Parallel Discrete Event Simulation

PDES: the execution of a single DES program on a parallel computer

Why PDES?

large simulations consume enormous amounts of time on sequential machines

- engineering
- computer science
- economics
- military applications

PDES is interesting because real-world problems often contain substantial amounts of parallelism yet are difficult to simulate in parallel

Parallel Computing

Parallel computing involves the use of multiple processors (CPUs) to solve a single problem

- reduces execution time
- able to increase scale of the problem

For the purposes of PDES, there are two basic classes of parallel computers

- shared memory: common memory shared by processors
- distributed memory: each processor has its own memory

Distributed memory applications require passing messages along network connections to relay information from one processor to another

Shared memory applications permit processors to access common areas in memory using locks

PDES research exists in both classes

Revisiting Sequential DES

In a DES model

- changes in system state occur only at discrete points in time
- *events* move the simulation model from state to state

We are concerned with *asynchronous* systems

- events are not synchronized by a global clock
- instead, events occur at irregular time intervals
- example: communication network
  - state variables: number of network nodes, length of message queues, status of com-links, etc.
  - events: msg arrival at network node, forwarding msg from one node to another, failure of a node, etc.

For asynchronous systems, parallelism based on lockstep execution performs poorly

Causality of Events

Consider a sequential DES:

- A sequential DES typically uses three data structures
  - the system clock variable
  - the state variables
  - an event list of pending events
- Each event corresponds to a future change in system state
- Each event is time-stamped
- Repeatedly remove and process smallest time-stamped event
  - change system state appropriately
  - schedule new events

Selecting the smallest time-stamped event is crucial!

Otherwise, we simulate a system in which future events can affect the past — **causality errors**

> "... you built a time machine ... out of a DeLorean?"
>
> — Marty McFly

> "The way I see it, if you're gonna build a time machine into a car, why not do it with some style?"
>
> — Doctor Emmet Brown

Why PDES is Hard

Processing events concurrently on different processors is our best opportunity for parallelism

How do we avoid causality errors?

- consider events E_1, E_2 with time-stamps T_1, T_2 $(T_1 < T_2)$
- if E_1 changes a state variable used by E_2, E_1 must be executed first
- if E_1, E_2 are executed on different processors, how do we know when E_2 can be executed?

i.e., *sequencing constraints* are needed for the computation to be correct

This problem is easy in a sequential DES, but quite difficult in parallel

The Logical Process Paradigm

Most PDES strategies forbid processes to have direct access to shared state variables

The Logical Process Paradigm:

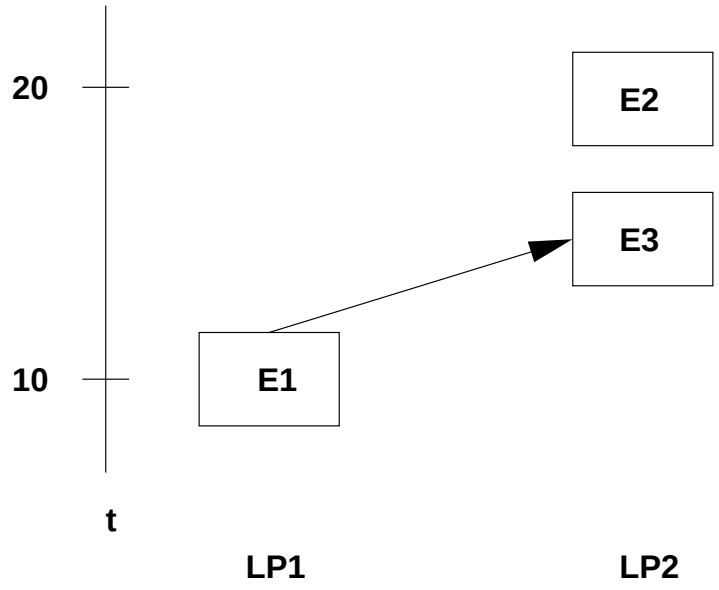
- the *physical system* is composed of *physical processes*
  - the processes interact at various points in time
  - e.g., communication network of switching centers
- the *simulation model* is constructed as a set of *logical processes* (LP_0, $LP_1, \ldots$), one per physical process
- interactions between physical processes are modeled using time-stamped event messages between corresponding LPs
- each LP contains
  - portion of the state corresponding to the physical process it models
  - local clock denoting how far the LP has progressed

