

CSE 260

Lecture 17

Irregular Problems

Announcements

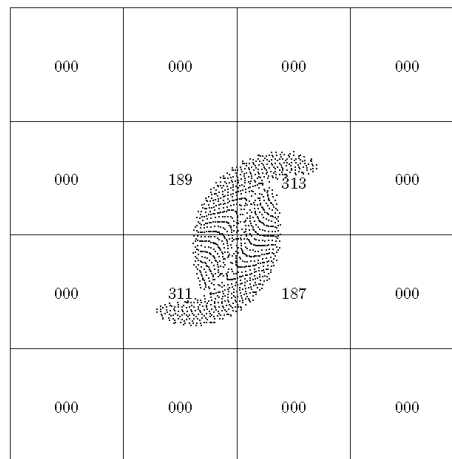
- We will be using Carver rather than Dirac
- Job submission files available on Carver
- Also try out the MPI profiler, enabled via the Makefile
 - ◆ Make command line flag: `fpmpi=1`

Today's lecture

- Non-uniform workload assignment and decomposition
 - ◆ Data Partitioning
 - ◆ Load balancing
- Motivating applications
 - ◆ Particle methods
 - ◆ Mandelbrot set

Non-uniform workloads

- In the assignment, particles are distributed randomly and evenly .. the work applied to each particle is uniform
- In some applications, the particles are distributed **unevenly** ... a uniform (BLOCK) partitioning incurs a workload imbalance



$T = 12.5$ [Eff = 0.200]

Load balancing efficiency

- We can characterize the load imbalance using an **efficiency** metric
- Ignore serial sections and other overheads
 - ♦ Let each processor i complete its assigned work in time T_i
 - ♦ Thus, the running time $T_P = \text{MAX} (T_i)$

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

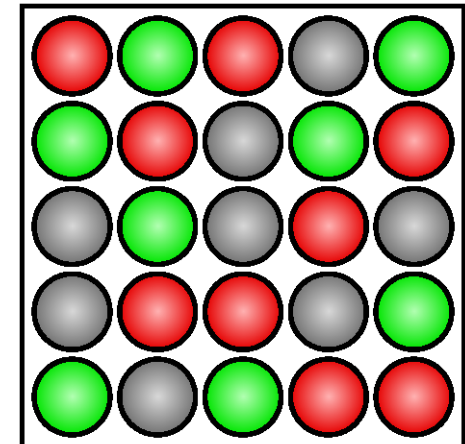
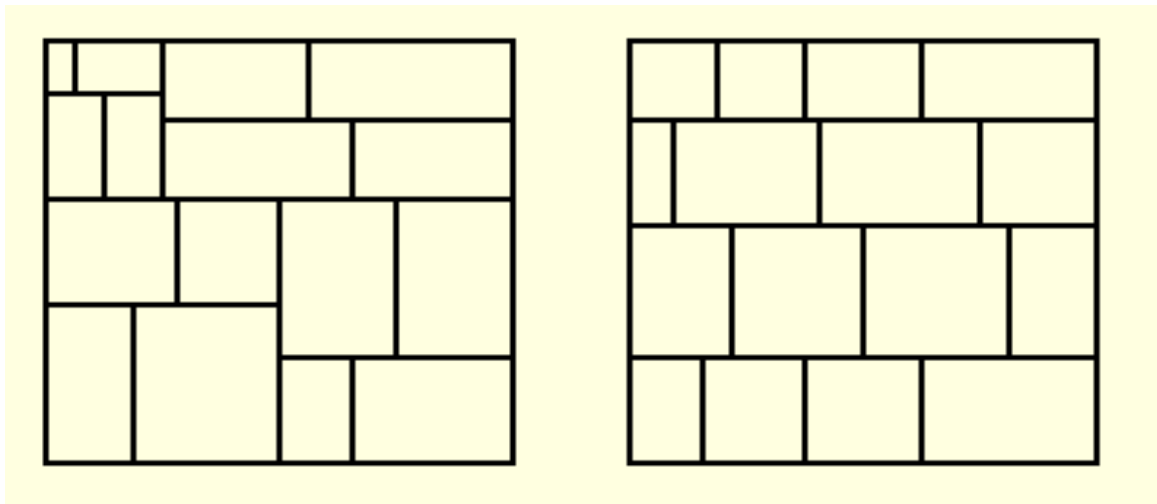
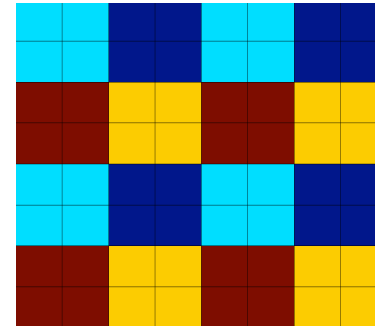
- We define the **load balancing efficiency**

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

- Ideally $\eta = 1.0$

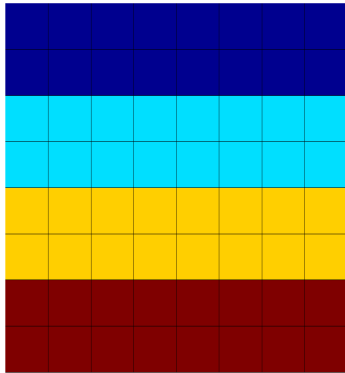
Load balancing strategies

- Static vs dynamic
- Partitioning vs mapping
- Uniform vs non-uniform
- Many to 1
- Shared vs distributed memory

