CSCI: 4210/6210
Simulation & Modeling

Time Parallel Simulations

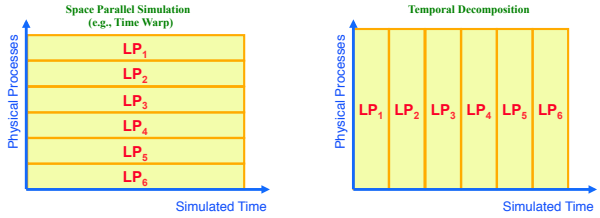
Problem-Specific Approach to Create Massively Parallel Simulations

Outline

- **Introduction**
  - » **Space-Time Simulation**
- **Time Parallel Simulation**
- **Fix-up Computations**
- **Example: Parallel Cache Simulation**

Space-Time Framework

A simulation computation can be viewed as computing the state of the physical processes in the system being modeled over simulated time.

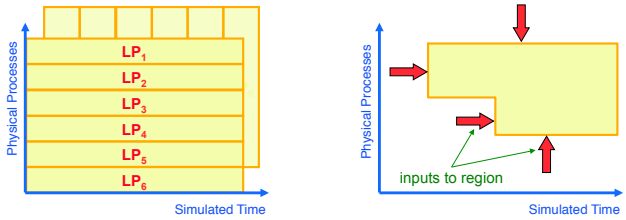


Algorithm:

1. Partition space-time region into non-overlapping regions
2. Assign each region to a logical process
3. Each LP computes state of physical system for its region, using inputs from other regions and producing new outputs to those regions
4. Repeat step 3 until a fixed point is reached

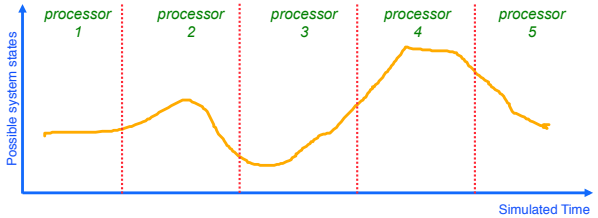
Space-Time Framework

A simulation computation can be viewed as computing the state of the physical processes in the system being modeled over simulated time.



Time Parallel Simulation

Observation: The simulation computation is a sample path through the set of possible states across simulated time.



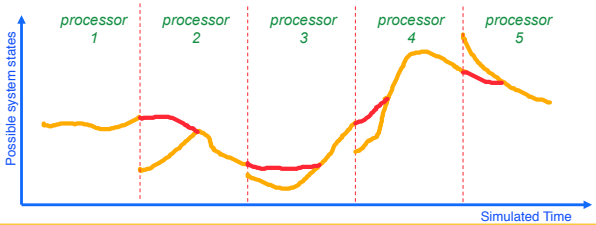
Basic idea:

- **Divide simulated time axis into non-overlapping intervals**
- **Each processor computes sample path of interval assigned to i**

Key question: What is the initial state of each interval (processor)?

Time Parallel Simulation: Relaxation Approach

1. **G**uess initial state of each interval (processor)
2. **E**ach processor computes sample path of its interval
3. **U**sing final state of previous interval as initial state, "fix up sample path
4. **R**epeat step 3 until a fixed point is reached



Benefit: Massively parallel execution (LPs are independent -- no synchronization required between them)

Liabilities: cost of "fix up" computation, convergence may be slow (worst case, N iterations for N processors), state may be complex

Example: Cache Memory

- Cache holds subset of entire memory
  - » Memory organized as blocks
  - » Hit: referenced block in cache
  - » Miss: referenced block not in cache
  - » Cache has multiple sets, where each set holds some number of blocks (e.g., 4); here, focus on cache references to a single set
- Replacement policy determines which block (of set) to delete to make room when the requested data is not in the cache (miss)
  - » LRU: delete least recently used block (of set) from cache
- Implementation: Least Recently Used (LRU) stack
  - » Stack contains address of memory (block number)
  - » For each memory reference in input (memory ref trace)
    - – if referenced address in stack (hit), move to top of stack
    - – if not in stack (miss), place address on top of stack, deleting address at bottom