The time-stamped event messages are used to control sequencing constraints

Other Causality Errors

Exclusion of shared states does not prevent all causality errors

Consider event E_1 at LP_1 with time-stamp 10 and event E_2 at LP_2 with time-stamp 20



If E_1 schedules a new event E_3 for LP_2 with time-stamp < 20, then E_3 could affect E_2

If there is no information about what events can be scheduled by other events, must process event with smallest time-stamp

Sequential execution of the associated events!

Amdahl's Law: speed-up $\leq k$-fold if $1/kth$ of computation is sequential

Why PDES is Hard (Cont.)

PDES is difficult because the sequence constraints that dictate computation order are complex and highly data-dependent

The simulation must adhere to cause/effect relationships in physical system

Fundamental dilemma of PDES:

- how do we know if E_1 affects E_2 without simulating E_1?
- E_1 can affect E_2 via a complex, time-stamp-dependent set of events

This dynamic nature is the reason for no general solution

In contrast, parallel computation in other areas experiences great success where much is known at compile-time about the computation structure

- e.g., vector operations on large matrices

The Two PDES Camps

There are two basic PDES sequence constraint mechanisms

1. Conservative
   - does not allow causality errors to occur
   - some strategy required to determine when it is safe to process an event
   - no event is processed until all events that could affect it have been processed
2. Optimistic
   - allows causality errors to occur
   - uses *detection and recovery*
   - when a causality error is detected, some rollback mechanism recovers to a state of correct computation

Which approach is better? Depends on the application ...

Conservative PDES

Consider an LP containing event E_1 with time-stamp T_1

- if no smaller time-stamp in the LP, and
- if the LP can determine impossibility of receiving an event with time-stamp $< T_1$,
- then the LP can safely process E_1, otherwise block (wait)

How does the LP determine this impossibility?

- Links between communicating LPs are *statically* specified
- Each link has an associated clock, either
  - time-stamp of first (unprocessed) event in the assoc. queue, or
  - time-stamp of last received msg (if queue is empty)
- LP selects the link with smallest clock
  - if there is a msg in the assoc. queue, process it
  - otherwise, the LP blocks

Can lead to deadlock — multiple LPs blocking

Handling Deadlock in Conservative PDES

Several approaches for handling/avoiding deadlock

- deadlock avoidance using special *null* messages
- deadlock detection and recovery
- look-ahead

  ability to predict what will or will not happen in future
- barrier synchronizations
- conservative time windows
- others

Regardless of the choice above, causally-linked events are *never* processed out of order

Optimistic PDES

Optimistic methods do not avoid causality errors

- process events "optimistically"
- determine when a causality error has occurred
- rollback the system to a correct point in simulated time

Advantages over conservative:

- exploit parallelism where errors *might* occur but do not
- easy to dynamically allocate LPs (static links not required)

The Time Warp mechanism is the most famous protocol

- aggressive cancellation
- lazy cancellation
- lazy reevaluation
- optimistic time windows
- space-time simulation (2D space-time graph)
- others

How a Typical Optimistic Protocol Works

Time Warp mechanism with aggressive cancellation:

- an error is detected when an incoming msg has smaller time-stamp than the local clock
- the offending event is called a *straggler*
- all LPs receiving straggler undo any events processed prematurely
- requires each LP to periodically save state
- may also require sending *anti-messages* to other LPs to undo effect of previous messages

Global Virtual Time (GVT): smallest time-stamp among all processed event messages

- no event with time-stamp $<$ GVT will ever be rolled back
- saved states affected by such events can be discarded

Conservative or Optimistic?