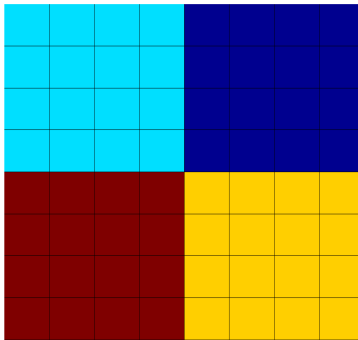


Workload Partitioning

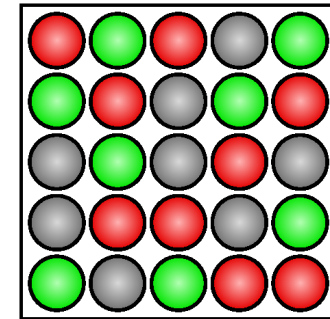
- Block vs. Cyclic
- Static vs. Dynamic



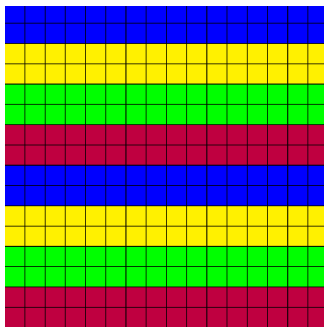
[Block, *]



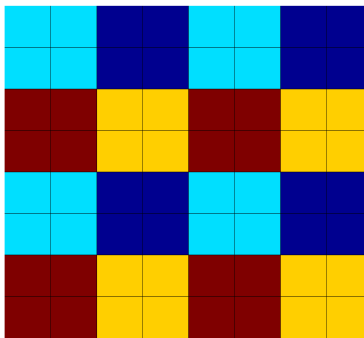
[Block, Block]



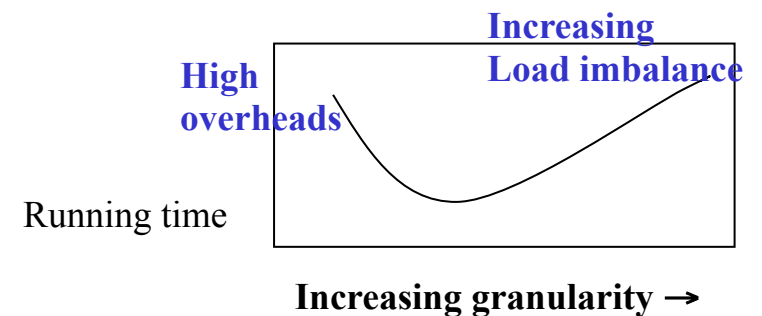
Dynamic



[Cyclic*2), *]



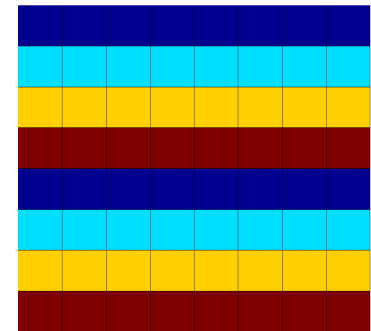
[Cyclic(2), Cyclic(2)]



Static approach: BLOCK CYCLIC

- Divide bins pieces of size CHUNK
- Core k gets chunks $k, k+NT*CHUNK, k+2*NT*CHUNK, \dots$
- All processors should get about the same amount of work if we choose a reasonable chunk size
- Supported by OpenMP

```
#pragma omp parallel for schedule(static, 2)
for( int i = 0; i < n; i++ ) {
    for (int j = 0; j < n; j++ ){
        doWork(i,j);
    }
}
```



Communication overhead

- With a chunk=2, each processor must obtain all the simulation data
- With chunk=4, this factor decreases to $\frac{1}{2}$
- If the dependence distance increases, so must the granularity

Jim Demmel, U. C. Berkeley

Tradeoffs in choosing the chunk size

- CHUNK=1: each box needs data from all neighbors
 - ◆ Every processor loads all neighbor data into its cache!
 - ◆ Compare with [BLOCK,BLOCK]
- CHUNK=2: each box in a chunk of 4 boxes needs $\frac{1}{4}$ of the data from 3 neighboring chunks
 - ◆ Each processor loads 3 chunks of neighbor data into cache
- CHUNK=4: Only edge boxes in a chunk need neighbor data, 20 boxes: processor loads 1.25 chunks of neighbor data



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

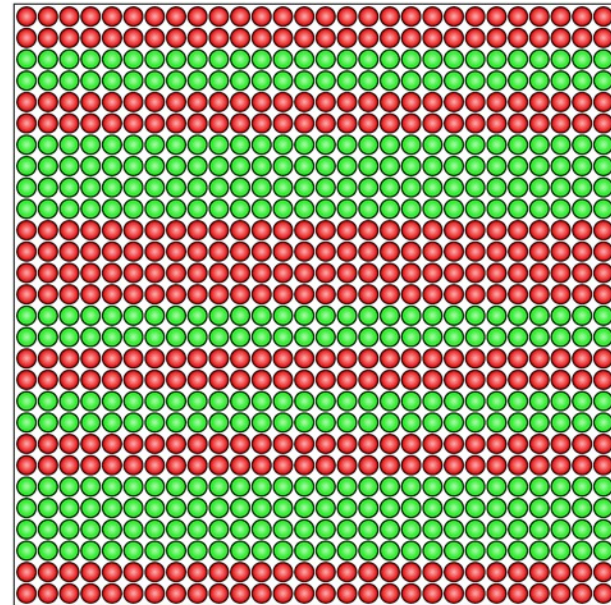
0	0	1	1	0	0	1	1
0	0	1	1	0	0	1	1
2	2	3	3	2	2	3	3
2	2	3	3	2	2	3	3
0	0	1	1	0	0	1	1
0	0	1	1	0	0	1	1
2	2	3	3	2	2	3	3
2	2	3	3	2	2	3	3

	0	1	2	3	0	1	2	3	
	4	5	6	7	4	5	6	7	
	8	9	10	11	8	9	10	11	
	12	13	14	15	12	13	14	15	
	0	1	2	3	0	1	2	3	
	4	5	6	7	4	5	6	7	
	8	9	10	11	8	9	10	11	
	12	13	14	15	12	13	14	15	

