Day 7:
High Performance Computing
CMT106

David W. Walker

Professor of High Performance Computing

Cardiff University

http://www.cardiff.ac.uk/people/view/118172-walker-david

Day 7

- 9:30 – 10:30am: Lecture on dynamic and static load balancing techniques: orthogonal recursive bisection, hierarchical recursive bisection, block cyclic data decompositions.
- 10:30 – 10:50am: Break.
- 10:50am – 11:40pm: Lecture on cellular automaton model of catalytic converter.
- 11:40 – 1:10pm: Lunch break.
- 1:10 – 3:00pm: Self study, do the questions on the worksheet.
- 3:00 – 4:30pm: Review of the worksheet questions.
- 4:30 – 5:00pm: Review module, structure of the exam.

Topics Covered on Days 1-4

- *Day 1:* Introduction to parallelism; motivation; types of parallelism; Top500 list; classification of machines; SPMD programs; memory models; shared and distributed memory; OpenMP; example of summing numbers.
- *Day 2*: Interconnection networks; network metrics; classification of parallel algorithms; speedup and efficiency.
- *Day 3*: Scalable algorithms; Amdahl's law; sending and receiving messages; programming with MPI; collective communication; integration example.
- *Day 4:* Regular computations and simple examples – the wave equation and Laplace's equation.

Topics Covered on Days 5-7

- *Day 5*: Programming GPUs with CUDA; CUDA device memory architecture; simple programming examples.
- *Day 6*: Dynamic communication and the molecular dynamics example; irregular computations; the WaTor simulation.
- *Day 7*: *Load balancing strategies; block-cyclic data distribution, surface catalysis model.*

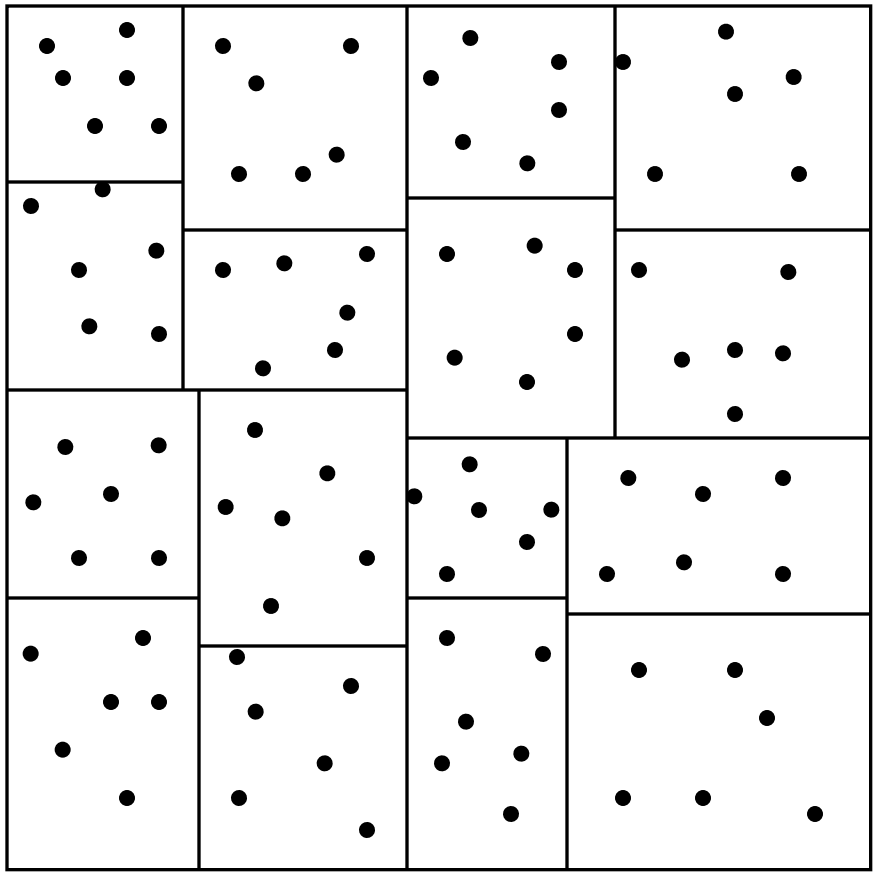
Dynamic Load Imbalance

- In dealing with dynamic load imbalance the following two approaches are important:
- Use of a dynamic load balancer so that the distribution of the ocean among the processes changes as the fish and shark system evolves. When dealing with grids some form of *recursive bisection* is often used.
- Use of a *cyclic*, or *scattered*, data distribution. The parts of the grid assigned to one process do not form a contiguous block but are scattered in a regular way over the whole domain. The aim in this case is to achieve statistical load balance.

