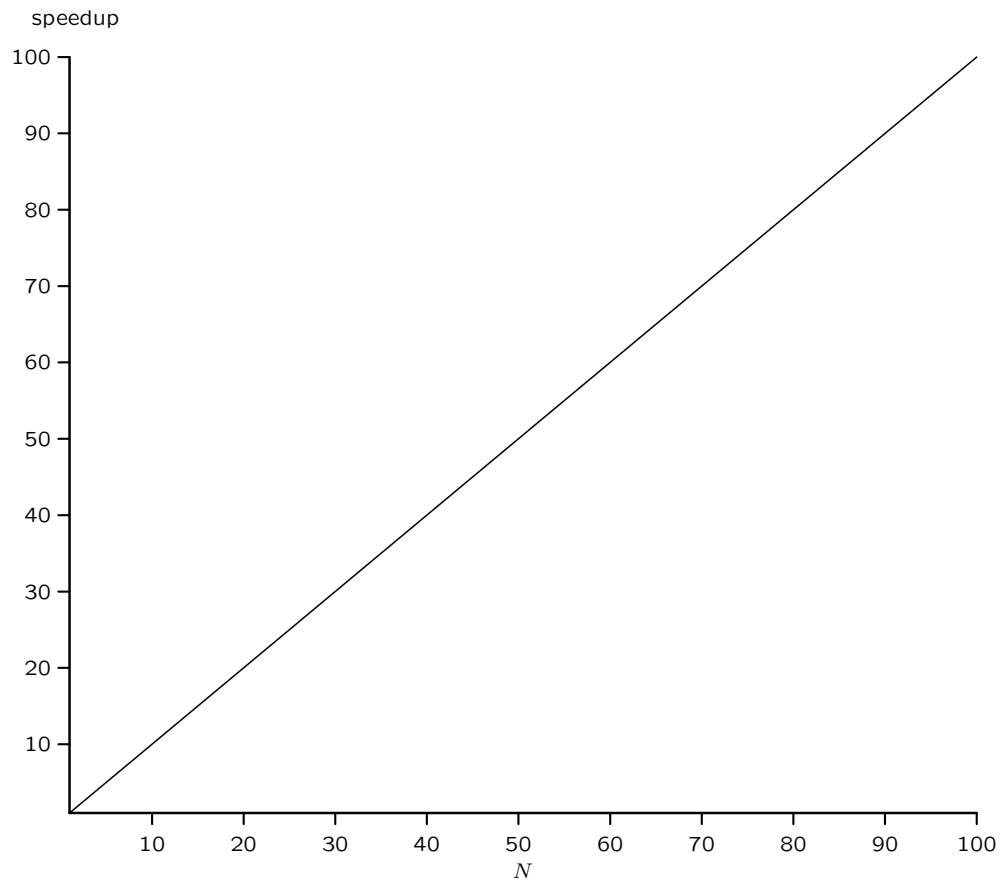
- use a *Newtonian* model:

“If you push twice as hard,
you will go twice as fast.”

- claim: ideally $T_n = T_1/n$

~> ideally speedup = n





What is Amdahl's Law

- more realistic assumptions (first-order)
- workload consists of two parts:
 - sequential part, S
 - parallel part, P