Dynamic scheduling

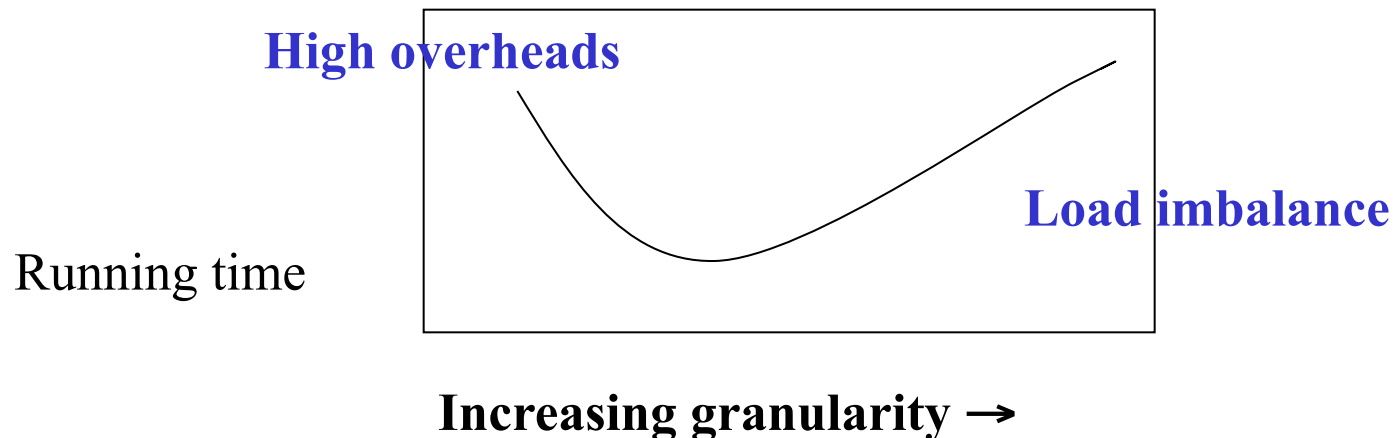
- Unlike CYCLIC decomposition, workload assignments are made on demand
- We subdivide workload into pieces of size **CHUNK**
- Also called *processor self scheduling*
- Shared memory only

```
#pragma omp parallel for \  
    schedule(dynamic, 2)  
for( int i = 0; i < n; i++ ) {  
    for (int j = 0; j < n; j++ ){  
        doWork(i,j);  
    }  
}
```



Dynamic Scheduling on Shared Memory

- Dynamic workload assignments for irregular problems, or when processor performance is not predictable
- **Processor self-scheduling:** processors schedule themselves
- Each thread samples a unique set of indices, changes from run to run
- User tunes work granularity (“chunk size”), trading off sampling overhead against increased load imbalance
 - ♦ Finest granularity: each point is a separate task
 - ♦ Coarsest granularity: one block per core

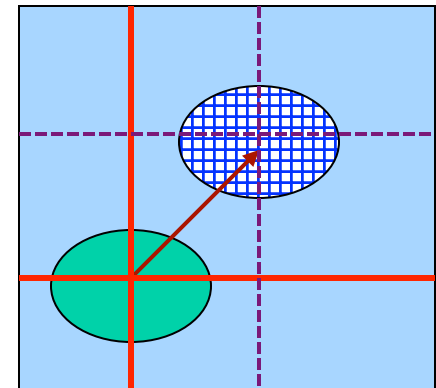
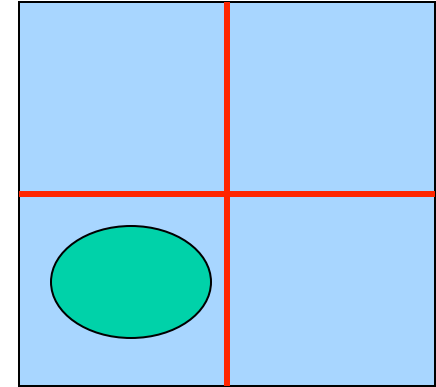


How does self scheduling work?

```
SelfScheduler S(n, P, chunk);  
  
while ( S.getChunk(mymin))  
    for i = mymin to mymin + chunk - 1  
        work(i, chunk);  
    end for  
end while
```

