

Lecture 17

Irregular problems: workload
decomposition, programming models

Announcements

- Quiz in class on Tuesday 11/21
 - You may bring your text book, lecture slides, and handout readers
 - 25 minutes
 - Covers material for lectures 13-16
- A5 is now due Weds 11/22 at 5pm
- Come see my talk at the “Last Lecture Series”
 - Wed 11/22 at 12Noon
 - Gallery A, Price Center
 - Free Pizza and Drinks
- Office hours on Weds 11/22 delayed to 2pm to 3pm

Issues in building a parallel program

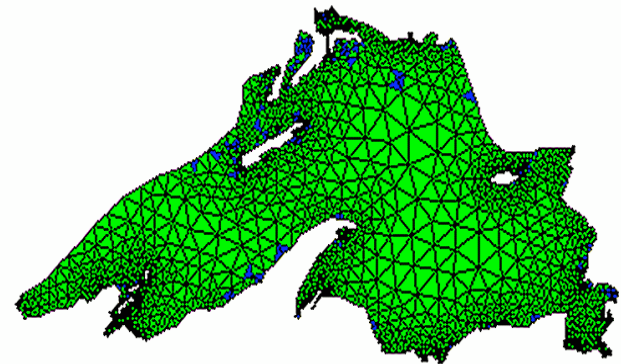
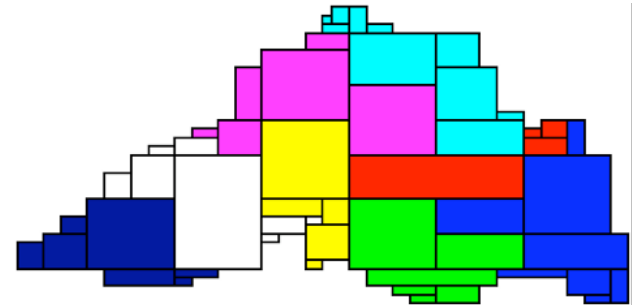
- How do processes communicate?
 - Blocking vs. non-blocking communication
 - Collective vs. point-point communication
 - Granularity: collect data into long messages rather than sending piecemeal
- How do we divide up the computation and assign to processors?
 - *Decomposition* or *partitioning*
 - *Processor mapping*
 - Dynamic or static?

Issues in data decomposition

- Functional decomposition
 - *Task parallelism*
 - *Pipelining*
- Data decomposition
 - The most common technique in parallel computing
 - Are the data decomposed uniformly or non-uniformly?

Irregular Problems

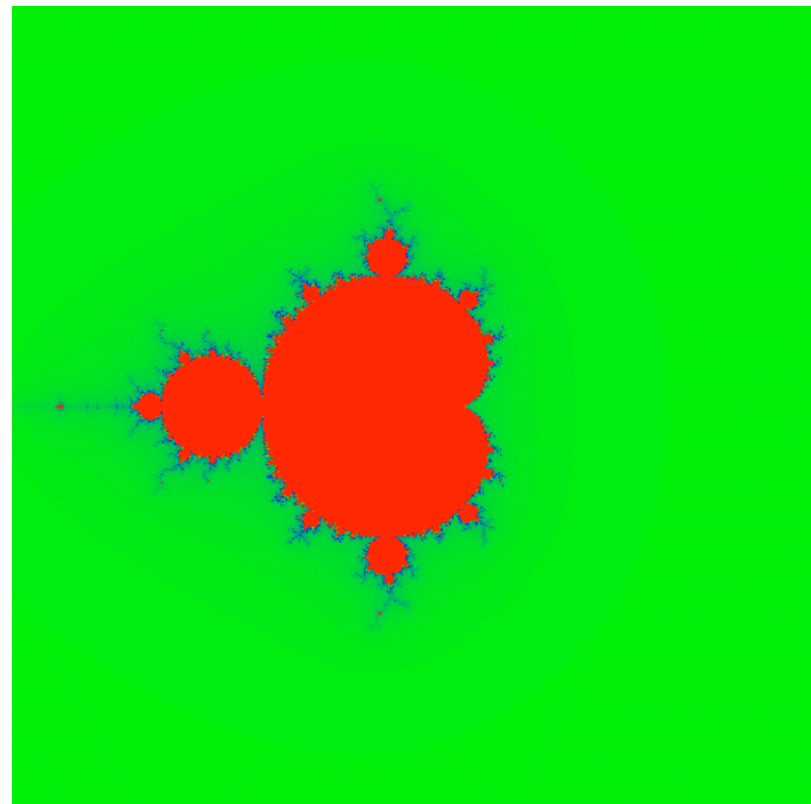
- In the Jacobi application, computational effort is applied uniformly
- **Irregular** applications apply computational effort non-uniformly
- A load balancing problem arises when partitioning the data



Courtesy of Randy Bank

Dynamic non-uniform workload assignment

- Compute the Mandelbroit set for a region of the complex plane
- A quick review of complex numbers



A quick review of complex numbers

- We define $\mathbf{i} = \sqrt{-1}$
- A *complex number* $z = x + \mathbf{i} y$
 - x is called the *real part*
 - y is called the *imaginary part*
- We associate each complex number with a point in the x - y plane
- The magnitude of a complex number is the same as vector length: $|z| = \sqrt{x^2 + y^2}$
- $z^2 = (x+\mathbf{i}y)(x+\mathbf{i}y) = (x^2 - y^2) + 2xy\mathbf{i}$

The Mandelbrot set

- Named for the mathematician B. Mandelbrot
- For which points c in the complex plane does the following iteration remain bounded?

$$z_{k+1} = z_k^2 + c$$

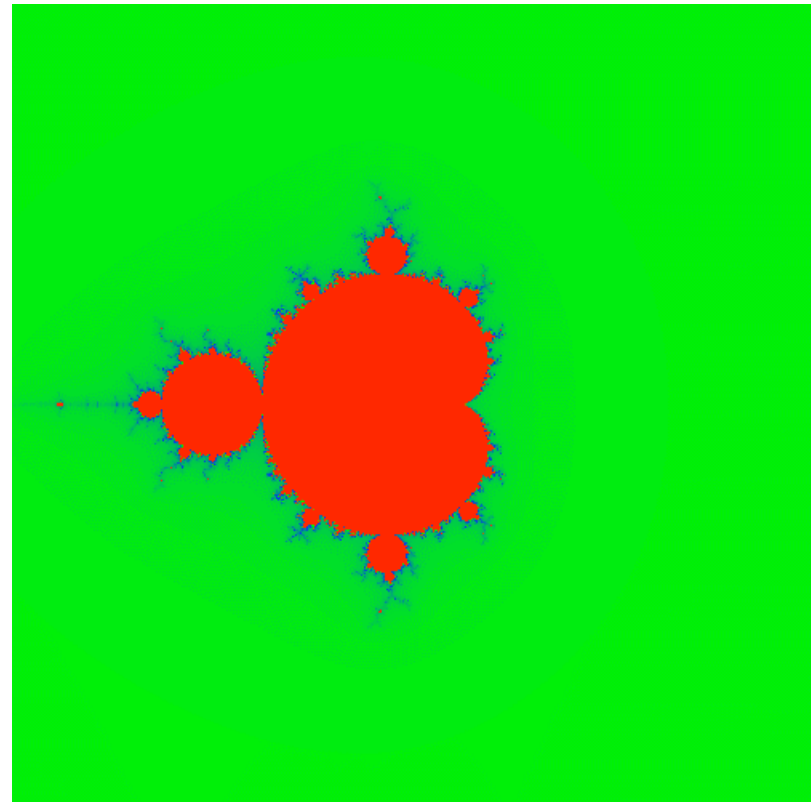
c a given complex constant

$$z_0 = 0$$

- When $c=0$, all points lay within a unit disk : $|z| \leq 1$
- If $|z| \geq 2$, the iteration is guaranteed to diverge to ∞

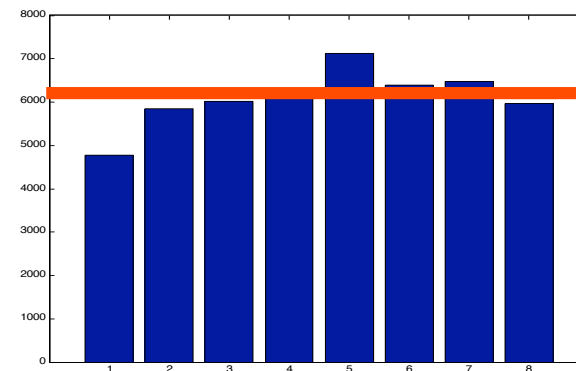
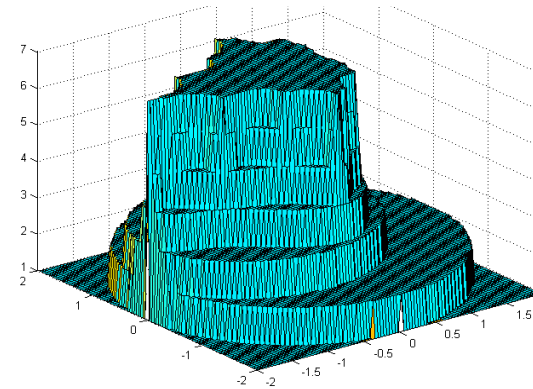
The computation

- Plot the rate at which points in a given region diverge
- Stop the iterations when $z_{k+1} \geq 2$ or k reaches some limit
- Plot k at each position and associate colors with k



A load balancing problem

- Some points iterate **longer** than others
- If we use block decomposition, some processors finish later than others
- We have a *load imbalance*



Cyclic decomposition

- Split up the complex plane into many small uniform blocks of size $b \times b$
- Assign the data in round robin fashion to processors
- All processors should get about the same amount of work if we choose a good value of b
- What are the tradeoffs?

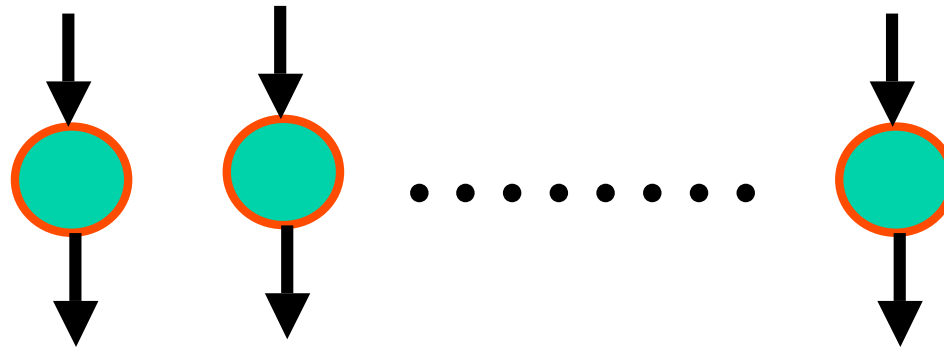
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3

Cyclic decomposition in context

- With cyclic decomposition the workload is statically partitioned in a uniform manner and statically assigned to processors
- The assignments and partitioning do not change
- Cyclic decomposition relies on statistical properties of the workload distribution to even out the workload assignments
- Sometimes a dynamic assignment is more effective
- When might this be the case?