Conservative Pros/Cons

- Works well for problems with good look-ahead
- Results suggest not robust to small changes in application
- Difficult to dynamically create new LPs
- Must be concerned with synchronization mechanism details

Optimistic Pros/Cons

- Poor look-ahead not a show-stopper
- Exploits parallelism that conservative methods can't
- State-saving and rollback time/space costs
- More complex to implement than conservative
- Best hope for *general-purpose* simulation mechanism

PDES Applications / Summary

Applications in areas similar to previous lecture topics:

- battlefield simulations
- communication networks
- biological systems
  - ant foraging
  - sharks world
  - Lyme disease
- digital hardware

PDES can be used to achieve speed-up and/or to model large-scale problems

Two main paradigms:

- Conservative: no causality errors
- Optimistic: causality errors with detection and recovery

The choice of paradigm depends heavily on the problem

Intractability

Intractable problem: one that is not easily solved

Scientists suffer from the curse of dimension, i.e., the more variables (dimensions), the harder a problem is to solve

- compute how a drug candidate will bind to receptor
- given the receptor, solvent, and 8000 atoms in the drug
- because of 3 spatial variables to describe atom position $\Rightarrow$ 24 000 variables!

Problems can have so many variables that no future increase in computer speed will solve them in reasonable time

Can intractable problems be made tractable?

An Integration Example

Consider computing a definite integral (WLOG, assume limits of integration are $0, 1$)

In practice, most integration problems are more complicated than those in calculus books

For these, the simple approaches that we all know will not work

The integral must be approximated numerically

- compute the integrand at finitely many points
- combine the values to produce the answer

Because the integrand is evaluated at discrete points, the integral can only be approximated

Specify accuracy of an approximation by quantifying the error

- error of approximation falls within some threshold ϵ

To guarantee an error $\leq \epsilon$, need global knowledge of integrand

- e.g., evaluate at $x = 0.2, 0.5$ but know nothing in between
- must make assumption to bound the error, e.g., slope is always < 45 degrees

Computational Complexity

Assume that determining integrand values and combining have fixed costs

Computational complexity of evaluating the integral: the minimum cost required to guarantee the approximation is within ϵ of the true value

For one variable (dimension), complexity is inversely proportional to desired accuracy

- i.e., solution can be approximated with cost $1/\epsilon$
- easy

Computational complexity scales exponentially with dimension

In general, we must also consider smoothness r of the function

$r \geq 0$, where $r = 0$ implies least smooth

Many problems have computational complexity $(1/\epsilon)^{d/r}$

- multivariate integration
- surface reconstruction
- PDEs

If ϵ, r are fixed, complexity depends exponentially on dimension

Intractable vs. Unsolvable

Some problems are intractable, i.e., not *easily* solved

Still others are *unsolvable*

cannot compute even an approximation at finite cost

When smoothness parameter $r = 0$, computational complexity becomes infinite

Therefore, for many problems with large number of dimensions, guaranteeing an approximation with desired error is intractable or even unsolvable

Breaking Intractability Using Randomization

Pick points to evaluate at random, rather than deterministically

Computational complexity is then at most on order of $(1/\epsilon)^2$

- problem is tractable even if $r = 0$

Monte Carlo Method: Metropolis and Ulam, 1940s

- evaluate integrand at *Uniformly* distributed points
- mean of these values is the integral approximation
- such randomization makes complexity independent of d

The price for success:

- ironclad guarantee that error $\leq \epsilon$ is lost
- must settle for weaker assurance
- error is *probably* no more than ϵ

Summary

Certain problems cannot be solved in a reasonable amount of time

Randomization make many of these computationally feasible

Must settle for weaker guarantee of error

Not a cure-all

- does not break intractability for surface reconstruction
- can look at average-case error

Still other problems are not even solvable!