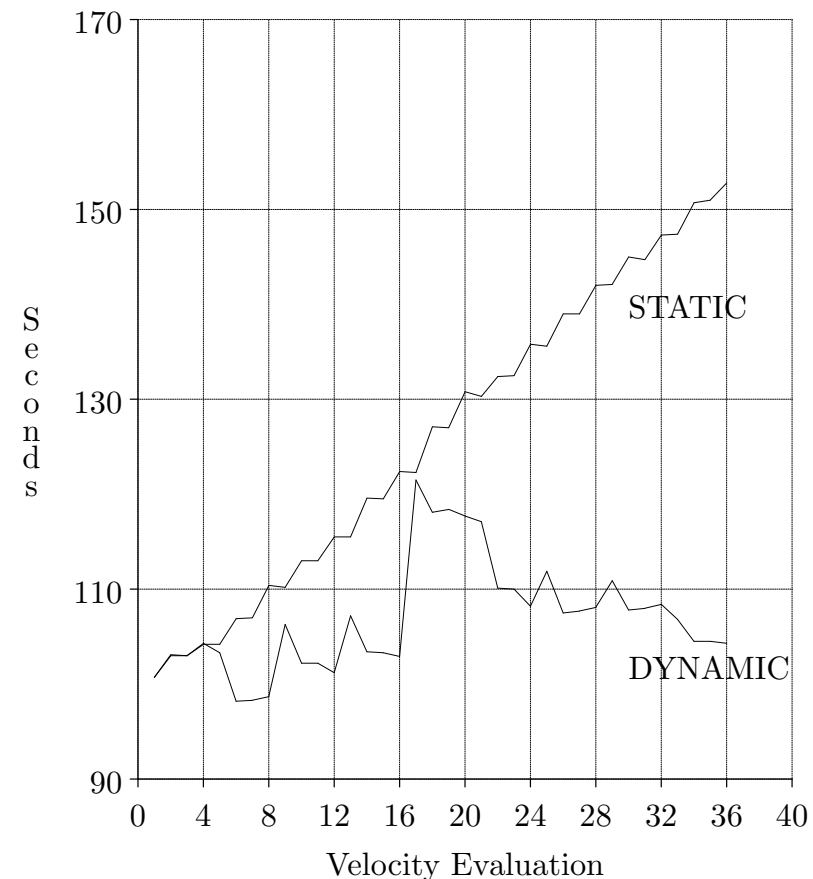
Non-uniform partitioning

- Cyclic partitioning is a static strategy
 - ♦ Partitions space uniformly and maps work irregularly
 - ♦ A statistical sampling method – does not attempt to measure load imbalance
 - ♦ May incur high communication overheads due to fine grain data decomposition
- Another approach is partition the work non-uniformly according to the space-time workload distribution
 - ♦ Each non-uniform region carries an equal work
 - ♦ Particles move: partitioning changes over time
- Avoids high surface to volume ratio of cyclic decomposition



Load imbalance drift increases running time

In many numerical problems, the solution changes gradually due to timestep constraints, so we don't have to repartition every timestep



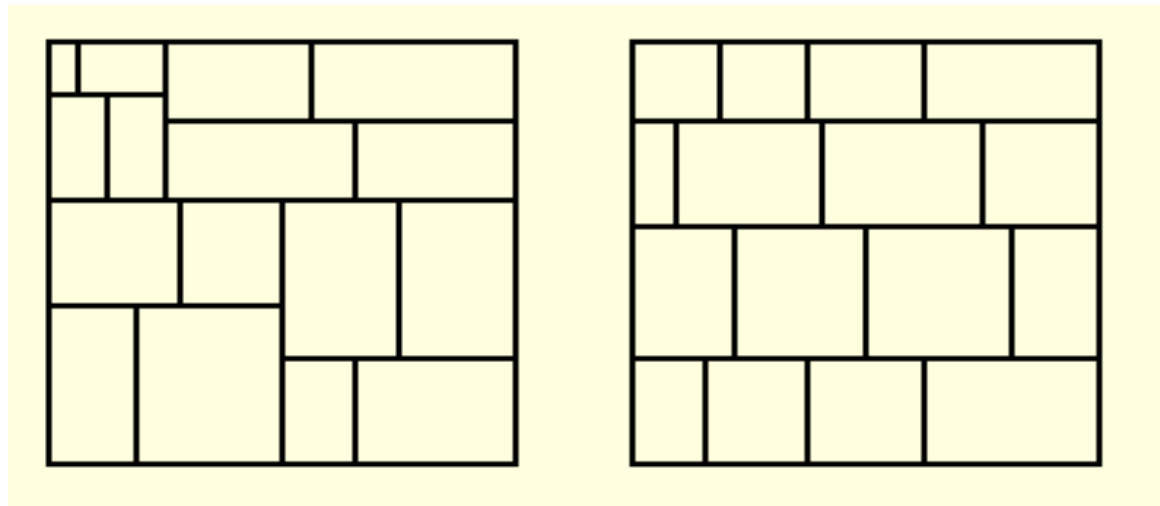
A comparison of static and dynamic load balancing of a vortex dynamics computation on 32 processors of the Intel iPSC-1. If the workloads are partitioned only at the beginning of the calculation (static load balancing, top curve), the loads drift gradually out of balance, and the time required to perform a velocity evaluation steadily increases with time. In contrast, a dynamic load balancing strategy (bottom curve) periodically rebalances the workloads and can maintain an almost steady running time; here loads were re-balanced every fourth velocity evaluation.

The workload density distribution

- Irregular partitioning algorithms require that we estimate the workload in space and time
- Construct a workload density mapping $\rho(\mathbf{x},t)$ giving the workload at each point $\mathbf{x}=(x,y,z)$ of the mesh
- 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
- Since timesteps are “small,” particle motion is localized in space and in time
- This condition holds for continuum methods such as classical partial differential equations: we can adjust the partitioning frequency accordingly

Non-uniform blocked decomposition

- A well known partitioning technique is **recursive coordinate bisection (RCB)**



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

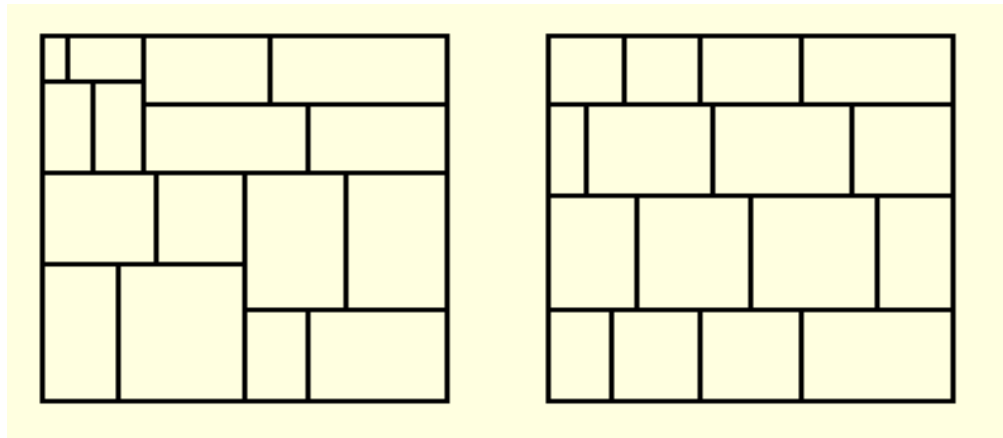
- Let $W[i] = \text{if } (G(i)) \text{ then } 2 \text{ else } 1$
1 1 2 1 2 2 2 1 1 1 1 1 1 2 2 1
- Compute the running sum (i.e. the **scan**) of W
1 2 4 5 7 9 11 12 13 14 15 16 17 19 21 22
- Split into 2 equal parts, at $22/2 = 11$
1 1 2 1 2 2 2 1 1 1 1 1 1 2 2 1