Example: Trace Drive Cache Simulation

Given a sequence of references to blocks in memory, determine number of hits and misses using LRU replacement

first iteration: assume stack is initially empty:

address:	1 2 1 3 4 3 6 7	2 1 2 6 9 3 3 6	4 2 3 1 7 2 7 4
LRU Stack:	1 2 1 3 4 3 6 7	2 1 2 6 9 3 3 6	4 2 3 1 7 2 7 4
	- 1 2 1 3 4 3 6	- 2 1 2 6 9 9 3	- 4 2 3 1 7 2 7
	- - - 2 1 1 4 3	- - - 1 2 6 6 9	- - 4 2 3 1 1 2
	- - - - 2 2 1 4	- - - - 1 2 2 2	- - - 4 2 3 3 1
	processor 1	*processor 2*	*processor 3*

second iteration: processor i uses final state of processor i-1 as initial state

address:	1 2 1 3 4 3 6 7	2 1 2 6 9 3 3 6	4 2 3 1 7 2 7 4
LRU Stack:	*(idle)*	2 1 2 6 9 7 2 1 2 6 match! 6 7 7 1 2 3 6 6 7 1	4 2 3 1 6 4 2 3 match! 3 6 4 2 9 3 6 4
Done!	*processor 1*	*processor 2*	*processor 3*

Parallel Cache Simulation

- Time parallel simulation works well because final state of cache for a time segment usually does not depend on the initial state of the cache at the start of the time segment
- LRU: state of LRU stack is independent of the initial state after memory references are made to (four) different blocks (if set size is four); memory references to other blocks no longer retained in the LRU stack
- If one assumes an empty cache at the start of each time segment, the first round simulation yields an upper bound on the number of misses during the entire simulation

State Matching Problem Approaches

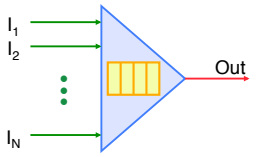
- Fix-up computations
  - » Guess initial state and compute based on guess then re-do computations as needed
  - » Example: LRU cache simulations
- Precomputation of state at specific time division points
  - » Selects time division points at places where the state of the system can be easily determined
  - » Example: ATM multiplexor
- Parallel prefix computation
  - » Example: G/G/1 queue (see text book)

ATM Networks

- Telecommunication technology to support integration of wide variety of communication services
  - » voice, data, video and faxes
- Provides high bandwidth and reliable communication services
- ATM atomic units: ATM messages are divided into *fixed-size cells*

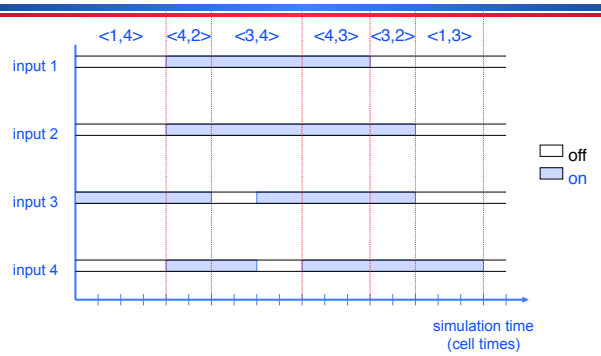
Example: ATM Multiplexer

A multiplexor combines streams into a single output stream



- Cell: fixed size data packet (53 bytes)
- N sources of traffic: Bursty, on/off sources (e.g., voice - telephone)
  - » stream of cells arrive if on
  - » 0 or 1 cell arrives on each input each time unit (cell time)
- Output link: Capacity C cells per time unit
- Fixed capacity FIFO queue: K cells
  - » Queue overflow results in dropped cells
  - » Estimate loss probability as function of queue size (design goal drop 1 in 10^9)
  - » Low loss probability (10^{-9}) leads to long simulation runs!

