Time Parallel Simulations II

ATM Multiplexers and G/G/1 Queues

Richard M. Fujimoto
Professor

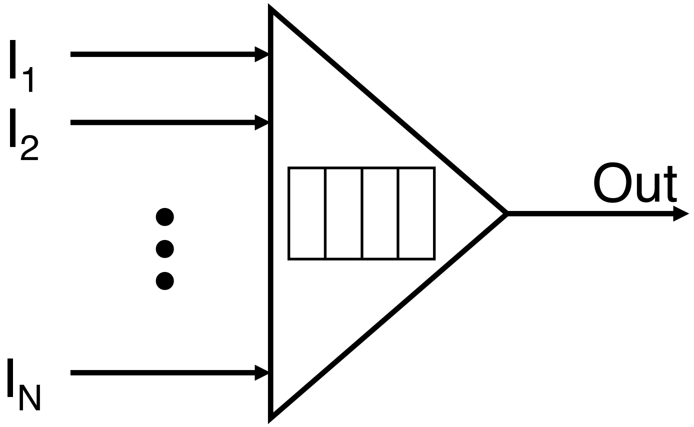
Computational Science and Engineering Division
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0765, USA

http://www.cc.gatech.edu/~fujimoto/

Outline

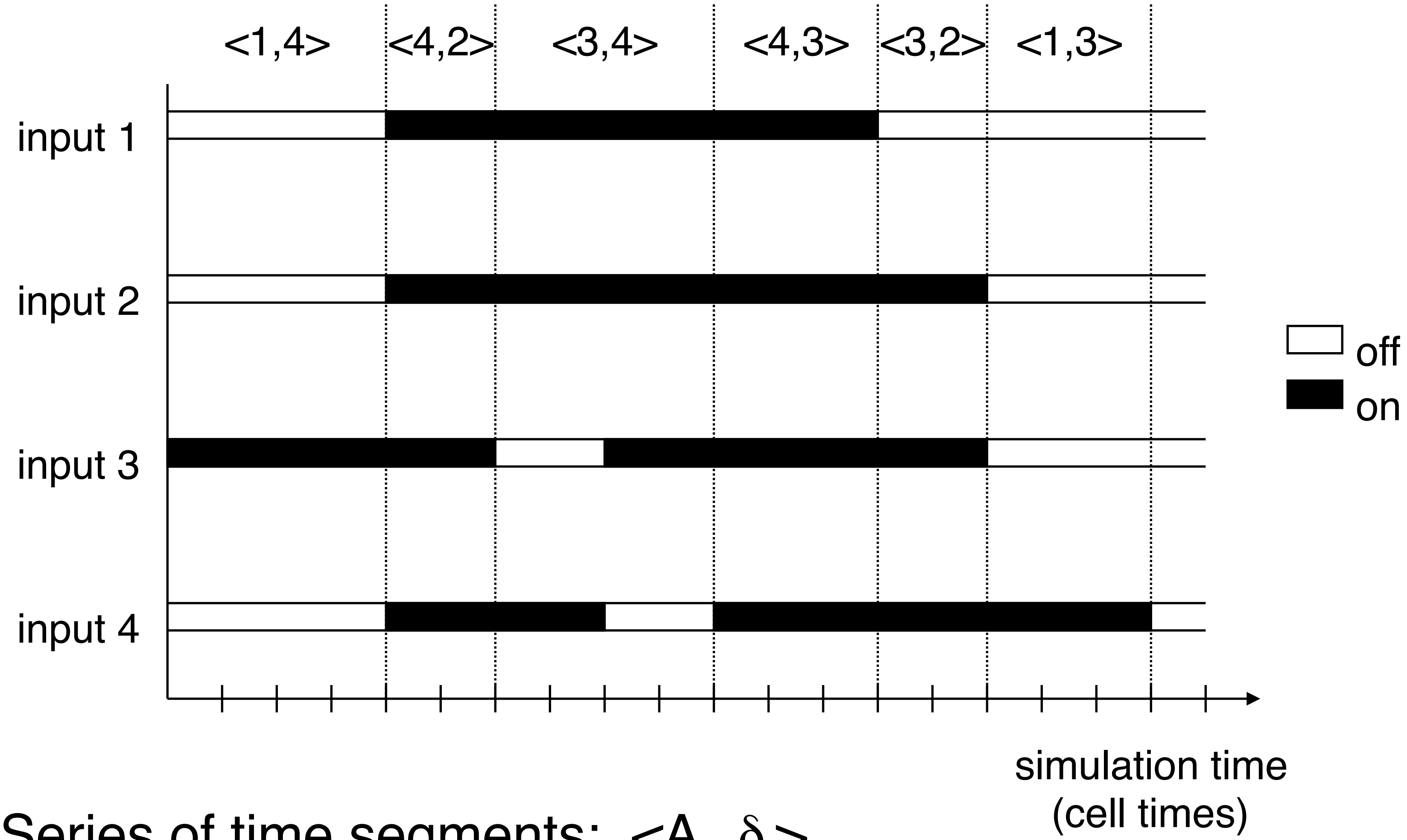
- Time Parallel Simulation using Regeneration Points
  - ATM Multiplexer Simulation
- Simulation Using Parallel Prefix
  - G/G/1 Queue Simulation
- Summary

