Lecture 21: Adaptive Mesh Computations

CS178: Programming Parallel and Distributed Systems

April 18, 2001
Steven P. Reiss

I. Overview

A. This time I want to continue looking at the techniques used in parallel programming

1. Our interest is in solving difficult problems

a) Otherwise why bother with the complexity here

2. Performance is a primary concern

a) These machines are very expensive

B. The basic principles that are emphasized

1. Maximizing the use of the processors

a) By minimizing computation

b) By ensuring the load is balanced

2. Algorithmic techniques

a) Divide and conquer

b) Block partitioning

c) Pipelining communications and data transfer

d) Minimizing interchanges

3. Programming techniques

a) Store/forward point-to-point messaging on some topology

b) Broadcast messages

c) Barriers

d) Reductions (inverse broadcasts)

e) Scatter-gather

C. We've looked at a variety of problems

1. We started with integration (pi)

2. Then we did grids (heat distribution)

3. **Then we did sorting**
4. **Then we did matrix computations**
5. **Then we did tree searching**
6. **Then we did genetic algorithms**

D. Today I want to look at

1. **Monte-carlo methods**
2. **Advanced grid computations**

II. Monte-Carlo Problems

A. Some problems can be looked at probabilistically

1. **Physical simulations with different starting points**
2. **Estimating mathematical functions**
3. **Hill-climbing search techniques**

B. Monte Carlo methods just involve doing lots of trials

1. **Sample set must be statistically valid**
2. **Accumulate information from the trials**

C. Examples

1. **Estimating PI using a circle**

D. Parallelizing

1. **These are very easy to parallelize**
2. **Little communication needed among samples**

III. Advanced Grid Computations

A. Physical modeling of time-dependent processes

1. **We've seen one example :: heat distribution**
2. **Other examples**
   a) Flow problems: Navier-Stokes equations
   b) Weather
   c) Biological simulations
   d) Chemical simulations (flame)
3. **Note that there are two flavors**
   a) Static -- looking for stable solution

b) Dynamic -- where inputs change over time

4. These are viewed as differential equations

a) Equations relating the next time step to the current

(1) Where time steps are infinitely small

(2) Using derivatives

b) Flow equations: $\frac{\partial U}{\partial t} + \nabla \bullet \bar{F} = 0$

(1) U -- vector of conserved variables (density, energy, temperature)

(2) F -- flux in the different timensions

c) Typically don't have closed-form solutions

d) Thus we solve them by iterative methods

5. The typical solution method is to build a grid or mesh

a) This provides a discrete view of the problem

b) Grid can be squares, but generally triangles

(1) Triangles are easier to compute over

(2) Smaller areas

(3) Easier to conform to in 3D

c) We can the translate the equations to provide a way of computing the next state from the current one

(1) For a given time step

(2) Based on the value at the state and its neighbors

d) If we make the time step and grid small enough

(1) We can get an accurate simulation

(2) Essentially find the solution

6. But this is expensive

a) Grids can be large

b) Computations can be costly (matrix computations)

c) Hence we tnd to parallelize this

B. Parallel Grid Computations

1. The basic idea is to divide the grid among the processors

a) Minimize the boundary size

b) Attempt to keep boundaries between neighboring processors to minimize communications

c) Guard regions are often maintained

d) Ensure that the workload is balanced

2. Computation is a heartbeat algorithm

a) Compute-communicate-...

b) Processors work in sync

3. We saw this in the heat flow example

a) We used rows, but we could have used blocks

C. Real-Life problems tend to be messier

1. Flow/heat/weather change more in some places than others

a) Hills/mountains/nozzles/heat sources

b) If the grid is too coarse we miss these

c) The result is an inaccurate simulation

2. Solution -- use a finer grid

a) However computation cost goes up as # of grid elements

(1) Which is square or cube of grid size

b) Actually, it is worse than that

(1) The time step that should be used depends on the grid size (time for action to be felt across)

