Lecture 16

Irregular problems
Announcements

- CSE 260 symposium date and time is set for Friday 12/1 2-5pm
- CSE Holiday party is Friday Dec 1st, staring at 3pm
- Need to shift time or date
Partitioning

• How do we divide up the computation and assign to processors?
• The process is called *decomposition* or *partitioning*
• Related issue: *processor mapping*
Issues in decomposition

• Dynamic or static?
• Functional decomposition
  – Task parallelism
  – Pipelining
• Data decomposition
  – The most common technique in parallel computing
  – Are the data decomposed uniformly or non-uniformly?
Irregular Problems

- In the Jacobi application, computational effort is applied uniformly
- **Irregular** applications apply computational effort non-uniformly
- A load balancing problem arises when partitioning the data

Courtesy of Randy Bank
Irregular representations

• The mesh may be defined non-uniformly in space (and time)
• There may not even be a mesh at all
• We’ll consider partitioning in three types of representations
  – Particles
  – Irregular block structured meshes
  – “Unstructured” meshes (undirected graphs)
Motivating application: the N-body problem

• A classical problem
• Compute trajectories (in time) of a system of N bodies moving under mutual influence
• The bodies can be molecules, planets, stars, charged particles…
Applications

• A **force law** governs the way the particles interact
• No general analytic (exact) solution when $N > 2$
• Numerical simulations needed
• The value of $N$ typically ranges from thousands to millions, depending on the problem
• See animations:
  http://atoms.org.uk/links/
  http://www.dynamical-systems.org/threebody/index.html
The calculation

• Evaluate force field at discrete points in time, called timesteps
  \( \Delta t, 2\Delta t, 3\Delta t, \ldots \)
  – \( \Delta t \) is called the *time step* (a parameter)

• “Push” the bodies according to Newton’s third law
  \[
  F = ma = m \frac{du}{dt}
  \]
Computing the force

• The running time of the computation is dominated by the force computation, so we focus on it.

• The simplest approach is to use the direct method, with a running time of $O(N^2)$

  \[
  \text{Force on particle } i = \sum_{j=0}^{N-1} F(x_i, x_j)
  \]

• $F(\cdot)$ is the force law.

• One example is the gravitational force law

  $G \frac{m_i m_j}{r_{ij}^2}$ where $r_{ij} = \text{dist}(x_i, x_j)$

  $G$ is the gravitational constant.
A simple parallel algorithm

• Each processor is assigned N/P particles
• Processors circulate a copy of the particles – ring broadcast
• After P-1 steps every processor has seen every particle
Localized Force laws

- Many force laws are localized in space
- Gravitational interactions decay as $1/r^2$
- Van Der Waals molecular bonding force (Lennart-Jones)

$$F(r) = \begin{cases} 
  C \left( 2 - \frac{1}{30r} \right) & \text{if } r < \frac{1}{30} \\
  \frac{C}{(30r)^5} & \text{if } r \geq \frac{1}{30}, \\
  C = 0.1 & 
\end{cases}$$
Implementation

- To speed up the search for nearby particles, sort into a *chaining mesh* (Hockney & Eastwood, 1981)
- Compute forces one mesh box at a time
- Only consider particles in the 8 surrounding cells
- Reduces the cost of ignoring distant particles

Jim Demmel, U. C. Berkeley
Communication

- As with stencil methods, use ghost cells to store copies of nearby particles.
- The ghost region also manages particles that have moved outside the subdomain and must be repatriated to their new owner.
Decomposition

- Boxes carry varying amounts of work depending on local particle density
- A uniform partitioning will result in a load imbalance
- To balance the workloads, assign non-uniform regions to processors
- Each non-uniform region of space carries an equal amount of work
- Also preserves spatial locality
- Particles move: partitioning changes over time
Non-uniform blocked decomposition

• A well known partitioning technique is recursive coordinate bisection (RCB)
The workload density distribution

- Irregular partitioning algorithms like RCB require that we estimate the workload in space and time.
- Embed the problem in a discrete space and construct a workload density mapping $\rho(x,t)$, that giving the workload at each point $x=(x,y,z)$ of the discrete problem.
- In static problems the workload density mapping depends only on position.
- In many applications we can come up with a good estimate of the mapping, $W[ ]$. 
Existence of the workload estimate

• Many physical problems exhibit locality in space and time
• The values of the solution at nearby points in space and time are closer than for values at distant points
• Timesteps are “small:” to avoid introducing unacceptable error, the solution changes gradually
• Particle motion is localized in space and in time
• These conditions hold for continuum methods such as classical partial differential equations
An example of RCB in one dimension

• Consider the following loop
  
  for i = 0 : N-1 do
    if ( G(i) )
      then y[i] = f1( .... );
    else y[i] = f2( .... );
  end for

• Assume f1( ) takes twice as long to compute as f2( )
Partitioning procedure

• Let \( W[i] = \textbf{if} \ (G(i)) \ \textbf{then} \ 2 \ \textbf{else} \ 1 \)

\[
1 \ 1 \ 2 \ 1 \ 2 \ 2 \ 2 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 2 \ 2 \ 1
\]

• Compute the running sum (i.e. the \textit{scan}) of \( W \)

\[
1 \ 2 \ 4 \ 5 \ 7 \ 9 \ 11 \ 12 \ 13 \ 14 \ 15 \ 16 \ 17 \ 19 \ 21 \ 22
\]