Example: ATM Multiplexer



- Cell: fixed size data packet (53 bytes)
- N sources of traffic: Bursty, on/off sources
  - 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 cell
- Estimate loss probability as function of queue size
  - Low loss probability (10^{-9}) leads to long simulation runs!

Burst Level Simulation

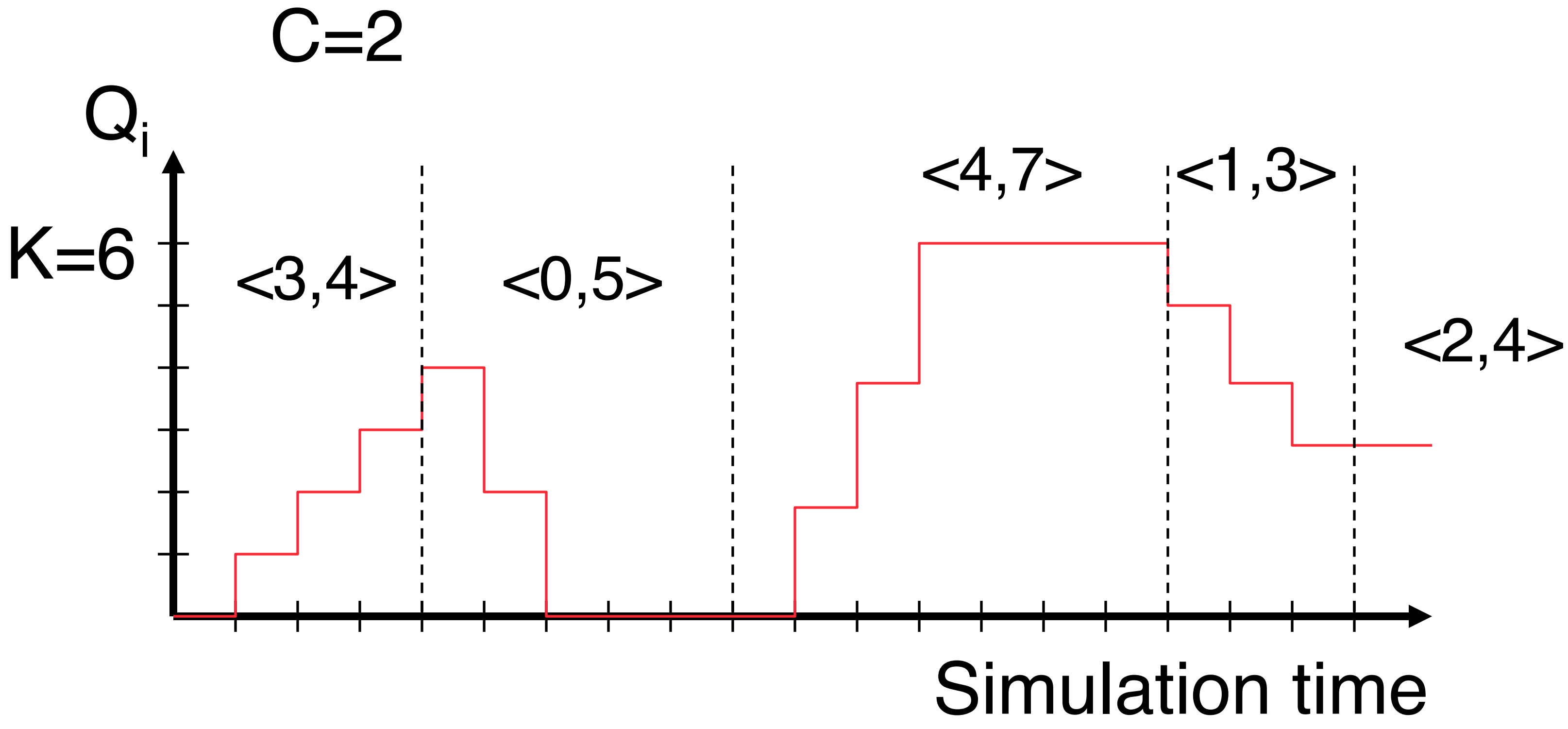


Series of time segments: $<A_i, \delta_l>$

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