- time to execution on 1 processor

$$T_1 = S + P$$

- time to execution on n processors

$$T_n = S + P/n$$

$$speedup = \frac{1}{1 + (1/n - 1)f}$$

- $f = P/(S + P)$ is the fraction of the workload that is parallel



Implications of Amdahl's Law

$$speedup = \frac{1}{1 + (1/n - 1)f}$$

- $\lim_{n \rightarrow \infty} speedup = \frac{1}{1-f}$

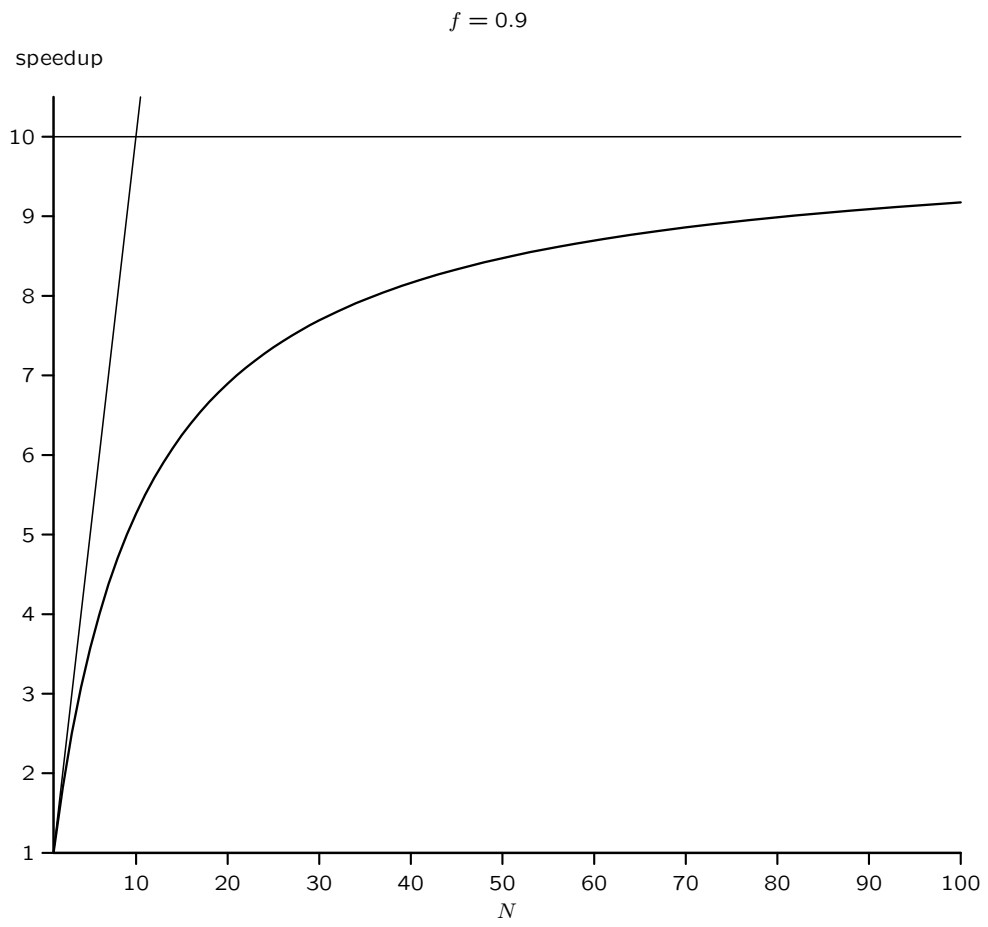
Example 1

- $f = 1 \implies speedup = n$ (100% parallel)

Example 2

- $f = 0.9 \implies speedup < 10$ (90% parallel)