• Split into 2 equal parts, at \( 22/2 = 11 \)

\[
1 \ 1 \ 2 \ 1 \ 2 \ 2 \ 2 \ \textcolor{red}{1} \ \textcolor{red}{1} \ \textcolor{red}{1} \ \textcolor{red}{1} \ \textcolor{red}{1} \ \textcolor{red}{1} \ \textcolor{red}{2} \ \textcolor{red}{2} \ 1
\]
Recursive coordinate bisection

- Recurse until the desired number of partitions have been rendered
  \[
  \begin{array}{cccccccccccc}
  1 & 1 & 2 & 1 & 2 & 2 & 2 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 1 \\
  \end{array}
  \]

- May be applied in multiple dimensions
Another application: adaptive quantization

- Problem: We are given an image which represents each pixel with a fine quantization of intensities, e.g. 24 bits.
- But a coarser quantization suffices, e.g. 8 bits.
Fast Adaptive Bisection

• Use RCB
• Split the initial region along the longest axis
• Place the two rendered regions on a work list
• Iterate until there are $2^k$ regions
  – Select the region on the work list with the highest objective function
  – This is popularity $\times$ color error (more coming)
  – Split along the longest axis
  – Place the two regions on the work list
Selection criteria

• The popularity of a region is the number of pixels with values in the designated region of the color map
  – We compute most of this information once at initialization time, a kind of histogram
• The color error is the square of the length of diagonal of the bounding box
  – Quantization error is lower than with uniform quantization
• When done, we assign a color to each rendered region: the average of all colors represented by the region, weighted by the frequency of their occurrence
Results

  http://www.icase.edu/Dienst/UI/2.0/Describe/ncstrl.icase/TR-97-29
Load Balancing with space filling curves

- Multidimensional RCB suffers from granularity problems
- We can reduce the granularity with many partitions per processor
- Introduces higher surface to volume effects
- Another approach: spacefilling curves
Partitioning with filling curves

• Maps higher dimensional physical space onto a line
  – Load balancing in one dimension
  – Many types of curves *Hilbert* shown here.

• Creates irregular communication surfaces

• Useful for managing locality: memory, databases
Dynamic repartitioning

• If the spatial workload distribution changes with time, so must the decomposition
• In many numerical problems, the solution changes gradually due to timestep constraints
  – We can reduce the partitioning frequency to reduce the cost of repartitioning
  – We can employ optimization strategies like simulated annealing to incrementally partition the mesh
Load balancing efficiency

- If we ignore serial sections and other overheads, then we may express load imbalance in terms of a **load balancing efficiency** metric.
- Let each processor $i$ complete its assigned work in time $T_i$.
- Thus, the running time $T_P = \text{MAX}(T_i)$.

Define

$$\bar{T} = \sum_i T_i$$

- We define the load balancing efficiency

$$\eta = \frac{\bar{T}}{PT_P}$$

- Ideally $\eta = 1.0$
Block structured representations

• Computational structures comprise a finite union of box-like shapes
  – Multiblock methods
  – structured adaptive mesh refinement
• Irregular data structures carry varying amounts of work
• Complicated spatial relationships among the data
• May respond to data-dependent conditions evolving at run-time
An application of multiblock meshes

Courtesy Mary Wheeler, University of Texas, Austin
A structured adaptive mesh

Courtesy Phil Colella, Lawrence Berkeley National Laboratory
Turbulent flow over a step

Applied Mathematics Group
Center for Applied Scientific Computing, Lawrence Livermore National Laboratory
http://www.llnl.gov/CASC/groups/casc-amg.html
Parallelization

• We update patches in parallel
• Patches communicate periodically, and the data transfers are long: \( n >> n^{1/2} \)
• Ghost cells, irregular communication
• Geometric abstractions simplify the bookkeeping
Region Abstraction

- A geometric calculus to manipulate the regions, e.g. intersection
- KeLP (Baden et al), Titanium (UC Berkeley), CHOMBO (Colella et al), BoxTools and Boxlib (Colella et al)
Aggregate abstractions

- Following KeLP (Fink and Baden)
- XArray: a distributed collection of container objects
- FloorPlan: assignment of blocks to processors data decomposition
- Shape of each block is a Region
- Regions live in a global coordinate system, each with their own origin
Data motion model

- Unit of transfer is the MotionItem, a regular section copy
- Expresses a data dependence between two FloorPlan elements
- A MotionPlan describes a global pattern involving 1 or 2 FloorPlans
- The dependence structure is abstract
  - Doesn’t say how the data are represented
  - Doesn’t say how the data dependencies will be satisfied
MotionPlan construction: a halo update

- Dimension independent
- Topologically independent

FloorPlan2 U;
int NGHOST;
MotionPlan M;
for each U(i) ∈ U
    I = trim( U(i), NGHOST );
    for each U(j) ∈ U, j ≠ i
        Region2 R = I ∩ U(j)
        M.Copy( U(i) ∩ R, U(j) ∩ R)
    end for
end for
The Mover

- Instantiate a Mover object over a MotionPlan and two XArrays (source and destination)
- Executing the Mover satisfies dependencies
  - Application-specific coherence protocol
  - Does not specify the underlying mechanisms used to move the data
- Replaces low level point-to-point messages with high level geometric descriptions of data dependences
Phases of an SPMD computation

• Data Decomposition
  – MPI: Assign work by hand or declare a topology
  – KeLP: form logical groupings of distributed data which may have a many-to-one processor mappings: FloorPlan and XArray

• Set up Data Motion
  – MPI: Write out a low level description: send – receive pairs or persistent communication
  – KeLP: build a MotionPlan and then a Mover using geometric set operations