Orthogonal Recursive Bisection

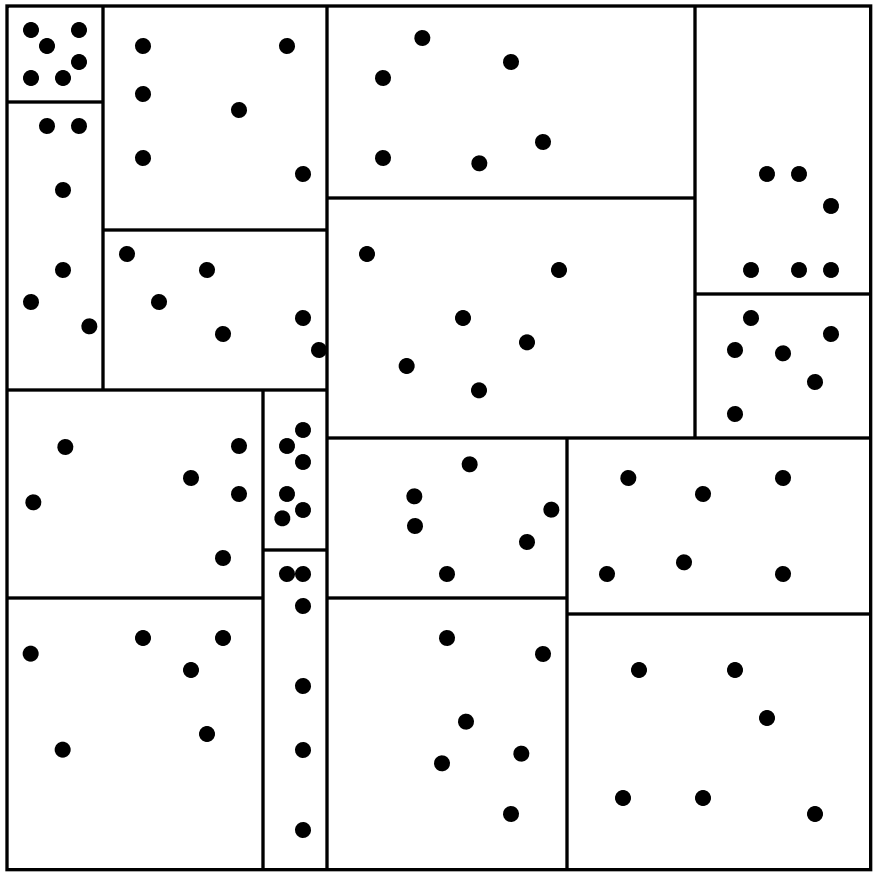
- Orthogonal Recursive Bisection (ORB) first divides the domain orthogonal to the x-direction so there are equal numbers of items in each of the two subdomains.
- Then each of these 2 subdomains is independently divided orthogonal to the y-direction, to give 4 subdomains each with approximately the same number of items in each
- This process of bisection continues, alternating between the x and y directions, until there is one subdomain for each process.

Example of ORB 1



ORB is not used when the items are distributed uniformly over the domain - in this case the subdomains would come out about the same size and shape.

Example of ORB 2



If the items are distributed unevenly over the domain, ORB can give rise to a variety of different shaped process subdomains.