Some observations

- In the Mandelbrot set computation tasks don't communicate with one another
- Input and output are small relative to the typical range of task execution times
- We call such computations **embarrassingly parallel**

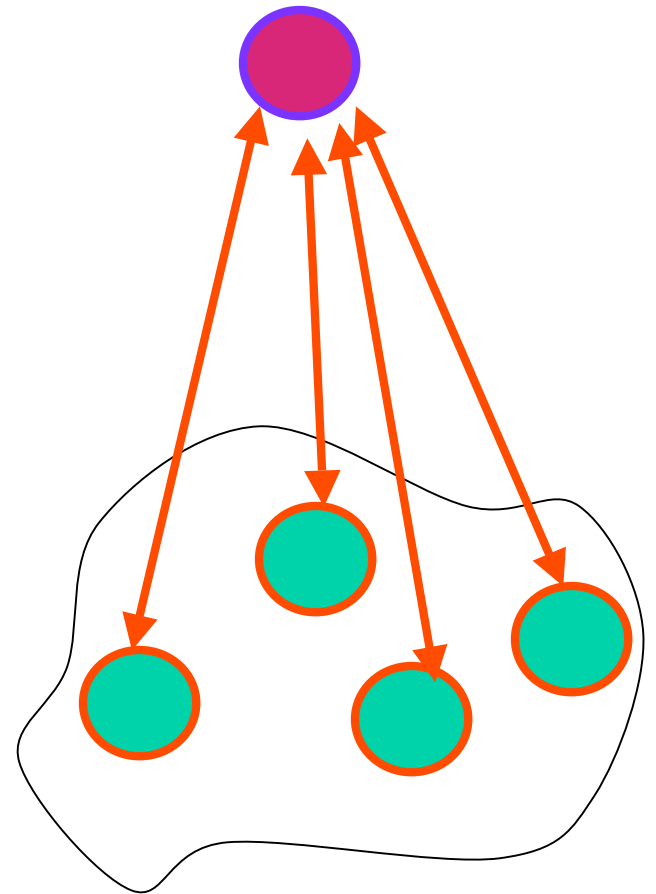


Another approach to dynamic assignment

- In some cases we can't assign the workload in advance because the available processing resources may change, or the description of the workload may change
- The tasks may communicate a small amount of information
- We'll next look at another type of dynamic assignment strategy: client/server

Client server

- We assume that the workload is broken up into small pieces or *tasks*
- Let one of the processors serve as the boss
- The boss generates a description for each block in turn and sends to an available worker processor
- When a worker finishes, it sends to the boss the result(s) and awaits new work



Termination

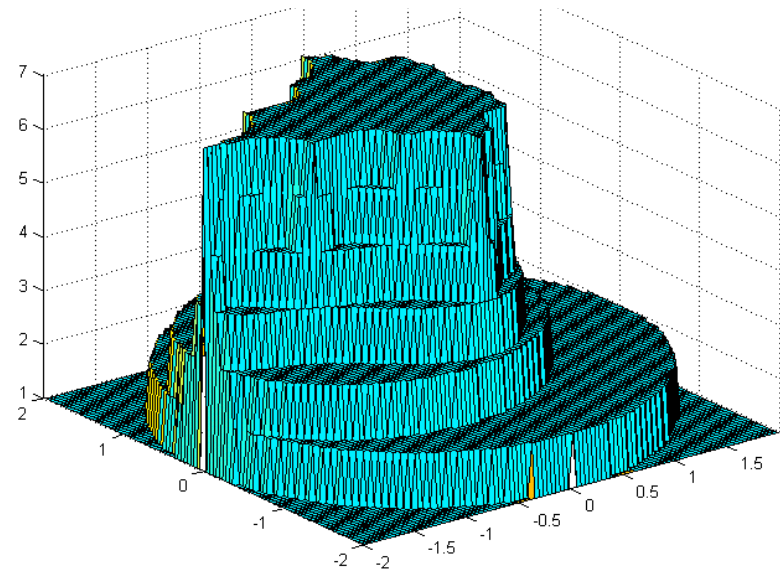
- When there is no more work to do, boss sends a null work order to the worker
- This causes the worker to terminate
- All workers must receive a termination signal

Benefits of client-server

- Client-server allows us to handle time varying processor pool sizes
- Workers receive work on demand at a rate determined by the speed of their computations ...
- ensuring that workloads are balanced

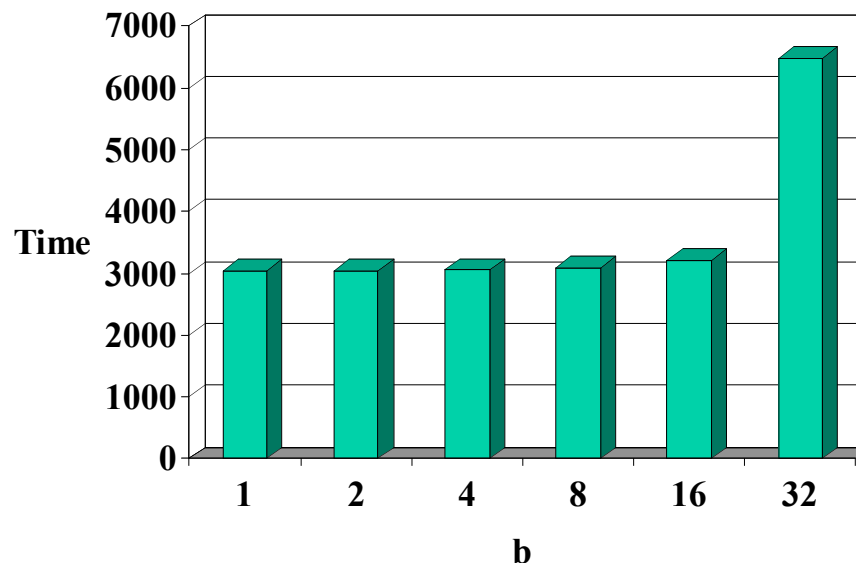
Granularity tradeoff

- We need to choose an appropriate task granularity
- The finest granularity: each point is a separate task
- Coarsest granularity: one block per processor
- What is the tradeoff?



Simulations

- Let's simulate the workload distribution
- Assumptions:
 - Work distribution is instantaneous (zero cost)
 - Running time is proportional to k
- Vary the chunk size b
- Optimal running time on 16 processors: 3044

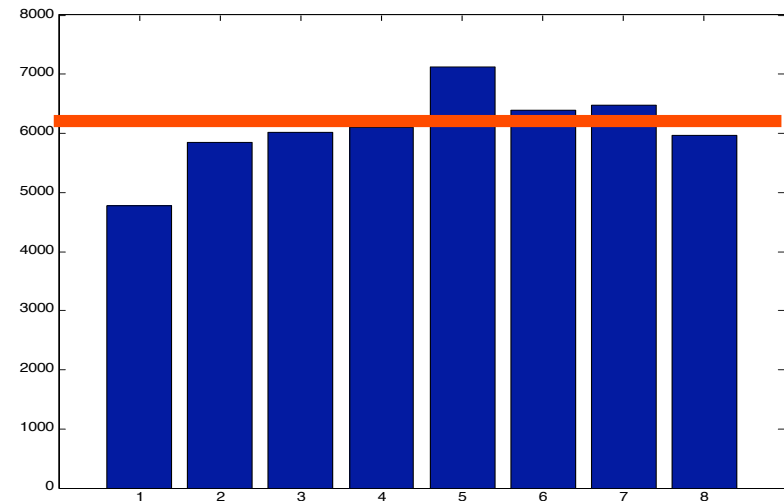


Limits to performance

- What happened when the chunk size was too large?
- Consider $P=8$
- The running times on the processors:

4771	5855	6018	6101
7129	6390	6470	5964

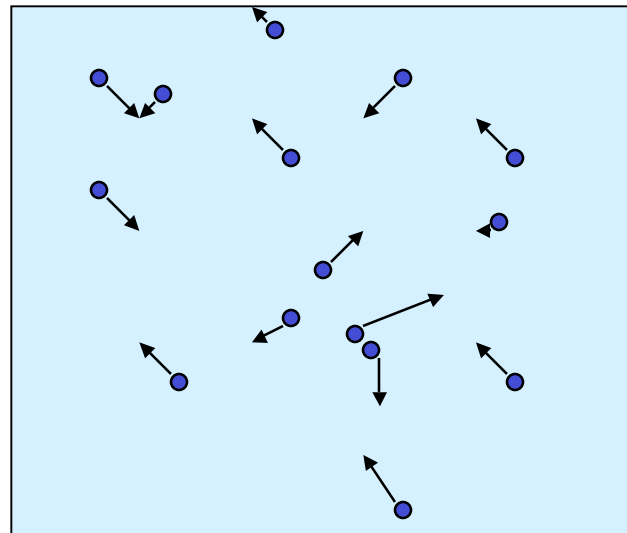
- The optimal running time is 6088



Other kinds of irregular decomposition
techniques:
particle methods

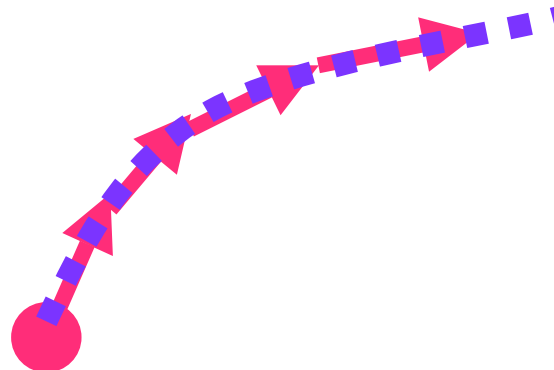
A non-uniform application: the N-body problem

- A classical problem
- Compute trajectories (in time) of a system of N bodies, moving under mutual influence
- A **force law** governs the way the particles interact



The calculation

- Evaluate force field at discrete points in time, called timesteps
 $\Delta t, 2\Delta t, 3\Delta t, \dots$
 - Δt is called the *time step* (a parameter)
- “Push” the bodies according to Newton’s third law
 $\mathbf{F} = m\mathbf{a} = m \, d\mathbf{u}/dt$



Computing the force

- The running time of the computation is dominated by the force computation, so we ignore the push phase
- The simplest approach is to use the direct method, with a running time of $O(N^2)$