(2) Thus smaller grids, require proportionally smaller time step

(3) Thus we need to do that many more computations

c) Moreover, we don’t need the extra effort at all places

3. Solution -- use a multi-level grid

a) Work at different levels of granularity over time

D. Multigrid Methods

1. Issues that arise

a) Determining how fine to make the grid

b) Retaining consistency mathematically between grids

c) One way of doing this is to ensure all points are connected

   (1) This can be done using triangles by appropriate splitting

   (2) Example: start with triangle, split into 4 by bisecting end points; split middle into 4 the same way

   (3) Now add points connecting outer edges to new points

d) How to divide regions up -- degeneracy problems

   (1) Split non-central small triangle at this point

2. Merging the two grids

a) From fine to coarser -- use a restriction operator to take into account the fine grid points near coarse ones

   (1) $$\begin{bmatrix} 0 & 1/8 & 0 \\ 1/8 & 1/2 & 1/8 \\ 0 & 1/8 & 0 \end{bmatrix}$$

b) From coarse to fine -- use an interpolation operator to find new values for the fine points

   (1) $$\begin{bmatrix} 1/4 & 1/2 & 1/4 \\ 1/2 & 1 & 1/2 \\ 1/4 & 1/2 & 1/4 \end{bmatrix}$$

3. Alternate between grid levels during solving

a) V cycle - fine ... coarse ... fine

b) W cycle -- 1-2-4-8-4--8-4-2-4-8-4-2-1

4. Solve completely at different levels

a) Full -- 8-4-8-4-2-4-8-4-2-1-2-4-8-4-2-1

b) Note that if you are close to a solution, the system should converge rapidly

5. Issues

a) This doesn’t handle dynamic problems

b) Still might involve excess computation

6. Solution -- use an adaptive grid

a) Vary the grid only where needed

E. Adaptive Grid methods

1. Issues

a) Setting up an initial grid with the right fineness

b) Handling different levels simultaneously

   (1) Can't just do one pass since different levels require different time steps

c) Handling merging solutions at the different levels

d) Load balancing

2. If you know the grid in advance, these can be worked out

a) Mathematically, you take K small steps for each coarse step

b) Between these you do the restriction/interpolation to get the starting/ending points

3. Load balancing is trickier

a) Want to break the grid along coarse boundaries where possible

b) Need to balance actual workloads

4. But the real problems come from dynamic situations

a) The areas that need fine tuning change over time

b) The density of the grid will change as well

F. Dynamic Adaptive Mesh

1. Issues

a) When to change the grid level

b) How to change the grid level

c) Load balancing

2. When is actually relatively easy

a) You can estimate the error in a computation given the time step, range, and current values

   (1) Look at the differences among neighbors

   (2) Look at how much things could change

b) You know what your error tolerance should be

c) If the error is greater than tolerance, you need a finer grid

3. How is a bit trickier

a) Want to work with larger regions -- not single cells

b) Need to ensure non-degeneracy

c) Need to do load balancing

4. Basic Strategy

```
Create an initial mesh partition
For ( ; ; ) {
   Compute over the mesh for K iterations
   If (error > tolerance) then
      Refine the grid where needed
      Repartition the mesh among processors
      Migrate data to appropriate nodes
   fi
next
```

5. Grid refinement

a) Requires global (at least shared) knowledge

b) This is generally done sequentially, but there are parallel approaches

c) Need to avoid degeneracies

d) Also involves grid coarsening as appropriate

6. Repartitioning

a) This again requires global knowledge

b) Need to take global workload into account

7. Data migration

a) Each processor is responsible for the values at the grid points it is processing on

b) Migrating grids between processors requires a transfer of the values from one to the other

8. Notes

a) Often you want to stop every K iterations anyway to get output (graphical or other)

b) Error computation is often a side effect of the normal computation

IV. Next Time

A. You wanted to go over the assignment

1. Come prepared to talk about your solution