Burst Level Simulation



Series of time segments: $<A_i, \delta_i>$

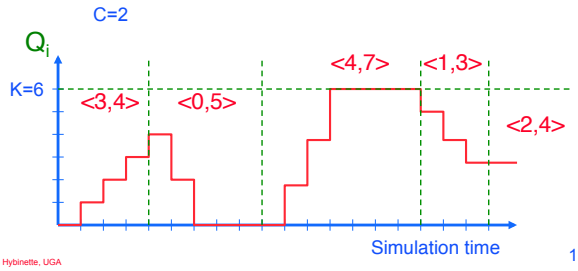
- Fixed number of 'on' sources during time segment
- A_i = # on sources, δ_i = duration in cell times

Problem Statement

- **Multiplexor with N input links of unit capacity**
- **Output link with capacity C (output burst)**
- **FIFO queue with K buffers**
- **Determine average utilization and number of dropped cells**

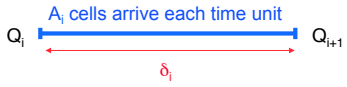
Example

- **Q_i = Number of cells in queue at start of ith tuple**
- **L_i = Number of lost cells at start of ith tuple**
- **Objective: Compute Q_i and L_i for i=1, 2, 3, …**
- **$Q_1 = L_1 = 0$**



Simulation Algorithm

- Generate tuples
- Compute Q_{i+1} and L_{i+1} for each tuple



Observation:
if $A_i > C$, queue is filling (overload)
if $A_i < C$, queue is emptying (underload)

Full

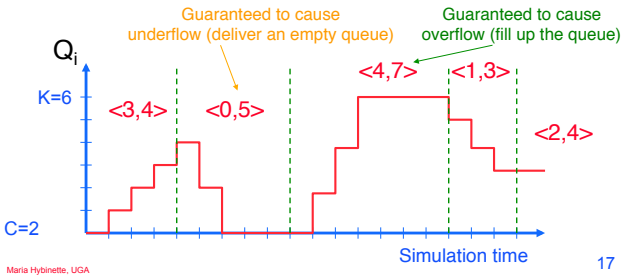
cells added to queue during tuple

Free space in queue at start of tuple

- **Q_{i+1} = if $A_i > C$, then min $[K, Q_i + (A_i - C)\,\delta_i]$**
  else max $[0, Q_i - (C - A_i)\,\delta_i]$
- **L_{i+1} = if $A_i > C$, then L_i + max $[0, (A_i - C)\,\delta_i - (K - Q_i)]$**
  else L_i

Parallel Simulation Algorithm

- **Generate tuples: can be performed in parallel**
- **Q_{i+1} depends on Q_i; appears sequential**
- **Observation:**
  - » **Some tuples guaranteed to produce overflow or empty queue, independent of all other tuples or Q_i at start of the tuple**
  - » **Q_{i+1} known for such tuples, independent of Q_i**



Guaranteed Underflow / Overflow

- **A tuple $<A_i, \delta_i>$ is guaranteed to cause overflow**
  - » **if $(A_i - C)\,\delta_i \geq K$**
  - » **$Q_{i+1} = K$ for guaranteed overflow tuples**
- **A tuple $<A_i, \delta_i>$ is guaranteed to cause underflow**
  - » **if $(C - A_i)\,\delta_i \geq K$**
  - » **$Q_{i+1} = 0$ for guaranteed underflow tuples**

The simulation time line can be partitioned at guaranteed overflow/ underflow tuples to create a time parallel execution

No fix-up computation required

Time Parallel Algorithm

Algorithm

- Generate tuples $<A_i, \delta_i>$ in parallel
- Identify guaranteed overflow and underflow tuples to determine time division points
- Map tuples between time division points to different processors, simulate in parallel

Summary of Time Parallel Algorithms

- The space-time abstraction provides another view of parallel simulation
- Time Parallel Simulation
  - » Potential for massively parallel computations
  - » Central issue is determining the initial state of each time segment
- Applications: Simulation of LRU caches well suited for time parallel simulation techniques
- Advantages:
  - » allows for massive parallelism
  - » often, little or no synchronization is required after spawning the parallel computations
  - » substantial speedups obtained for certain problems: queueing networks, caches, ATM multiplexers
- Liabilities:
  - » Only applicable to a very limited set of problems