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

- $F()$ is the **force law**
- One example is the **gravitational force law**

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

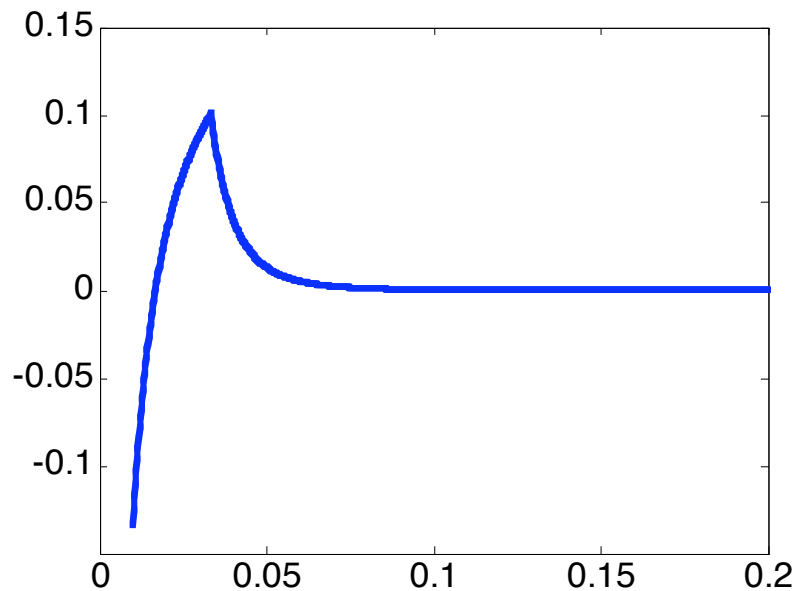
G is the **gravitational constant**

Significance of locality

- Many scientific problems exhibit both **spatial** and **temporal locality**
- Due to the underlying physics
- Gravitational interactions decay as $1/r^2$
- Numerically speaking, timesteps are “small:” the solution changes gradually
- Let’s consider a more highly localized force

Localized force

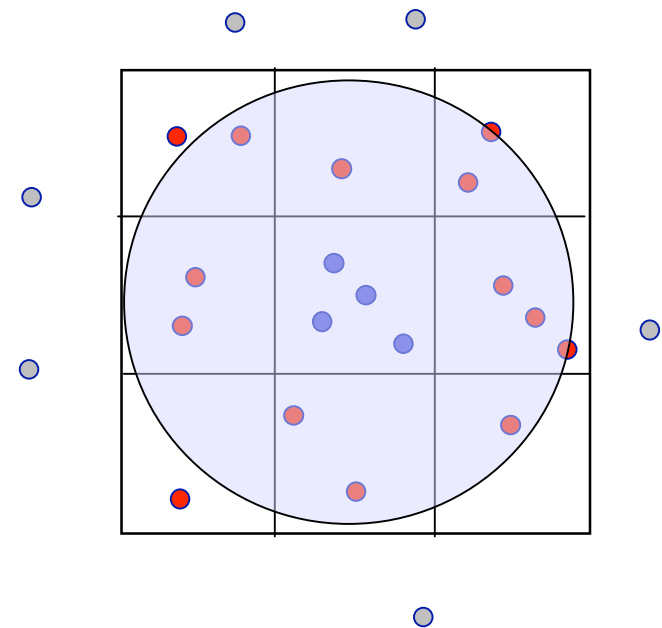
- Many force laws are localized in space
- Gravitational interactions decay as $1/r^2$
- Van Der Waals molecular bonding force (Lennart-Jones)



$$\begin{aligned} F(r) &= C \left(2 - \frac{1}{(30r)} \right) & r < 1/30 \\ & \quad C / (30r)^5 & r \geq 1/30, \\ & & C = 0.1 \end{aligned}$$

Implementation

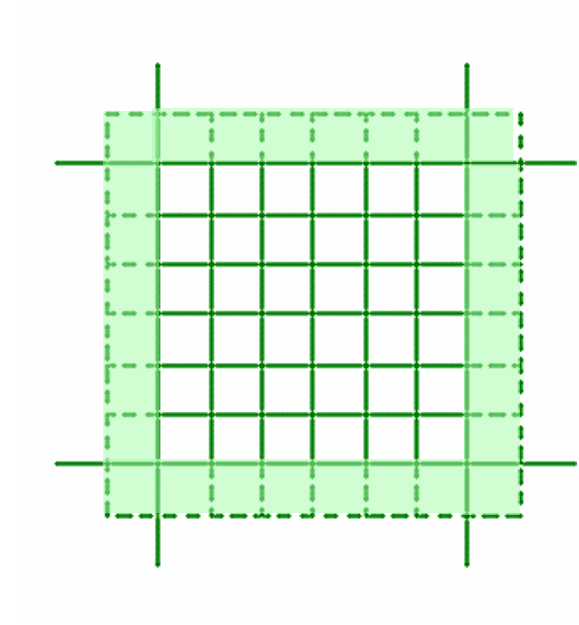
- To speed up the search for nearby particles, sort into a *chaining mesh* (Hockney & Eastwood, 1981)
- Compute forces one mesh box at a time
- Only consider particles in the 8 surrounding cells
- Reduces the cost of ignoring distant particles



Jim Demmel, U. C. Berkeley

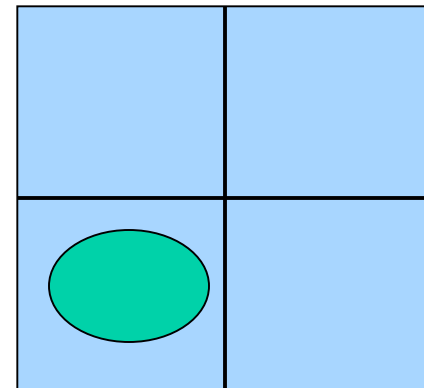
Communication

- 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



Load balancing

- Each box carries around a different amount of work depending on the local density of particles
- A uniform (BLOCK) partitioning will result in load imbalance
- CYCLIC Decomposition may not be appropriate either: What is the difficulty?



Jim Demmel, U. C. Berkeley

Communication overhead

- With a $b=2$, each processor must obtain all the simulation data
- With $b=4$, this factor decreases to $1/2$
- If the dependence distance increases, so must the granularity

The image shows a 16x16 grid with a 4x4 subgrid of 4x4 blocks. The top-left and bottom-left blocks contain numbers 0-15 in a 4x4 grid. The top-right block contains numbers 0-15 in a 4x4 grid, with the first column (0-3) in blue and the other three columns (4-7) in orange. The bottom-right block is empty.

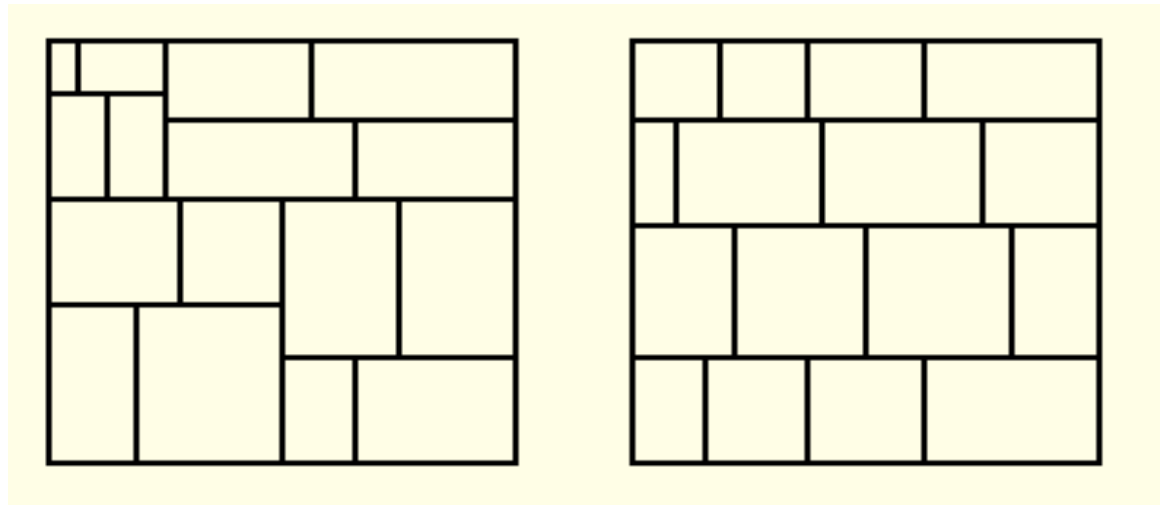
Jim Demmel, U. C. Berkeley

Non-uniform partitioning

- Cyclic partitioning is a static strategy
 - Partitions space uniformly and maps work irregularly to processors in order to balance the workloads
 - A statistical sampling method which does not attempt to measure load imbalance
 - May incur high communication overheads due to fine grain data decomposition
- Another approach is to employ partitioning rather than mapping to mitigate load imbalance
- We partition the work non-uniformly according to the spatial workload density and assign one task to a processor
- Avoids high surface to volume ratio of cyclic decomposition

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 $\rho(\mathbf{x},t)$, that giving the workload at each point $\mathbf{x}=(x,y,z)$ 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 $W[i] = \text{if } (G(i)) \text{ then } 2 \text{ else } 1$

1 1 2 1 2 2 2 1 1 1 1 1 1 2 2 1

- Compute the running sum (i.e. the **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 $22/2 = 11$