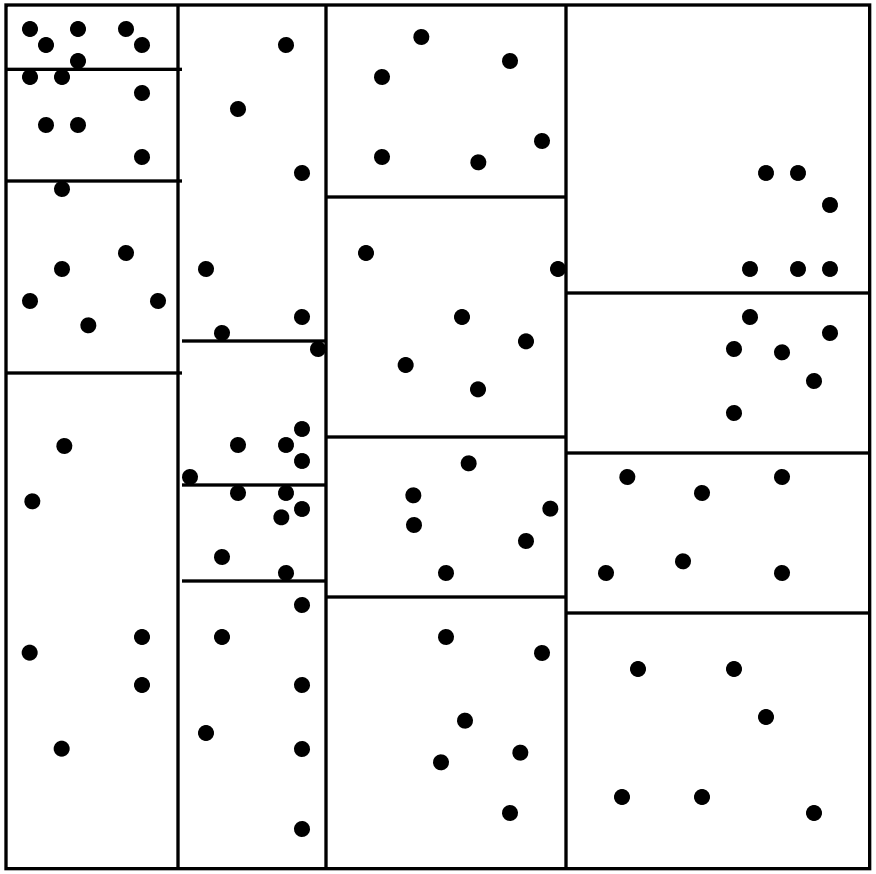
Notes on ORB

Using a dynamic load balance scheme such as ORB adds to the complexity of the software, particularly in deciding which boundary data must be communicated with which processes.

Hierarchical Recursive Bisection

- HRB is a variation of ORB in which we first make all the cuts in one direction, and then all the cuts in the second direction, rather than alternating directions.
- HRB allows the data distribution to be adjusted over just one direction, rather than both.
- ORB and HRB can easily be extended to 3 or more dimensions.

Example of HRB



Cyclic Data Distributions

- In a cyclic data distribution the data assigned to each process is scattered in a regular way over the domain of the problem.
- The figure on the next slide shows how a grid might be cyclically distributed over a 4x4 mesh of processes.
- The cyclic distribution is a simple way to improve load balance but can result in more communication as it increases the amount of boundary data in a process.

(0,0)	(0,1)	(0,2)	(0,3)	(0,0)	(0,1)	(0,2)	(0,3)	(0,0)	(0,1)	(0,2)	(0,3)
(1,0)	(1,1)	(1,2)	(1,3)	(1,0)	(1,1)	(1,2)	(1,3)	(1,0)	(1,1)	(1,2)	(1,3)
(2,0)	(2,1)	(2,2)	(2,3)	(2,0)	(2,1)	(2,2)	(2,3)	(2,0)	(2,1)	(2,2)	(2,3)
(3,0)	(3,1)	(3,2)	(3,3)	(3,0)	(3,1)	(3,2)	(3,3)	(3,0)	(3,1)	(3,2)	(3,3)
(0,0)	(0,1)	(0,2)	(0,3)	(0,0)	(0,1)	(0,2)	(0,3)	(0,0)	(0,1)	(0,2)	(0,3)
(1,0)	(1,1)	(1,2)	(1,3)	(1,0)	(1,1)	(1,2)	(1,3)	(1,0)	(1,1)	(1,2)	(1,3)
(2,0)	(2,1)	(2,2)	(2,3)	(2,0)	(2,1)	(2,2)	(2,3)	(2,0)	(2,1)	(2,2)	(2,3)
(3,0)	(3,1)	(3,2)	(3,3)	(3,0)	(3,1)	(3,2)	(3,3)	(3,0)	(3,1)	(3,2)	(3,3)
(0,0)	(0,1)	(0,2)	(0,3)	(0,0)	(0,1)	(0,2)	(0,3)	(0,0)	(0,1)	(0,2)	(0,3)
(1,0)	(1,1)	(1,2)	(1,3)	(1,0)	(1,1)	(1,2)	(1,3)	(1,0)	(1,1)	(1,2)	(1,3)
(2,0)	(2,1)	(2,2)	(2,3)	(2,0)	(2,1)	(2,2)	(2,3)	(2,0)	(2,1)	(2,2)	(2,3)
(3,0)	(3,1)	(3,2)	(3,3)	(3,0)	(3,1)	(3,2)	(3,3)	(3,0)	(3,1)	(3,2)	(3,3)