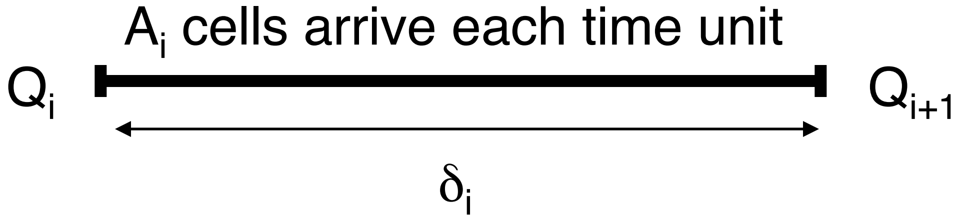
Example

- 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



A_i cells arrive each time unit

Q_i

Q_{i+1}

δ_i

Observation:

if $A_i > C$, queue is filling (overload)

if $A_i < C$, queue is emptying (underload)

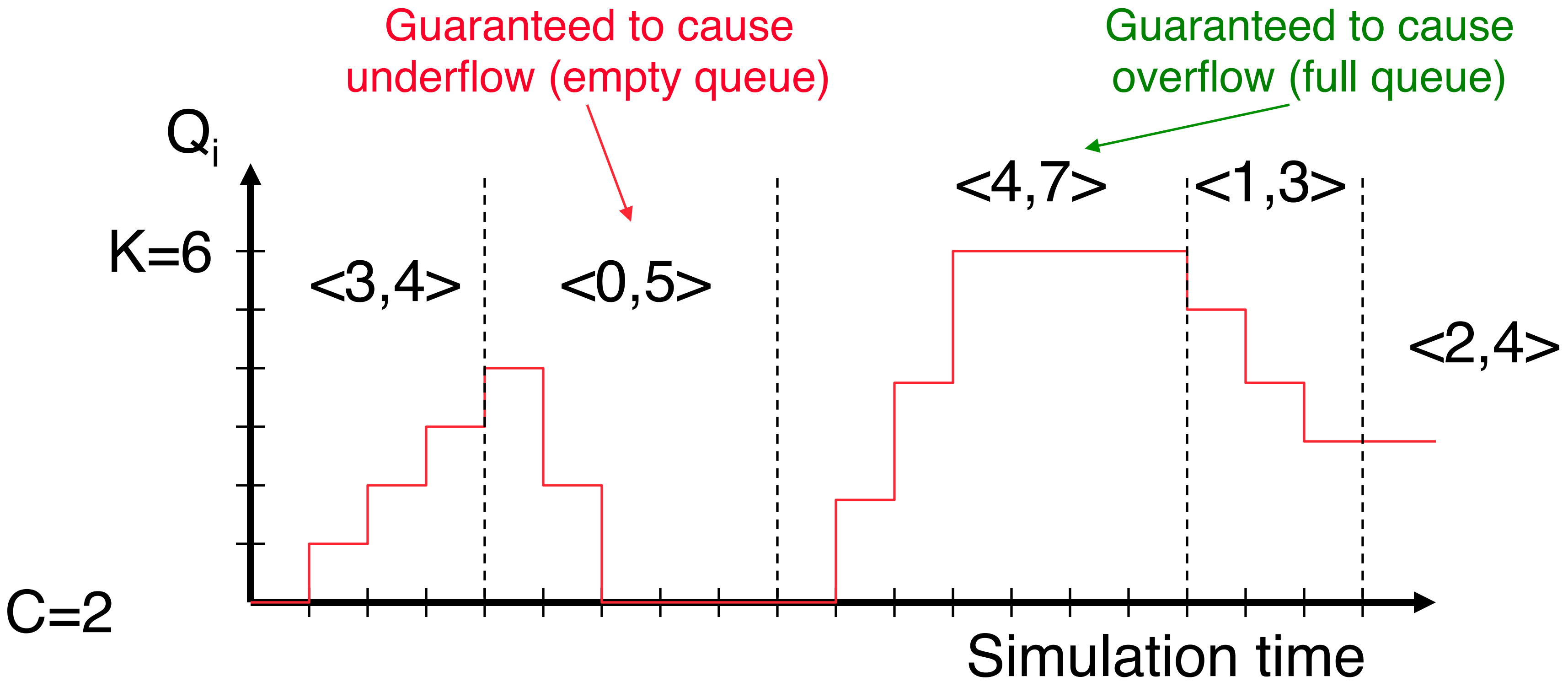
cells added to queue during tuple

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 to cause underflow (empty queue)