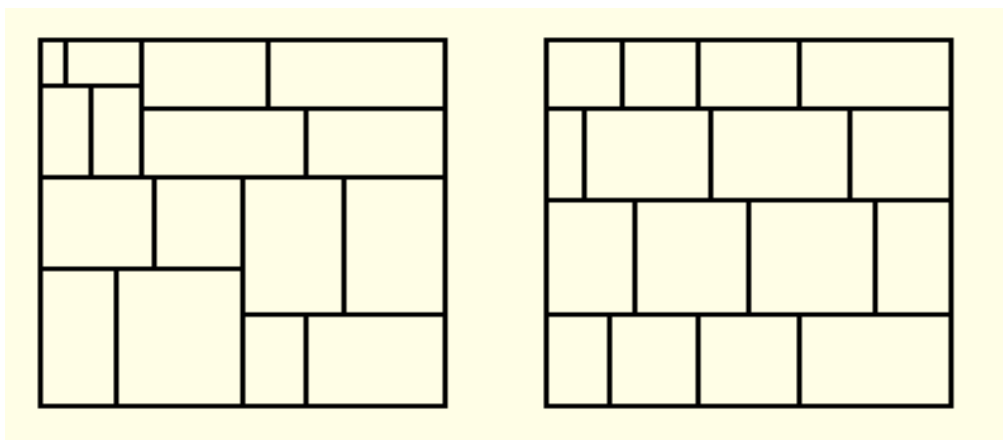
1 1 2 1 2 2 2 1 1 1 1 1 1 2 2 1

Recursive coordinate bisection

- Recurse until the desired number of partitions have been rendered

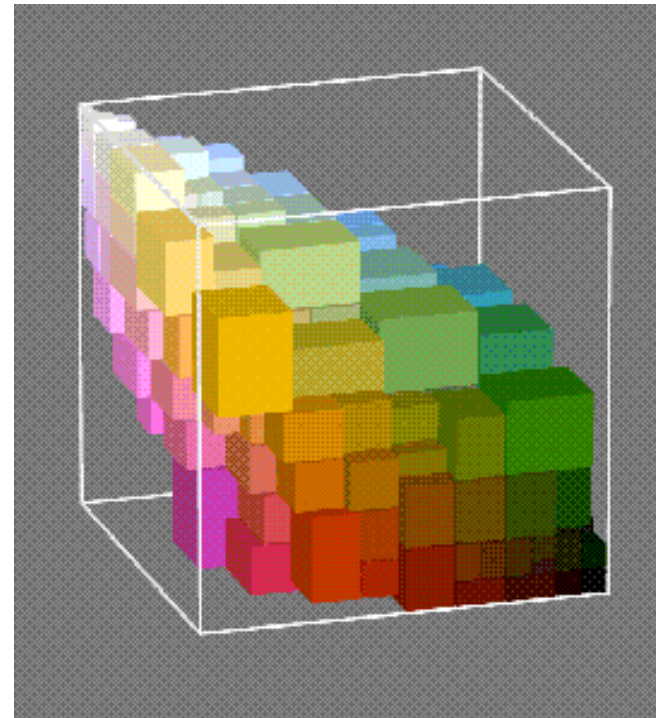
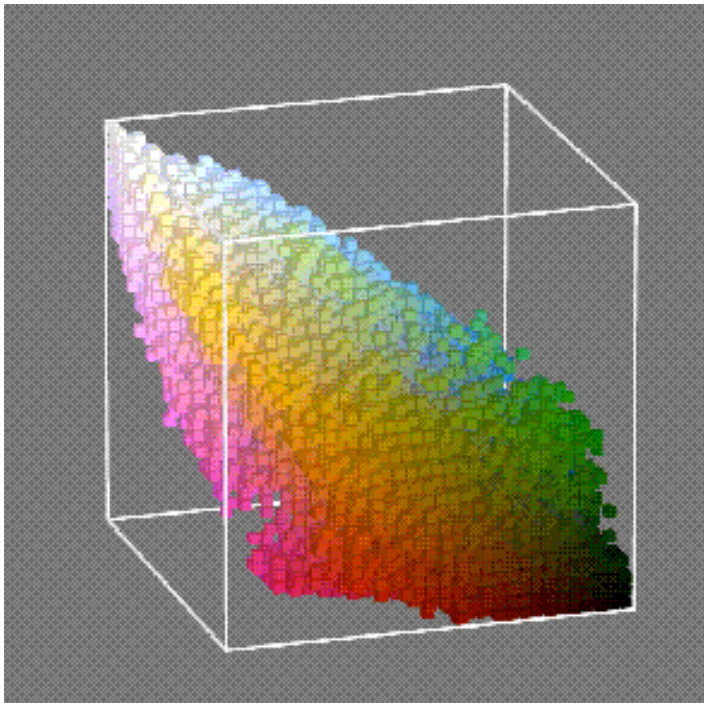
1 1 2 1 2 2 2 1 1 1 1 1 1 2 2 1

- 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

- “Binary Dissection: Variants & Applications,” S. H. Bokhari, T. W. Crockett and D. M. Nicol, TR-97-29, July 1997, <http://www.icasel.edu/Dienst/UI/2.0/Describe/ncstrl.icasel/TR-97-29>



15 bits—uniform

11/17/06



8 bits—adaptive

Scott B. Baden / CSE 160 / Fall 2006



8 bits—uniform

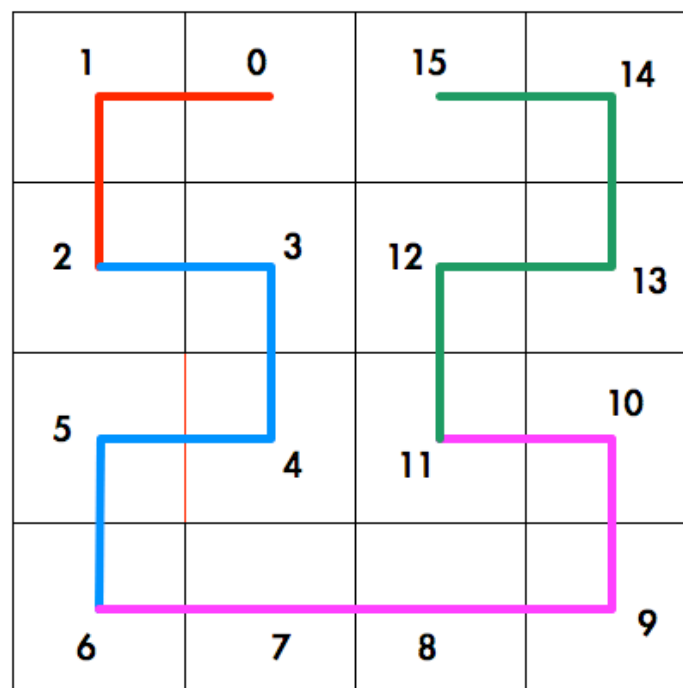
41

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 $T_P = \text{MAX} (T_i)$

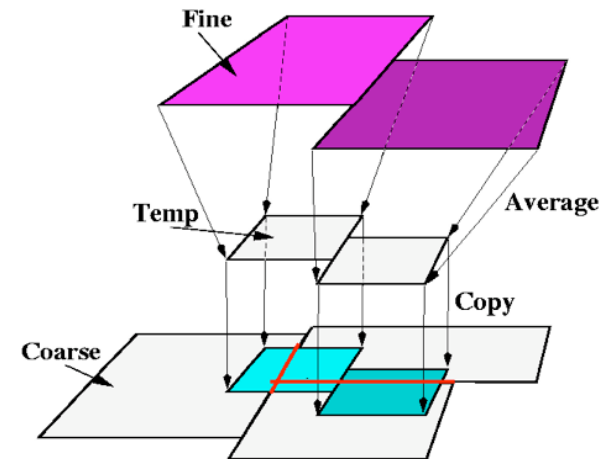
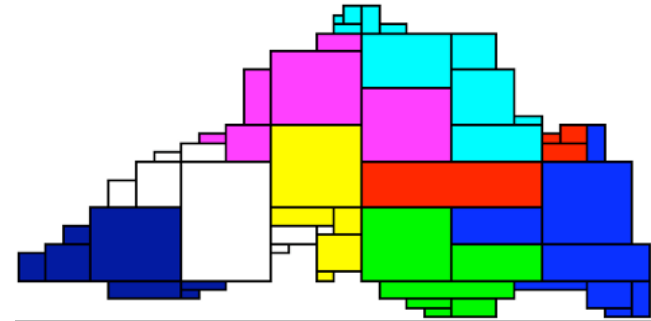
- Define $\bar{T} = \sum_i T_i$