Recursive coordinate bisection

- Recurse until the desired number of partitions have been rendered

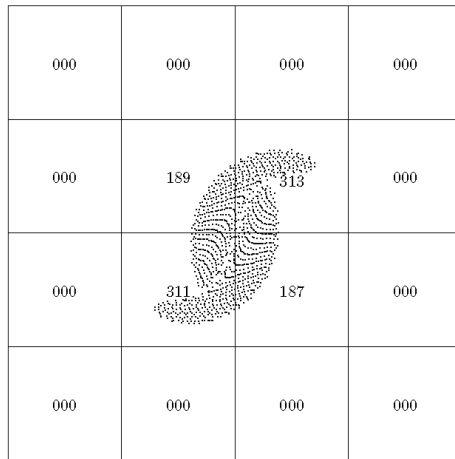
1 1 2 1 2 2 2 1 1 1 1 1 1 2 2 1

- May be applied in multiple dimensions

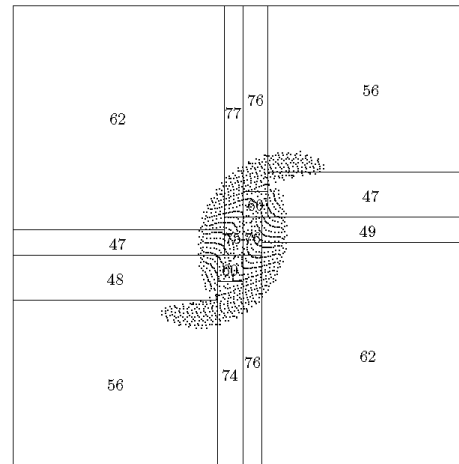


Implications for Communication

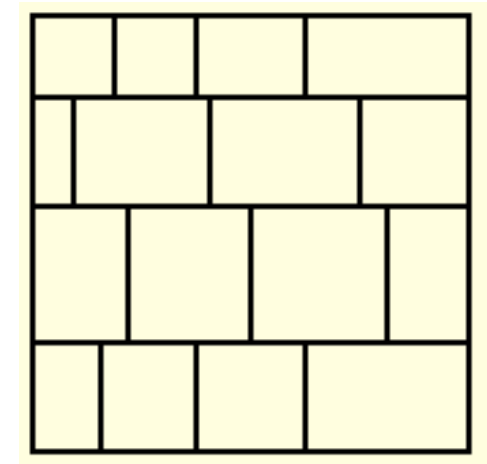
- Partitions can get thin have irregular neighbor structure
- Rectilinear partitioning maintains fixed connectivity, simplifying communication