Cyclic Data Distributions

Consider a one-dimensional cyclic data distribution of an array, such as:

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

This is known as a cyclic[1] data distribution, and can be regarded as mapping a global index, m, to a process location, p, and a local index, i.

Cyclic[1] Data Mappings

- The global index, m, maps to a process location, p, and a local index, i.

$$m \rightarrow (p,i)$$

where p and i are given by:

$$p = m \pmod N$$

$$i = \text{floor}(m/N)$$

and N is the number of processes. The inverse mapping is:

$$m = iN + p$$

Cyclic[k] Data Mappings

- If we arrange array entries in groups of size k and cyclically distribute these we get a cyclic[k] data distribution.
- For example, the following shows a cyclic[2] data distribution.

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

http://users.cs.cf.ac.uk/David.W.Walker/PHP/CyclicTable.php

Number of processes, N = 5
Block size, k = 3
First global index, m = 0
Final global index, m = 19

m	B	p	b	i	j
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	0	2	2
3	1	1	0	0	0
4	1	1	0	1	1
5	1	1	0	2	2
6	2	2	0	0	0
7	2	2	0	1	1
8	2	2	0	2	2
9	3	3	0	0	0
10	3	3	0	1	1
11	3	3	0	2	2
12	4	4	0	0	0
13	4	4	0	1	1
14	4	4	0	2	2
15	5	0	1	0	3
16	5	0	1	1	4
17	5	0	1	2	5
18	6	1	1	0	3
19	6	1	1	1	4

Cyclic[k] Example

m is the global index
B is the global block index
p is the process number
b is the local block index
i is the local index within the block
j is the local index in the process

$$j = kb + i$$

where k is the block size

Cyclic[k] Data Mappings 2

Global index m is mapped to process location p, local block index b, and local index i within the block, as follows:

$$p = B \pmod N$$

$$b = \text{floor}(B/N)$$

$$i = m \pmod k$$

where B=floor(m/k) is the global block index. The inverse mapping is:

$$m=(bN+p)k + i$$

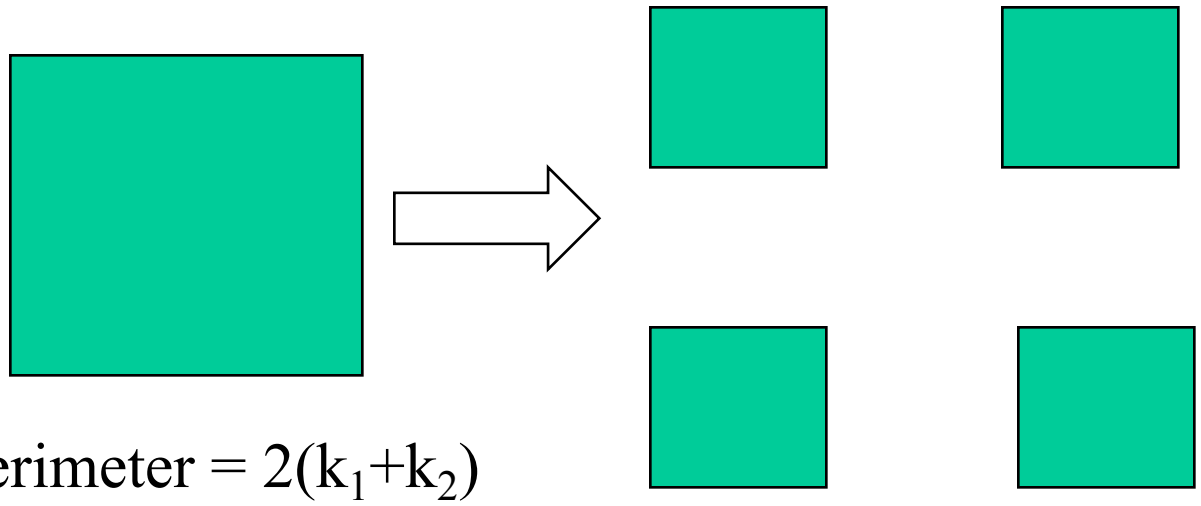
Communication and Load Imbalance Tradeoff