- We define the **load balancing efficiency** $\eta = \frac{\bar{T}}{PT_P}$

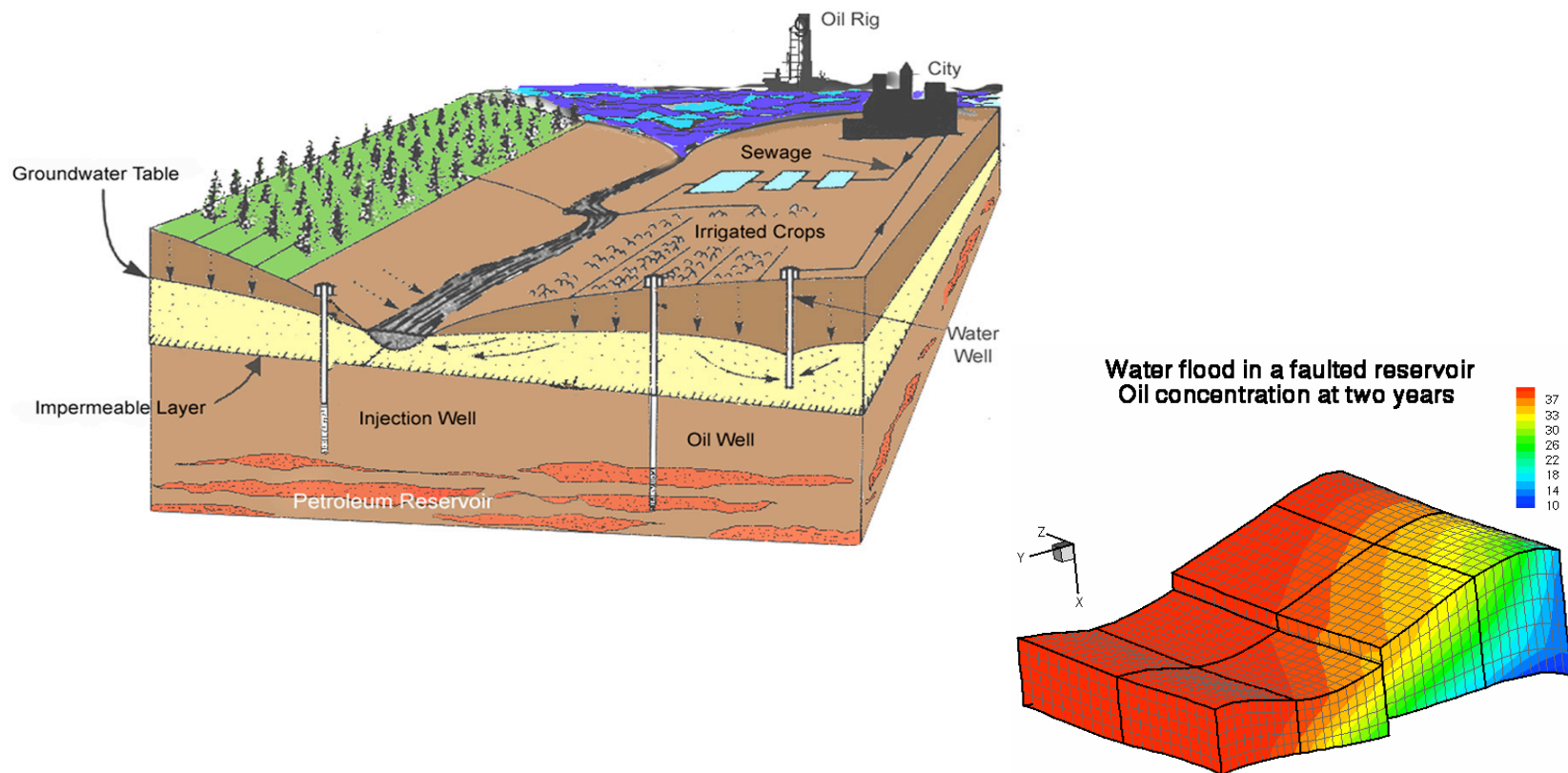
- Ideally $\eta = 1.0$

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

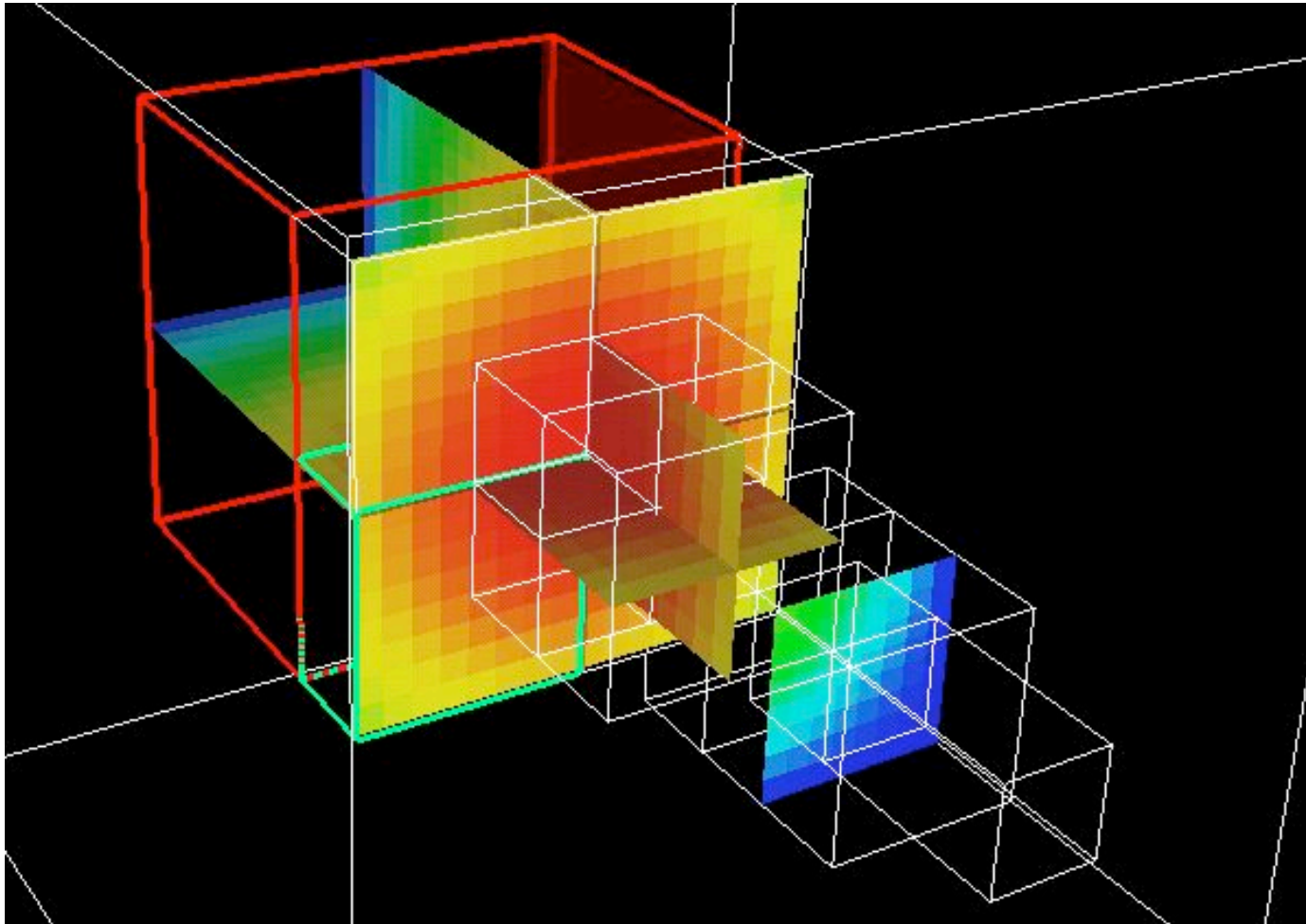


An application of multiblock meshes



Courtesy Mary Wheeler, University of Texas, Austin

A structured adaptive mesh



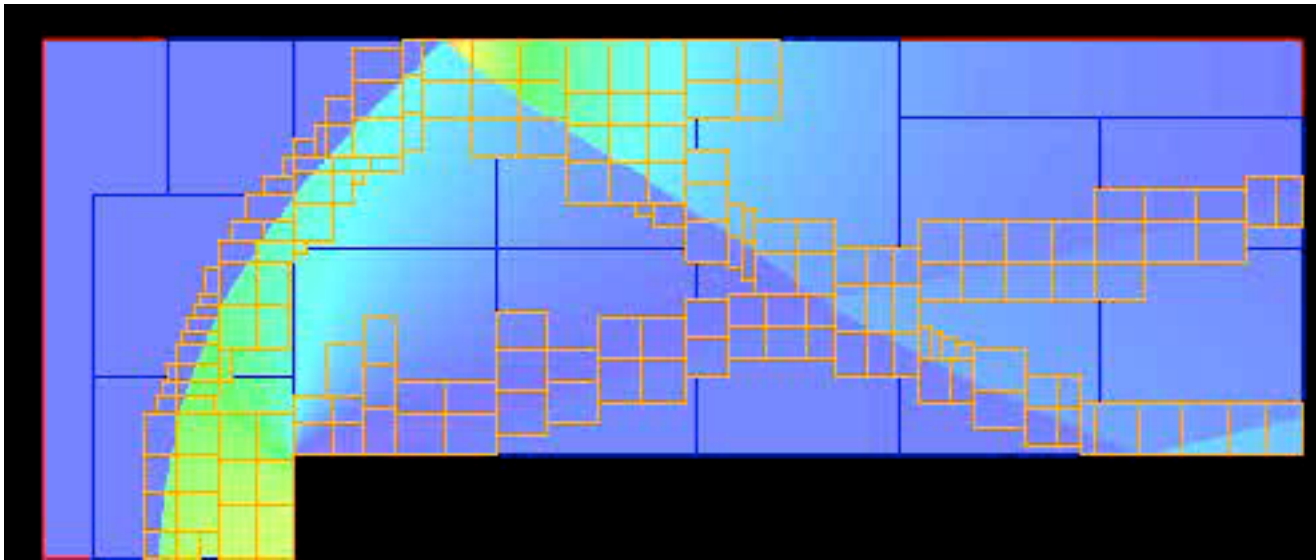
Courtesy Phil Colella, Lawrence Berkeley National Laboratory

11/17/06

Scott B. Baden / CSE 160 / Fall 2006

48

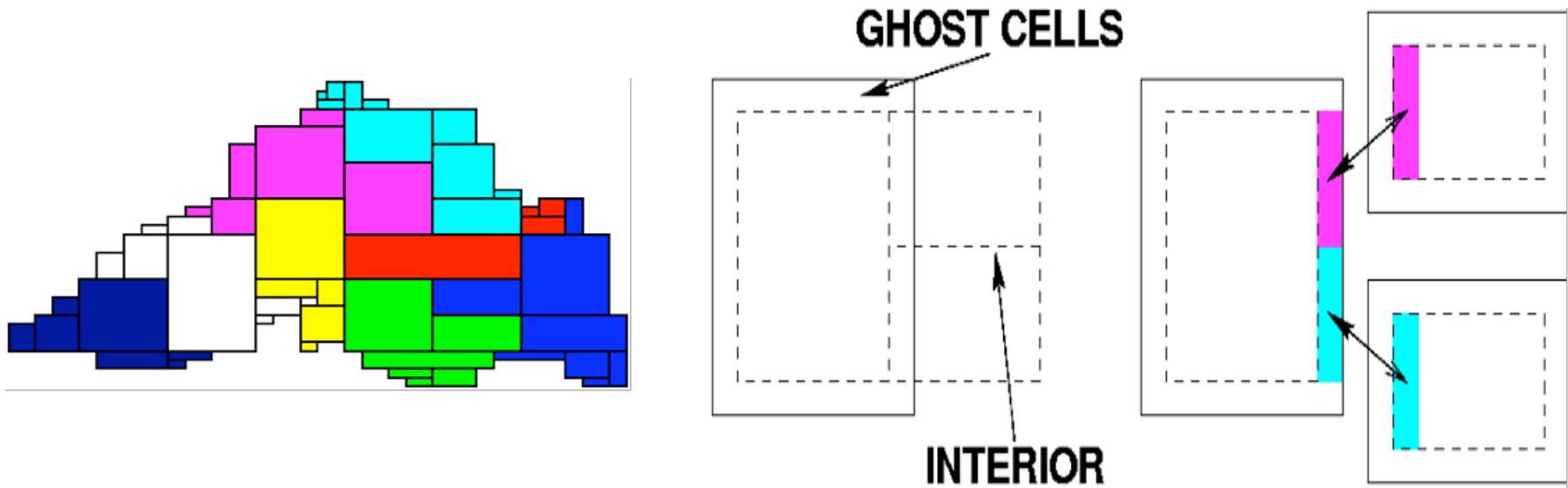
Turbulent flow over a step



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

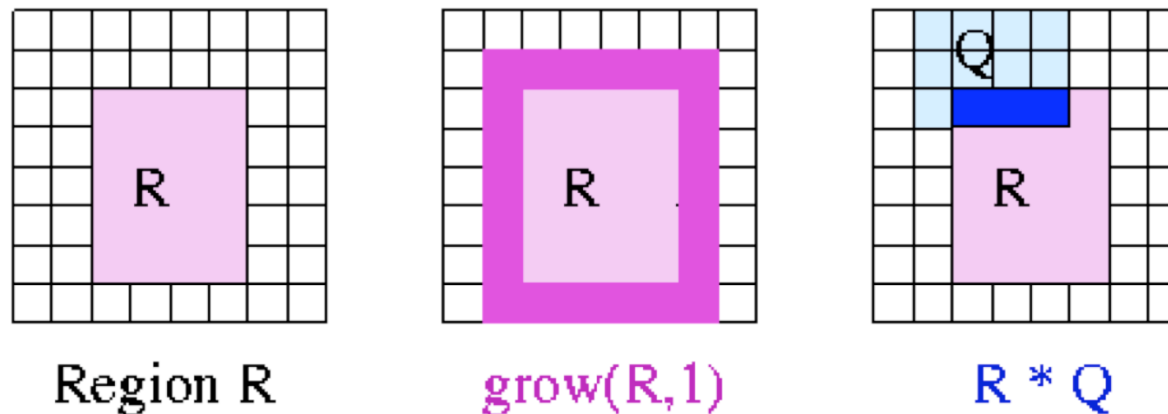
Parallelization

- We update patches in parallel
- Patches communicate periodically, and the data transfers are long: $n \gg n_{1/2}$
- Ghost cells, irregular communication
- Geometric abstractions simplify the bookkeeping