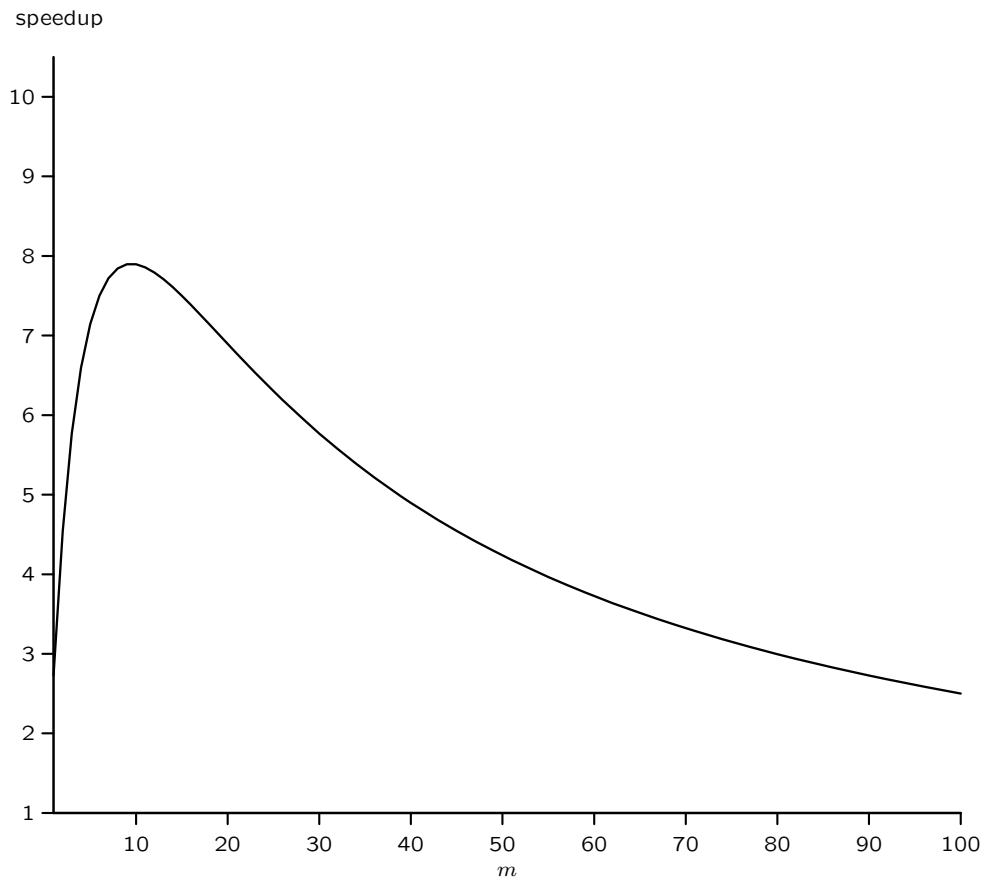




What About Granularity?

- take a computation X
- partition it into m granules
- for execution on n processors
- small $m \rightsquigarrow$ coarse granularity
 - not much parallelism
- large $m \rightsquigarrow$ fine granularity
 - too much overhead
- **conjecture** \exists optimal granularity





What is Regularity?

- Does the computation consist of a large number of identical operations on a large set of data?

Regular Applications

- vectors and matrices
- large data sets
- spatially distributed data
- e.g., image processing and signal processing



Irregular Applications

- everything else (?)
- many dissimilar operations
- high degree of data dependency
- asynchrony



Is there an Application that has the following Characteristics?

- needs speedup via parallelism
- exhibits medium to fine granularity
- exhibits irregular parallelism

Discrete-Event Simulation

- parallel simulation
- distributed simulation

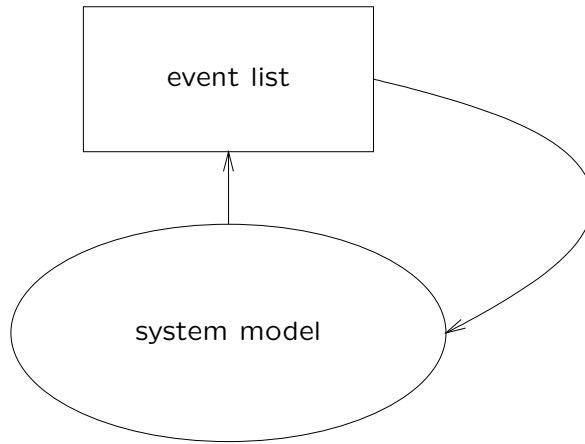
~> parallel and distributed simulation (PADS)