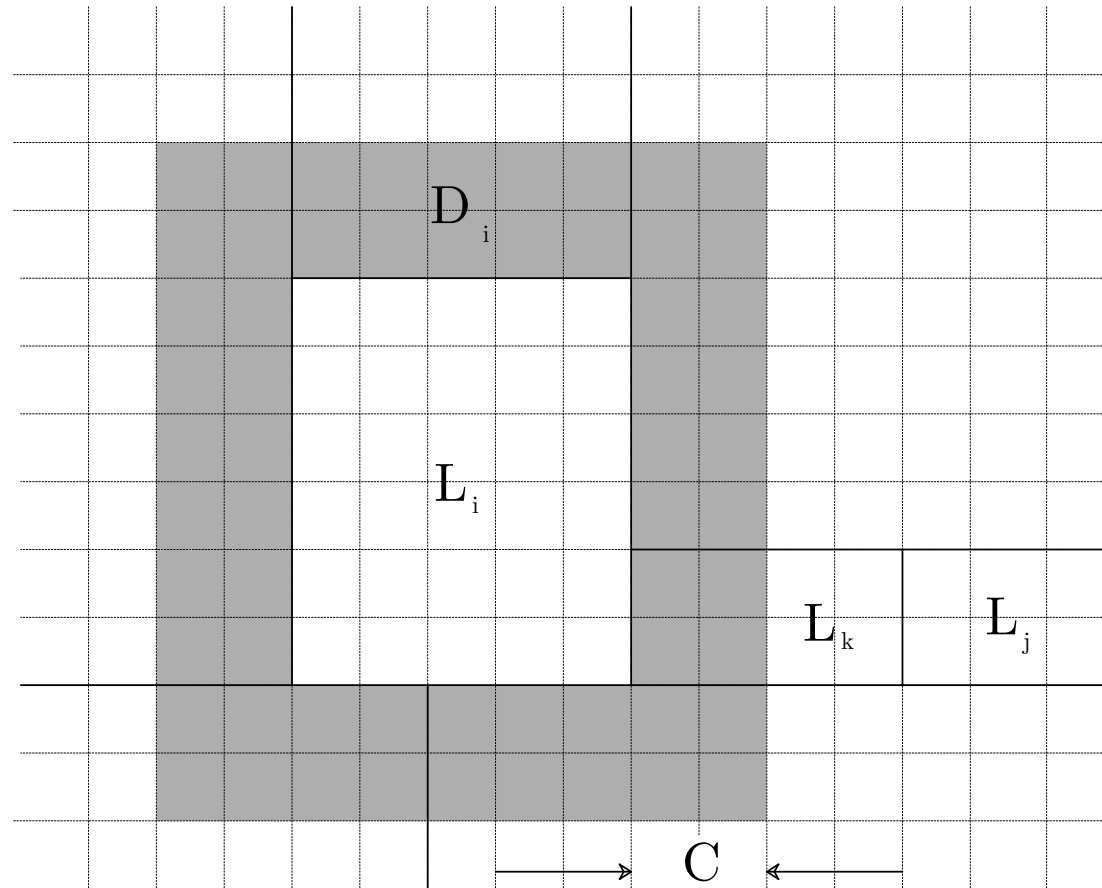
$T = 12.5$ $[Eff = 0.200]$



$T = 12.5$ $[Eff = 0.809]$

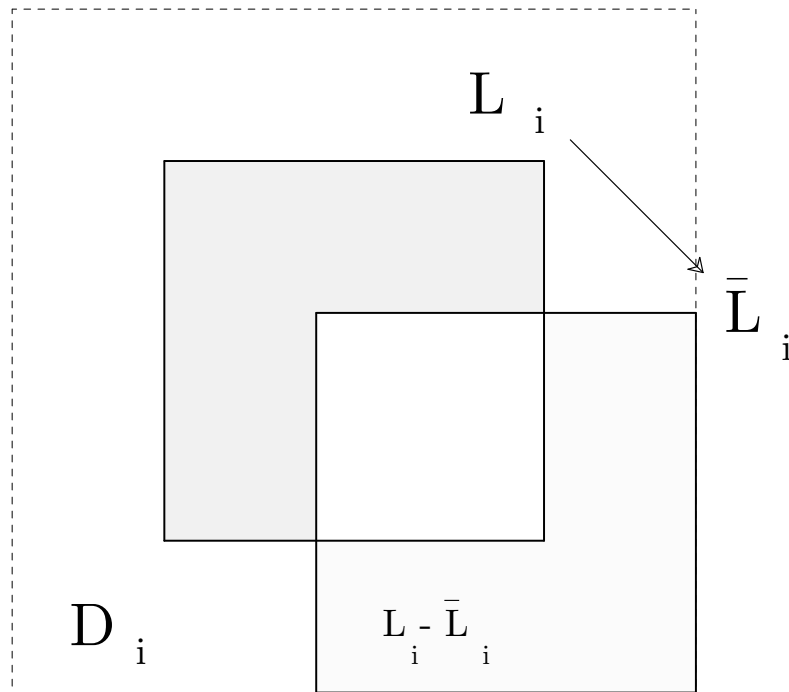


Ghost cells with irregular partitioning



Processor i is assigned L_i , a subregion of the work mesh, and an external interaction region D_i , which we have shaded. D_i is a surrounding shell of bins and is C bins thick, where we have chosen $C = 2$. This region divides into 6 subregions, one for each interacting processor. Subproblems L_i and L_k are locally dependent because D_i and L_k intersect. In comparison, subproblems L_i and L_j are locally independent, because D_i and L_j do not intersect.

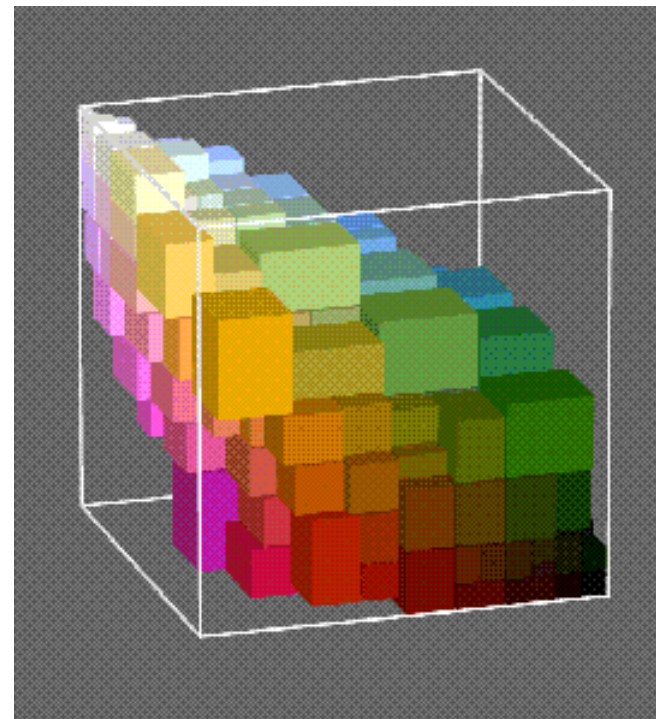
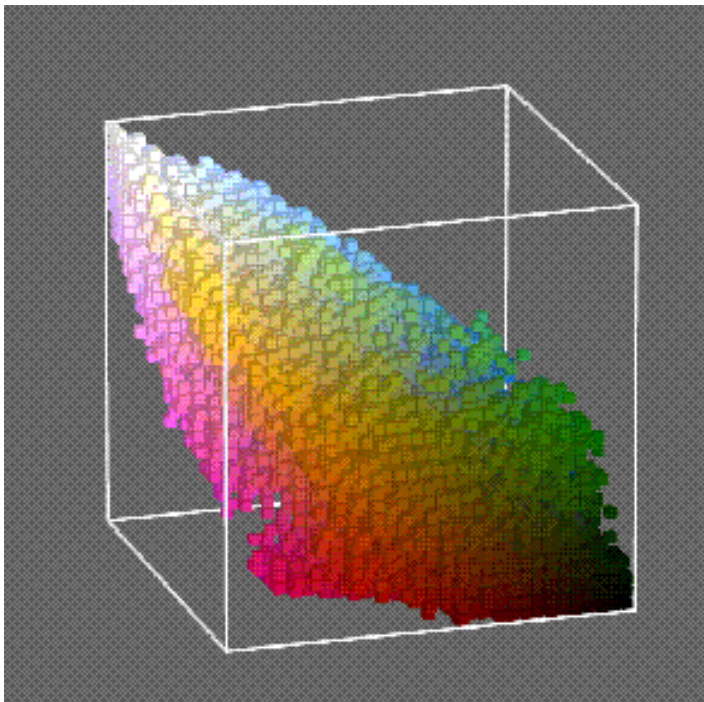
Repatriating particles



Processor i 's assigned subproblem changes from L_i to $\bar{L}_i$ as the result of repartitioning. The processor must give up the work contained in the region $L_i - \bar{L}_i$ to other processors.