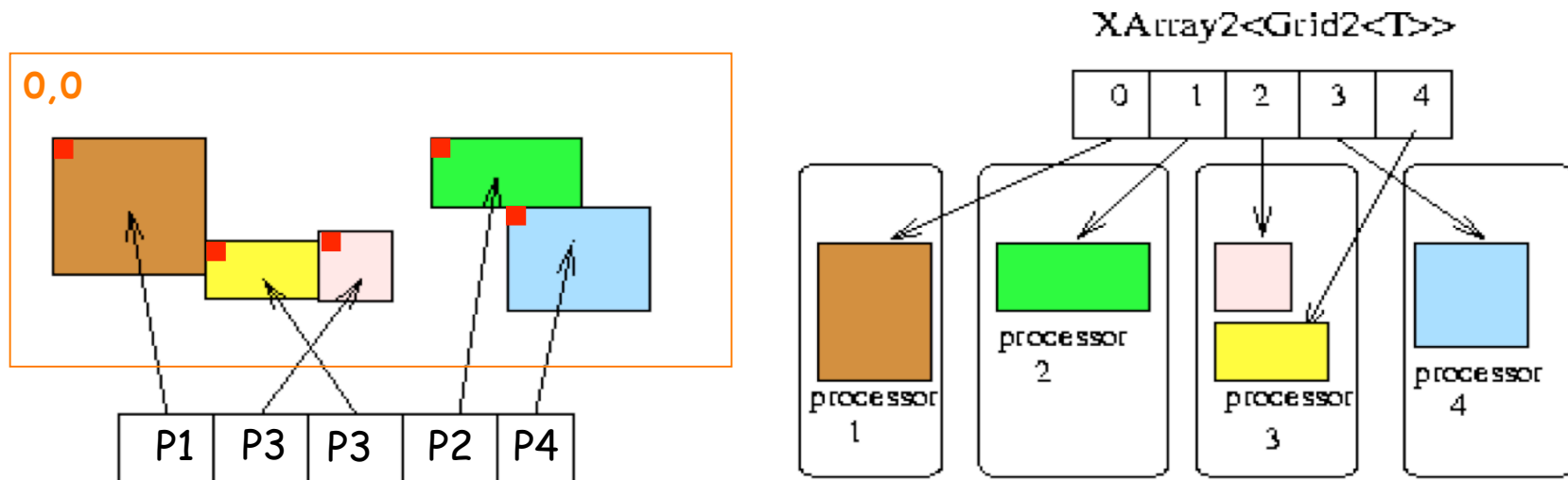
Region Abstraction

- 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)



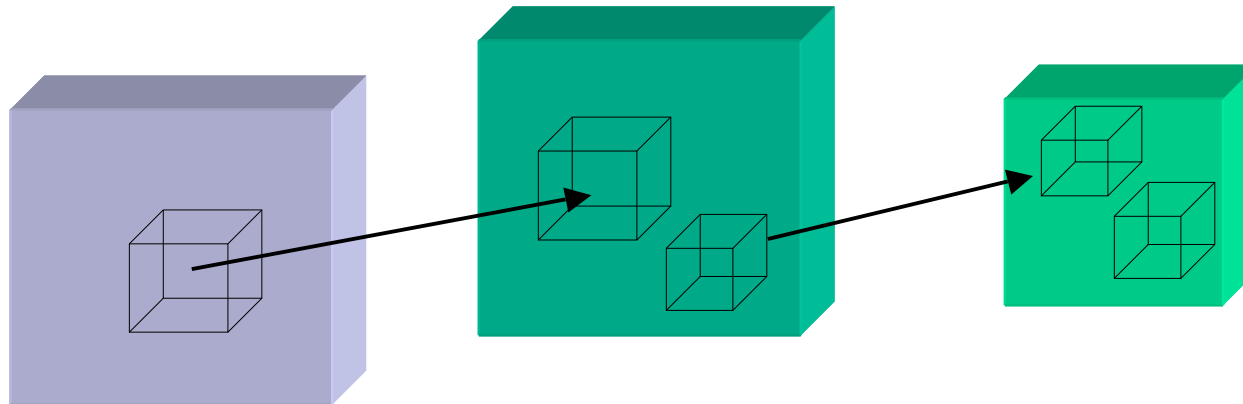
Aggregate abstractions

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



Data motion model

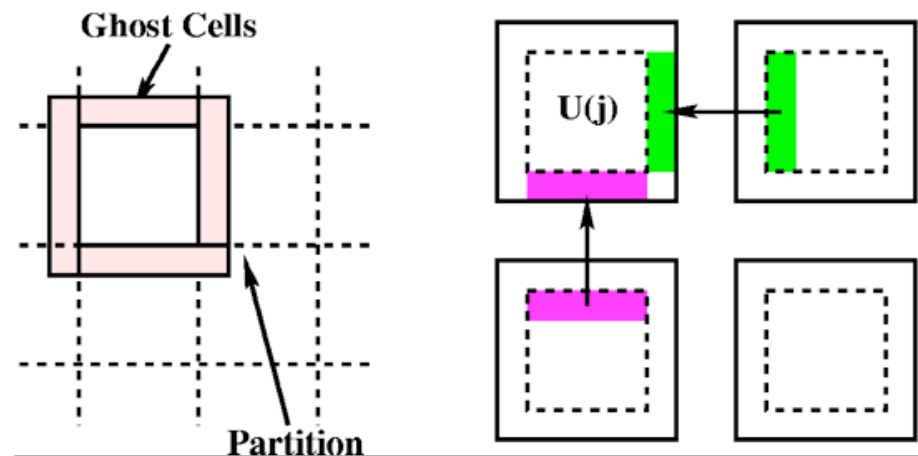
- Unit of transfer is the **MotionItem**, a regular section copy
- Expresses a data dependence between two FloorPlan elements
- A **MotionPlan** describes a global pattern involving 1 or 2 FloorPlans
- The dependence structure is abstract
 - Doesn't say how the data are represented
 - Doesn't say how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



MotionPlan construction: a halo update

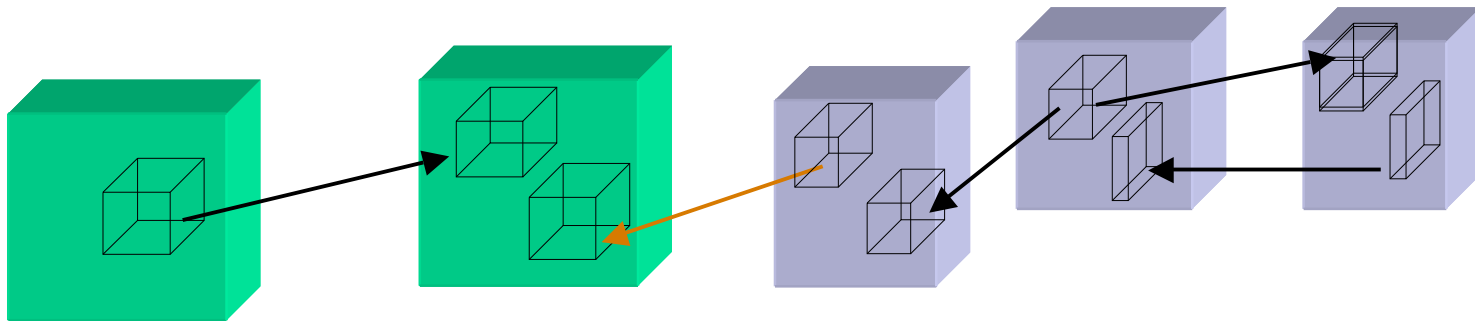
- Dimension independent
- Topologically independent

```
FloorPlan2 U;  
int NGHOST;  
MotionPlan M;  
for each  $U(i) \in U$   
   $I = \text{trim}(U(i), \text{NGHOST})$ ;  
  for each  $U(j) \in U, j \neq i$   
    Region2  $R = I \cap U(j)$   
    M.Copy( $U(i) \cap R, U(j) \cap R$ )  
  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

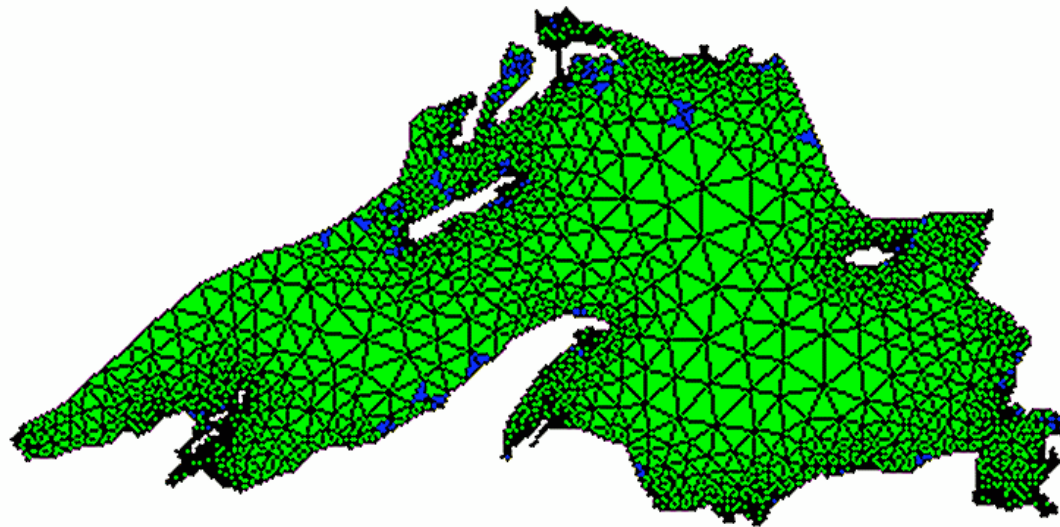


Phases of an SPMD computation

- 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

Unstructured meshes

- “Unstructured” meshes are another type of non-uniformly spaced mesh
- Useful when the boundary or surface of the object is complicated



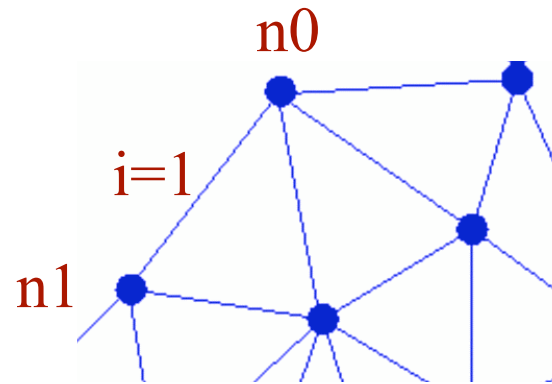
Randy Bank, UCSD

A typical irregular mesh sweep loop