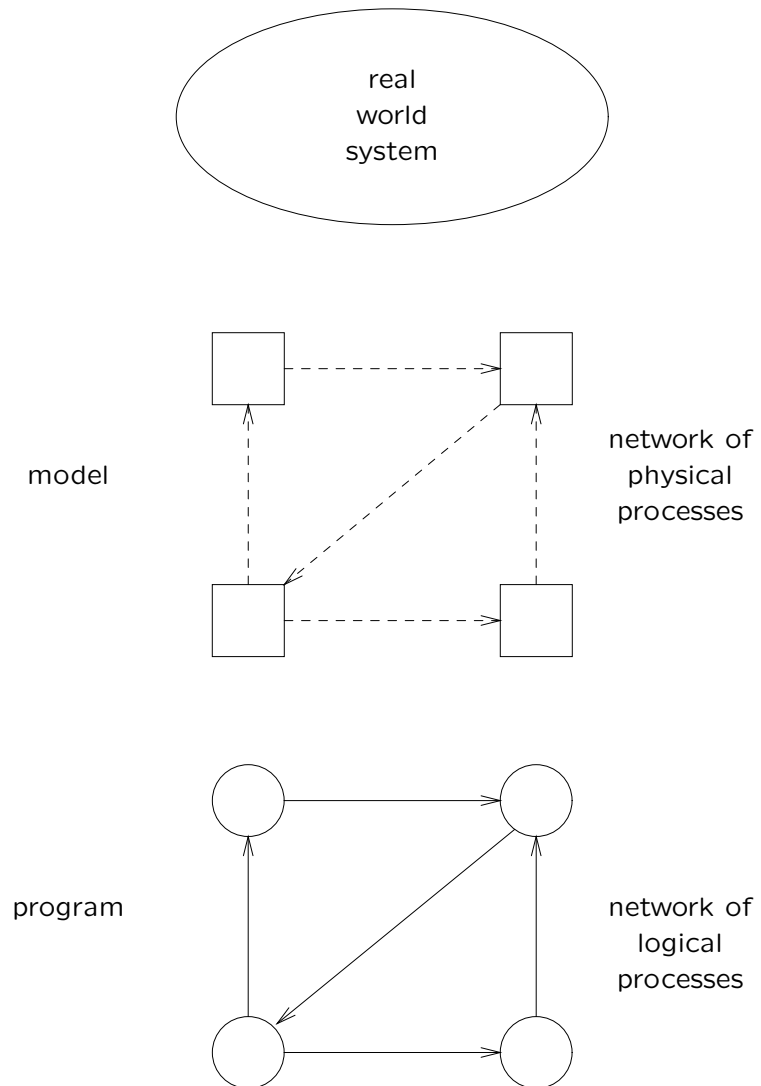
Discrete Event Simulation

- model real-world process as a sequence of events
- events occur at discrete points in time
- events are causally related
- events are processed in order
- the processing of an event creates new events
- events may cancel (preempt) other events





Parallel Discrete-Event Simulation



Real-World System

- large amount of inherent, irregular parallelism
- many independent but interacting components

Model

- consists of physical processes (PPs)
- physical processes periodically exchange information
- information exchange occurs at discrete points in time—events

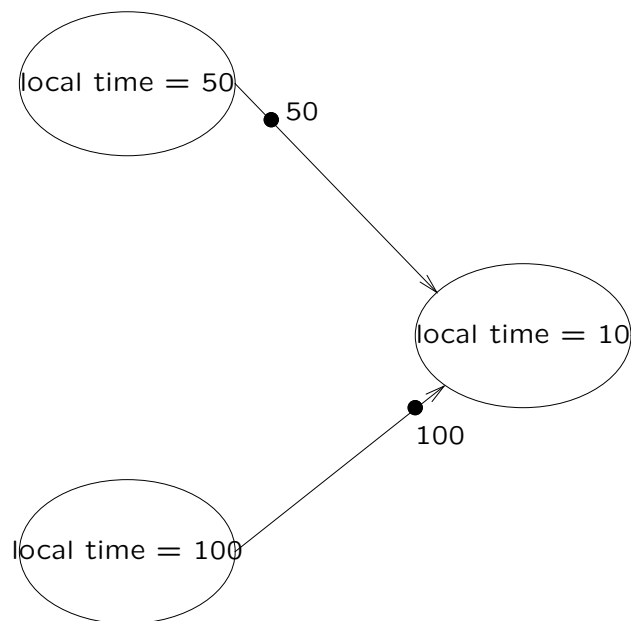


Simulation

- consists of logical processes (LPs)
- communicating sequential processes (CSPs) [Hoare]
- events are timestamped messages