Another application of RCB

- Adaptive quantization
- We are given an image which represents each pixel with a fine quantization of intensities, e.g. 24 bits
- Come up with a coarser quantization, 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 colormap
 - ♦ 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.icas.edu/Dienst/UI/2.0/Describe/ncstrl.icas/TR-97-29>



15 bits—uniform



8 bits—adaptive



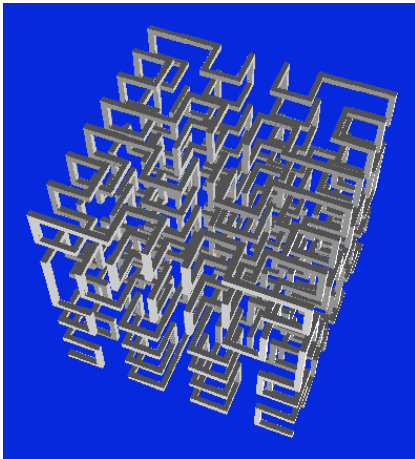
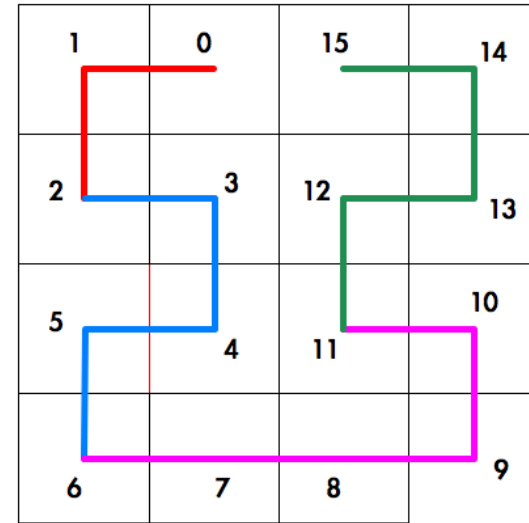
8 bits—uniform

Load Balancing with space filling curves

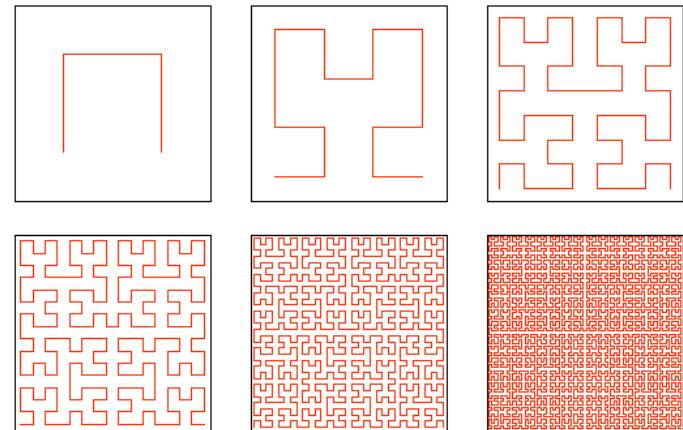
- Multidimensional RCB suffers from granularity problems
- We can reduce the granularity with many partitions per processor
- Introduces higher surface to volume effects
- Another approach: spacefilling curves

Partitioning with filling curves

- Maps higher dimensional physical space onto the line
 - ♦ Load balancing in one dimension
 - ♦ Many types;
Hilbert curves shown here
- Irregular communication surfaces
- Useful for managing locality:
memory, databases



mathforum.org/advanced/robertd/lsys3d.html



<http://www.math.ohio-state.edu/~fedorow/math655/Peano.html>

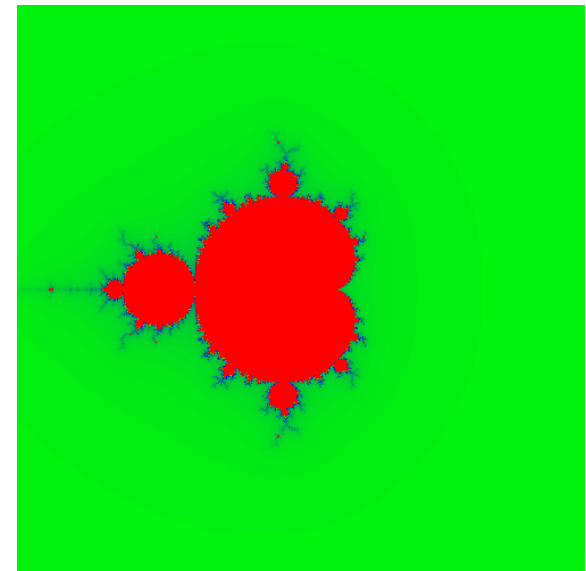
Load balancing the Mandelbrot set computation

- Application: for which points c in the complex plane does the following iteration remain bounded?

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

c is a complex number, $z_0 = 0$

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



A quick review of complex numbers

- 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: $|z| = \sqrt{x^2 + y^2}$
- $z^2 = (x+\mathbf{i}y)(x+\mathbf{i}y) = (x^2 - y^2) + 2xy\mathbf{i}$

A load balancing problem

- Some points iterate **longer** than others
- A uniform decomposition, some processors finish later than others
- We have a *load imbalance*

#pragma omp parallel for

for i = 0 to n-1

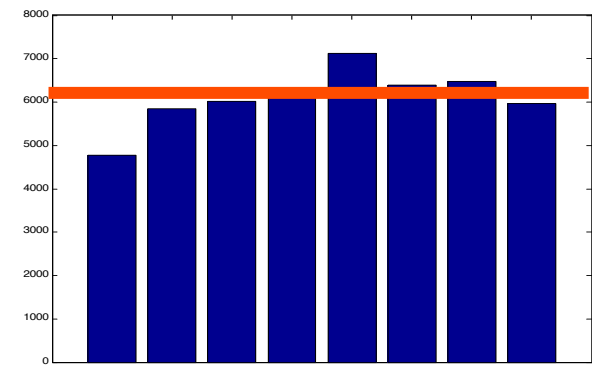
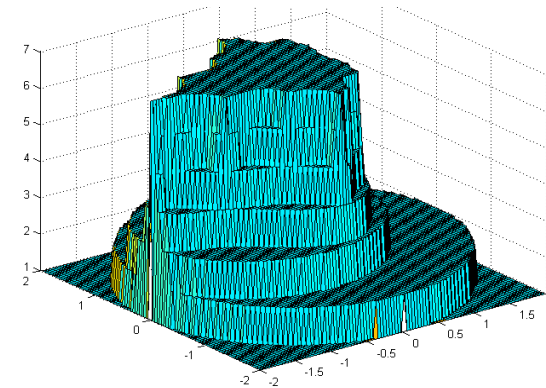
for j = 0 to n-1