```
float x[n_node], y[n_node]
int E0[n_edge], E1[n_edge]

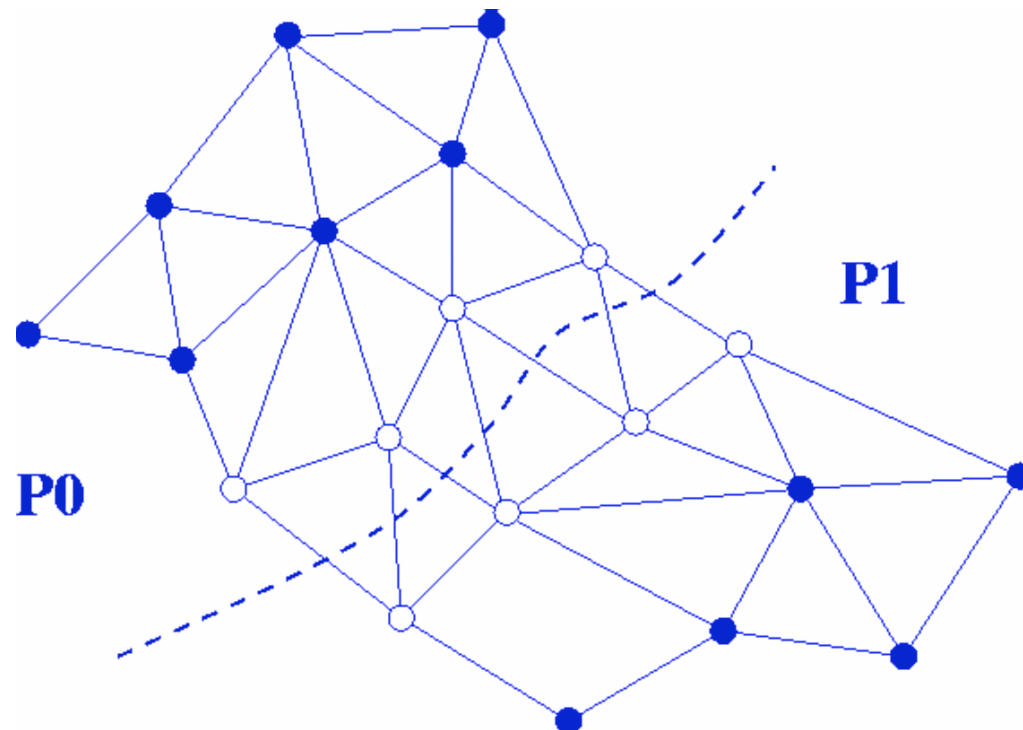
for i = 1 : n_edge
    int n0 = E0[i]
    int n1 = E1[i]
    y[n0] += f(x[n0], x[n1]);
    y[n1] += g(x[n0], x[n1]);
end for
```

// Loop over all edges



Run time support for unstructured meshes

- Data dependences across processor boundaries
- Irregular communication patterns



Implementation issues

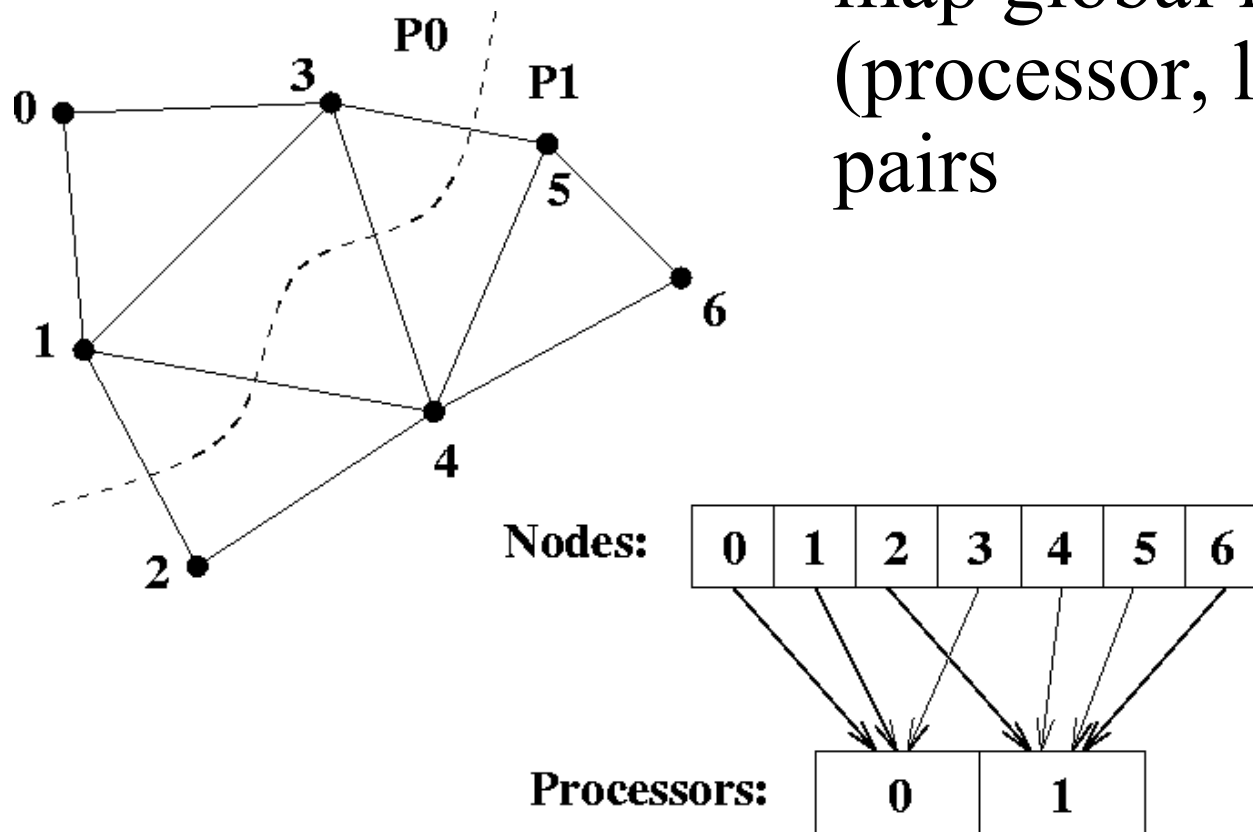
- Because the mesh does not have a regular structure, we must keep processor mapping information for each *point* in the mesh
- Compare with a uniform mesh, in which the mapping information is much coarser grained: mapping is handled at the level of a subdomain comprising many points

Requirements

- We don't know the assignment of nodes to processors until run time
- Assume that the mesh doesn't change once initialized
- We need to include ghost cells, and want to keep the same loop structure
- We execute loops using local indices

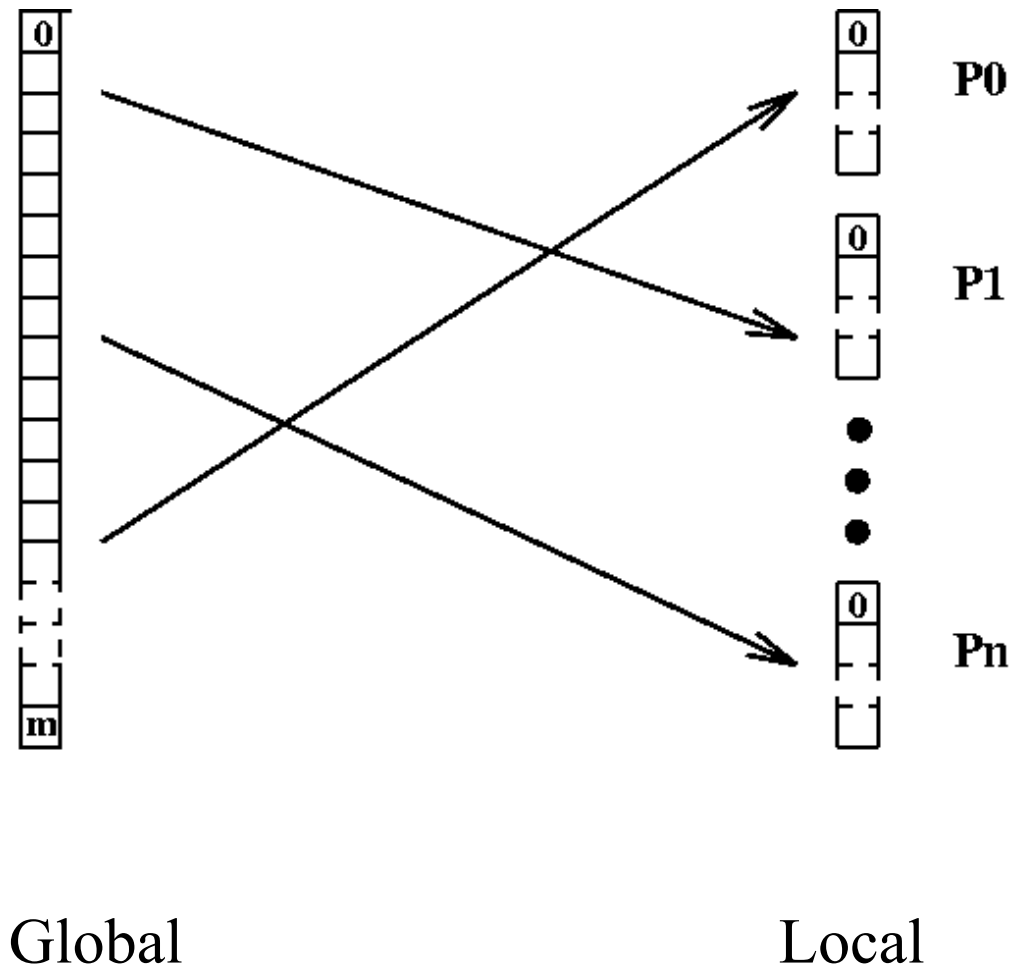
Processor Mapping

- Invoke a partitioner to map global indices to (processor, local index) pairs



Global translation

- Each global index is mapped to a processor and a local index
- A **global translation table** expresses this mapping



Global translation

- Because we require mapping information for each individual mesh node, we cannot fit it in the memory of a single processor
- We must distribute it
- But, where is the mapping information located for a given global index?

Global translation adds a level of indirection

- Let's partition the table into blocks
- For global index g , we look at the translation table on processor $g \text{ div } P$ at local index $g - P (g \text{ div } P)$
- The translation table entry tells us which processor owns the value
- Adds a level of indirection
- Because mapping is fine grained and irregular, lookups are costly if we perform them one at a time

Inspector Executor Model

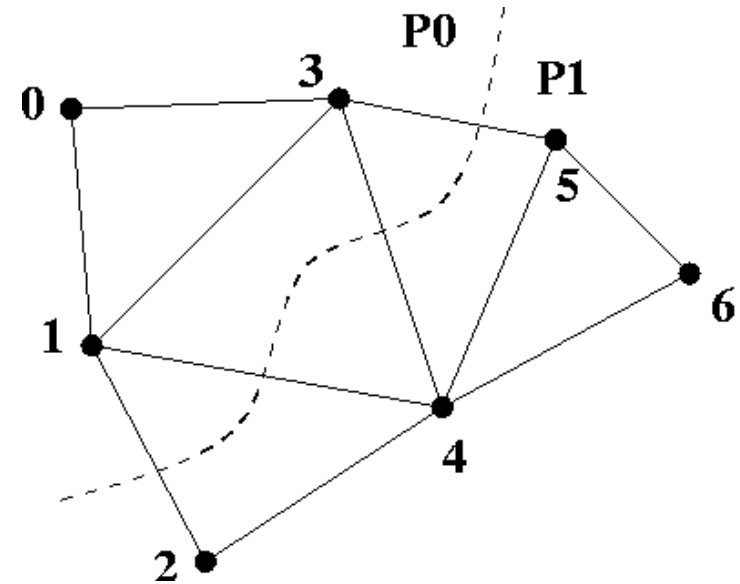
- First articulated by Saltz et al. in the PARTI system
- PARTI divides communication into two separate phases
 - ▶ Inspector: generate communication schedule
 - ▶ Convert global indices to local indices
 - ▶ Aggregate translations to avoid high overhead
 - ▶ Executor: execute communication schedule
 - Gather data into local arrays
 - Compute
 - Scatter back

Schedule generation

- Communication schedules are costly to generate
- We can take advantage of physical locality inherent to the problem
 - Rather than map individual points, we map aggregate collections of points that share a communicating processors
- Reuse the schedule to amortize the cost of generating it

Building a translation table

- Constructs a global translation table
 $g \rightarrow (p, l)$
 - p: processor number
 - l: local address