- A block cyclic data distribution can be used to improve load balance when data is distributed inhomogeneously across the problem domain.
- However, a smaller block size results in more boundary data and hence gives rise to increased communication.
- There is, therefore, a tradeoff between load imbalance and communication cost.
- It is important to choose the correct block size so that the total overhead is minimised.

Example

- Assume that the amount of communication associated with a block is proportional to its perimeter.
- Suppose we have a 2-D block cyclic distribution with block size k_1 by k_2.
- Now we reduce the block size by a factor of 2 in each direction, so each block in the original data distribution is split into 4 blocks, each of size $k_1/2$ by $k_2/2$.

Example (continued)



Perimeter = $2(k_1+k_2)$

Perimeter = $2(k_1/2+k_2/2) = k_1+k_2$

- The perimeter of the original block is $2(k_1+k_2)$.
- After it is split into 4 smaller blocks the total perimeter of these blocks is $4(k_1+k_2)$.
- So, for a 2D problem, we expect the communication cost to double when the block size is halved in each direction

Multi-Dimensional Data Distributions

- Multi-dimensional arrays are distributed by applying the desired data distribution separately to each array index.
- Thus, for a two-dimensional data distribution the global index (m,n) is mapped so that m➔ (p,i) and n ➔ (q,j), where (p,q) is location on a PxQ process mesh, and (i,j) is the index into the local 2D array.
- Different data distributions can be applied over each array dimension.

Multi-Dimensional Data Distributions 2

- For a 2D (cyclic[1],cyclic[1]) data distribution we would have:

$$m \rightarrow (p,i) = (m(\text{mod } P), \text{floor}(m/P))$$
$$n \rightarrow (q,j) = (n(\text{mod } Q), \text{floor}(n/Q))$$

(0,0)	(0,1)	(0,2)	(0,3)	(0,0)	(0,1)	(0,2)	(0,3)	(0,0)	(0,1)	(0,2)	(0,3)
(1,0)	(1,1)	(1,2)	(1,3)	(1,0)	(1,1)	(1,2)	(1,3)	(1,0)	(1,1)	(1,2)	(1,3)
(2,0)	(2,1)	(2,2)	(2,3)	(2,0)	(2,1)	(2,2)	(2,3)	(2,0)	(2,1)	(2,2)	(2,3)
(3,0)	(3,1)	(3,2)	(3,3)	(3,0)	(3,1)	(3,2)	(3,3)	(3,0)	(3,1)	(3,2)	(3,3)
(0,0)	(0,1)	(0,2)	(0,3)	(0,0)	(0,1)	(0,2)	(0,3)	(0,0)	(0,1)	(0,2)	(0,3)
(1,0)	(1,1)	(1,2)	(1,3)	(1,0)	(1,1)	(1,2)	(1,3)	(1,0)	(1,1)	(1,2)	(1,3)
(2,0)	(2,1)	(2,2)	(2,3)	(2,0)	(2,1)	(2,2)	(2,3)	(2,0)	(2,1)	(2,2)	(2,3)
(3,0)	(3,1)	(3,2)	(3,3)	(3,0)	(3,1)	(3,2)	(3,3)	(3,0)	(3,1)	(3,2)	(3,3)
(0,0)	(0,1)	(0,2)	(0,3)	(0,0)	(0,1)	(0,2)	(0,3)	(0,0)	(0,1)	(0,2)	(0,3)
(1,0)	(1,1)	(1,2)	(1,3)	(1,0)	(1,1)	(1,2)	(1,3)	(1,0)	(1,1)	(1,2)	(1,3)
(2,0)	(2,1)	(2,2)	(2,3)	(2,0)	(2,1)	(2,2)	(2,3)	(2,0)	(2,1)	(2,2)	(2,3)
(3,0)	(3,1)	(3,2)	(3,3)	(3,0)	(3,1)	(3,2)	(3,3)	(3,0)	(3,1)	(3,2)	(3,3)