Guaranteed to cause overflow (full queue)

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

Guaranteed Overflow/Underflow Points

- We need at least N-1 guaranteed overflow or underflow points to distribute the computation over N processors
- Ideally, would like many more than N points and a (roughly) uniform distribution of the points across the tuples in order to provide flexibility to balance the computation workload across the N processors
- In practice, there are usually many guaranteed underflow points

Outline

- Time Parallel Simulation using Regeneration Points
  - ATM Multiplexer Simulation
- Simulation Using Parallel Prefix
  - G/G/1 Queue Simulation
- Summary

Time Parallel Simulation Using Parallel Prefix

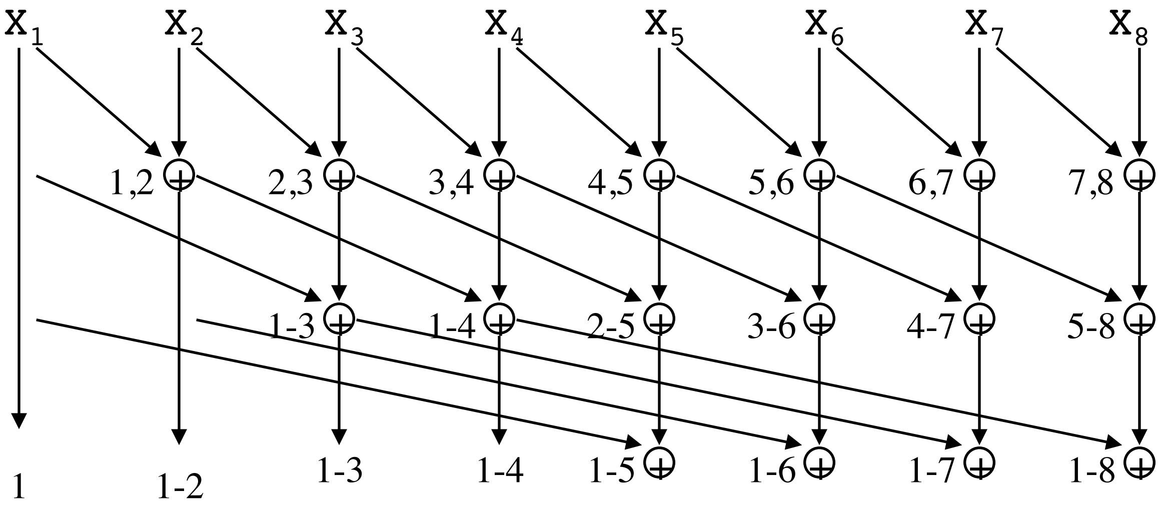
Basic idea: formulate the simulation computation as a linear recurrence, and solve using a parallel prefix computation

parallel prefix: Give N values, compute the N initial products

$P_1 = X_1$

$P_2 = X_1 \cdot X_2$

$P_i = X_1 \cdot X_2 \cdot X_3 \cdot \ldots \cdot X_i$ for $i = 1, \ldots N$; $\cdot$ is associative



add value one position to the left

add value two positions to the left

add value four positions to the left

Parallel prefix requires O(log N) time

Example: G/G/1 Queue

Example: G/G/1 queue (general interarrival time and service time distribution, one server), given

- r_i = interarrival time of the ith job
- s_i = service time of the ith job

Compute

- A_i = arrival time of the ith job
- D_i = departure time of the ith job, for i=1, 2, 3, … N

Solution: rewrite equations as parallel prefix computations:

- $A_i = A_{i-1} + r_i \quad (= r_1 + r_2 + r_3 + \ldots r_i)$
- $D_i = \max(D_{i-1}, A_i) + s_i = \max(D_{i-1}+s_i, A_i+s_i)$

Parallel prefix can be applied to both computations

Summary of Time Parallel Algorithms

Pro:

- 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

Con:

- only applicable to a very limited set of problems