- Each processor specifies the mapping of a portion of the global index domain



P0

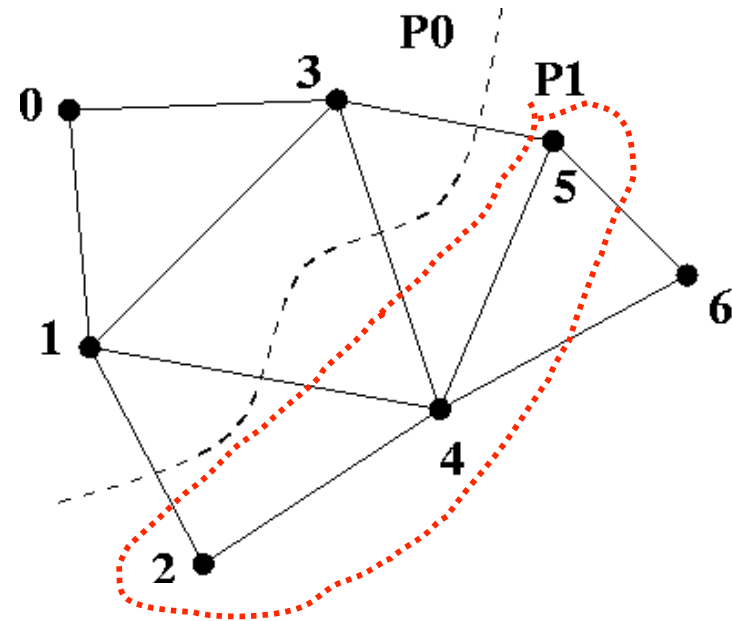
0,0	0,1	1,0	0,2
-----	-----	-----	-----

P1

1,1	1,2	1,3
-----	-----	-----

Adding ghost cells

- P0 (P1) increases #nodes by 3 (2) to accommodate ghost cells
- We generate a communication schedule: how to update ghost cells



Building a communication schedule

- Each processor needs to collect the global to local mappings for all global indices corresponding to ghost cells
- With shared memory, we could simply load the appropriate entries individually
- But with message passing, we need to send and receive a message to map each global index

Aggregate mapping

- Mapping each node individually is expensive
- But with large graphs, each processor will map many different global indices requested a given processor
- Sort each node's global index requests by $g \bmod P$, the owner of the mapping table entry for g
- Collect each sorted segment with the same owner into a single message, and send a bulk request
- Receive the (p,l) pair for each value of g requested

Generating the communication schedule

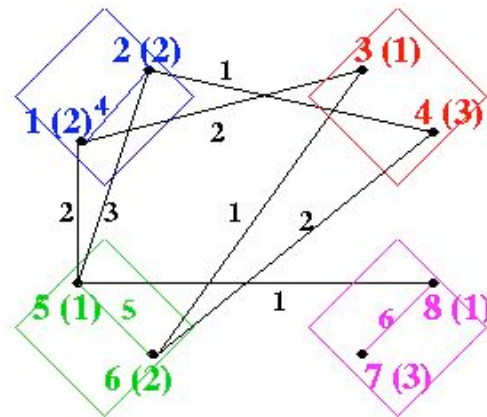
- Sort the (p,l) pairs on p
- Generate a list of local addresses (the l), one for each communicating processor
- The collection of these lists is called the **communication schedule**
- Processors exchange the communication schedules
- Each processor now has a send list

Partitioning the mesh

- Consider techniques that don't use nodal coordinates
 - Depends on edges
 - No assumptions about where “nearest neighbors” are
- A well known technique is called *spectral bisection*
- Kernighan/Lin (1970) is another important algorithm
 - Take a initial partition and iteratively improve it
 - Greedy strategy: splits graph into 2 parts and swaps nodes between sets until cost can no longer be reduced
 - $O(|N|^3)$ running time

Definition of Graph Partitioning

- Given a graph $G = (N, E, W_N, W_E)$
 - N = nodes (or vertices), E = edges
 - W_N = node weights, W_E = edge weights
- Choose a partition $N = N_1 \cup N_2 \cup \dots \cup N_p$ that balances the work load while minimizing communication
 - The sum of the node weights in each N_j is about equal
 - The sum of the edge weights of the cut edges is minimized
- Special case of $N = N_1 \cup N_2$: Graph Bisection

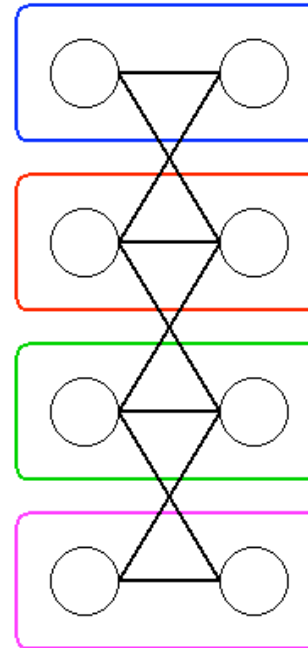


Courtesy Jim Demmel

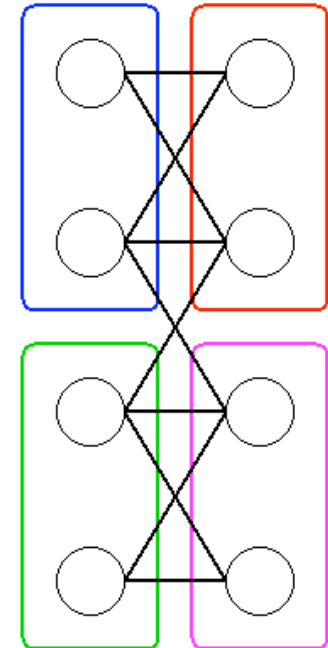
Cost of Graph Partitioning

- Optimal partitioning is NP-complete
- $\binom{n}{n/2}$ possible partitionings:
 $\sqrt{(2n/\pi)} \times 2^n$ possible bisections
- Need good heuristics

Sample Graph Partitionings



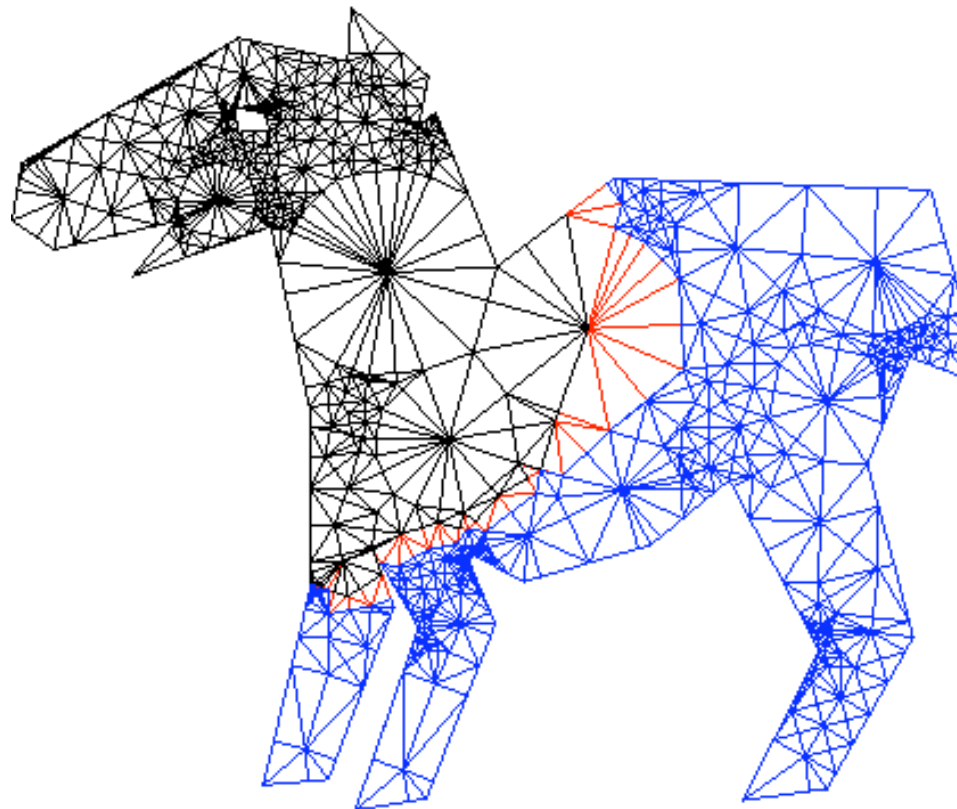
Edge Crossings = 6



Edge Crossings = 10

Recursive Graph Bisection

- To partition N into 2^k parts, bisect graph recursively k times



50 cut edges