$z_0 = \text{Complex}(x[i], y[i])$

while ($|z_k| < 2$ or $k > \text{MAXITER}$)

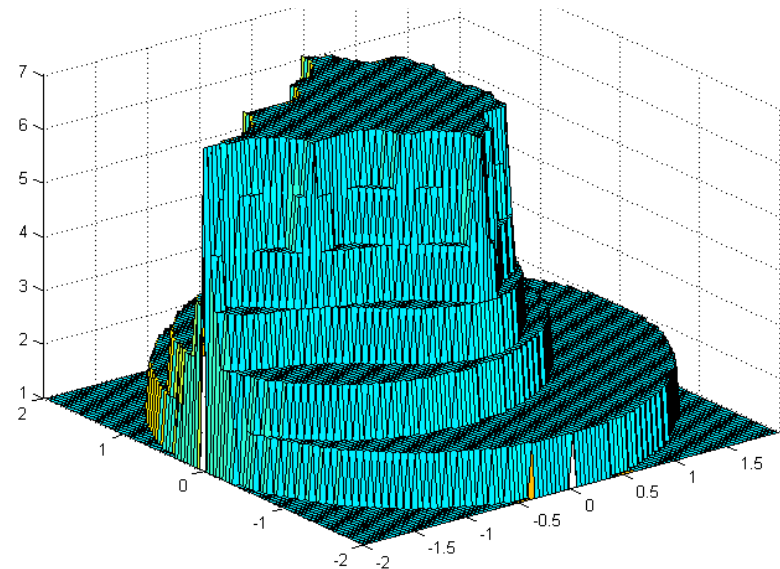
$z_{k++} += z_k^2 + c$

Output[i,j] = k



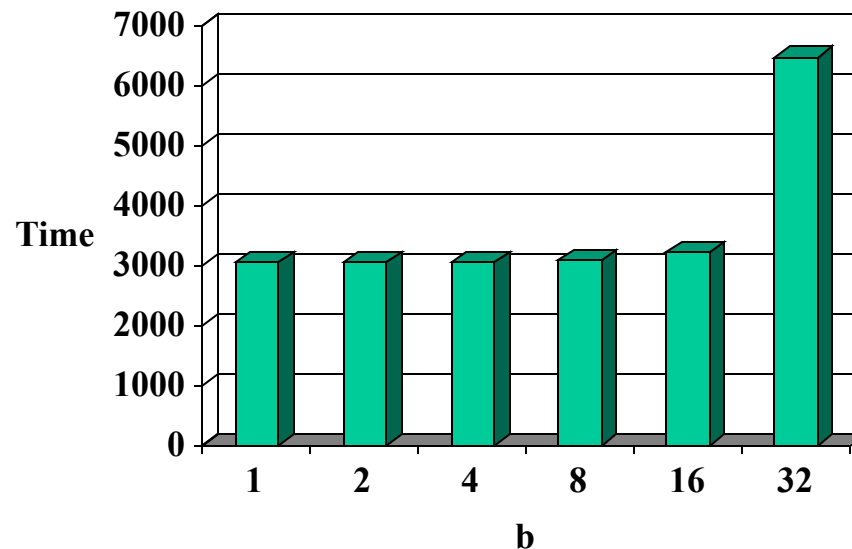
Granularity tradeoff

- 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

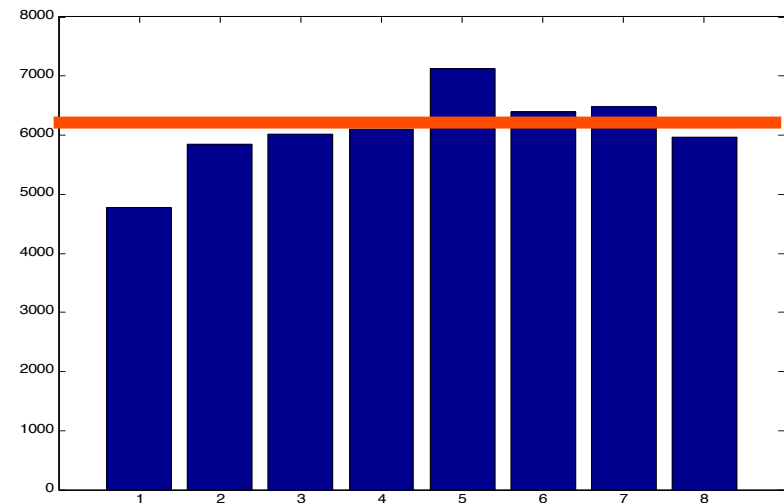


Limits to performance

- 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



Guided self-scheduling

- Observation: we need many more pieces of work than there are processors (say 10 times) at the time we are sampling the counter
- Adjust or “guide” task granularity in proportion to the amount of remaining work
- For example, $g = N_{\text{Remain}}/10P$
- When there's lots of work to do, we hand out large chunks. Toward the end, we hand out smaller chunks
- See Polychronopoulos, C.D., and Kuck, D.J. “Guided self-scheduling: a practical scheduling scheme for parallel supercomputers.” *IEEE Trans, on Computers*, Vol. C-36, No. 12, Dec. 1987, pp. 1425-1439.