The Fundamental Problem in Parallel Discrete-Event Simulation— Synchronization



Ultra-Conservative Synchronization

- keep all the local clocks synchronized
- parallel execution is guaranteed to be correct
- only available parallelism is due to simultaneous events in different LPs

~> very poor performance



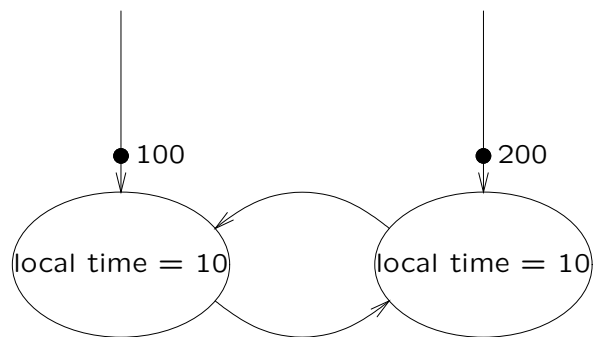
Conservative Synchronization

- each LP has its own local time
- causality is preserved by two rules—*input waiting rule* and *output waiting rule*
- input waiting rule:
 - each LP waits until it has a message in all input arcs before processing the next event
- output waiting rule:
 - an LP cannot send output messages with timestamps greater than the local time
- rules guarantee that consecutive events on each arc carry non-decreasing timestamps

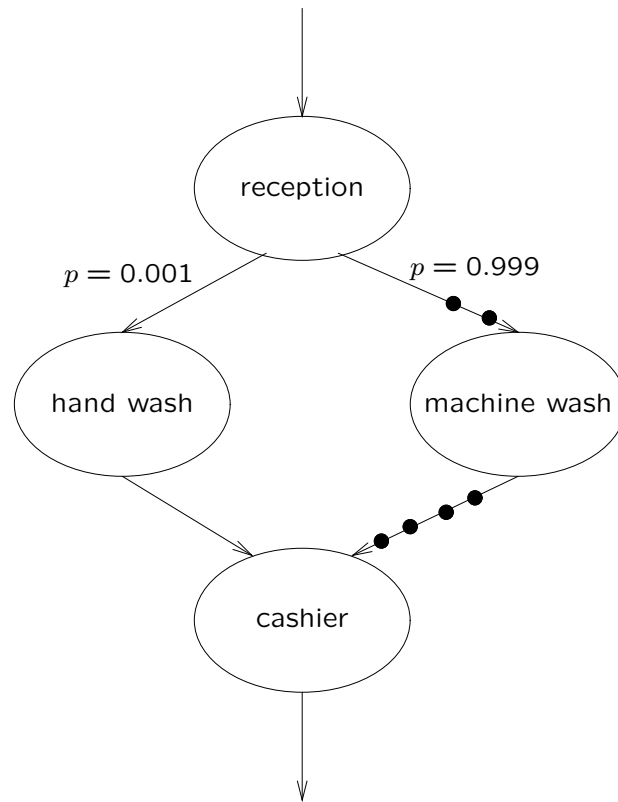


The Deadlock Problem

- conservative synchronization is prone to deadlock
- in particular cycles often lead to deadlock



Deadlock Example 2—Car Wash



Dealing with Deadlock

- deadlock avoidance
 - techniques which prevent deadlock from occurring
 - e.g., *null messages*
- deadlock detection and recovery
 - techniques which detect the occurrence of a deadlock
 - find the next event in the system
 - cause it to be executed



Optimistic Synchronization

- assume events arrive *in order*
- process events in order as soon as they arrive
- what if assumption was wrong?

~> *rollbacks*

- must be able to back up the computation to the point where synchronization was lost and start again!