(cyclic[k_r],cyclic[k_c]) Example

- Take (k_r,k_c) = (3,4) and (P,Q) = (5,4). Then where is element (m,n) = (18,22)?
- Rows: m = 18, k_r = 3, P = 5

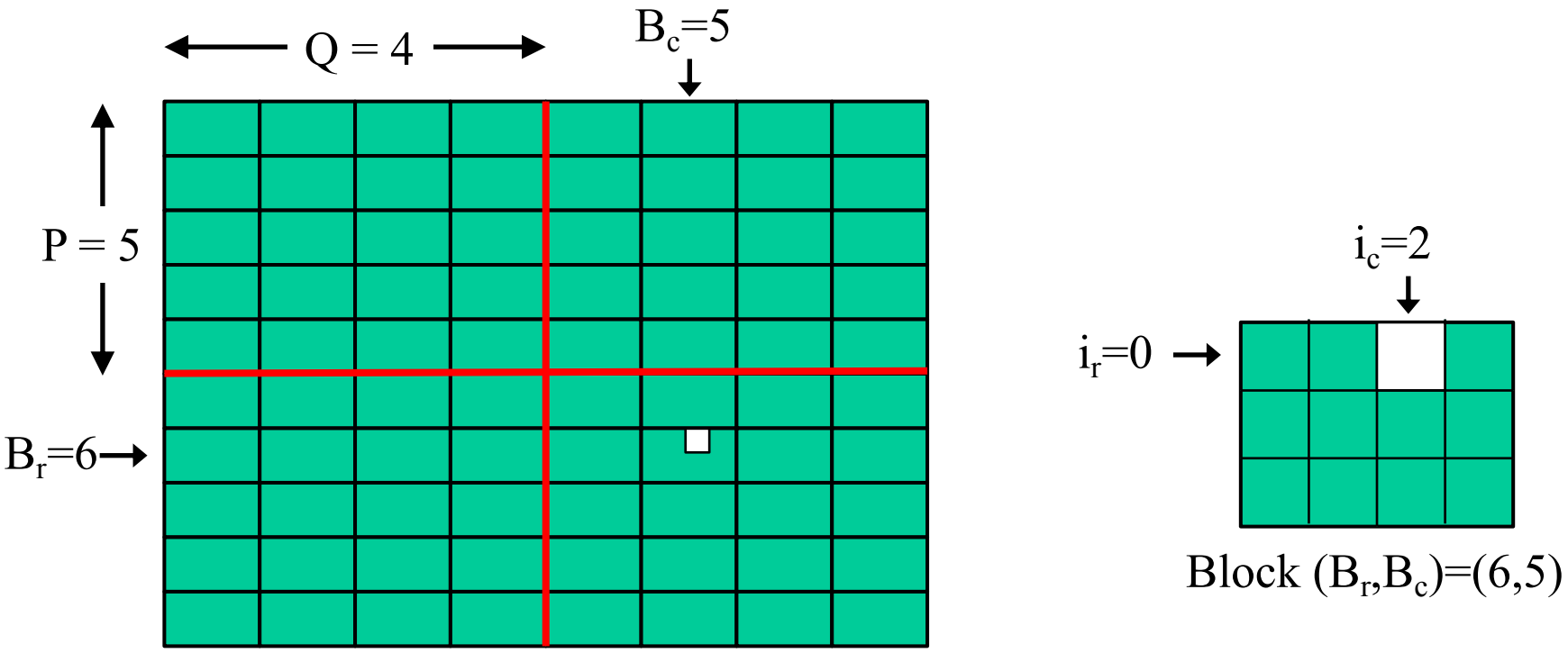
  $B_r = 18/3 = 6$,

  $p = B(\text{mod } P) = 1$, $b_r = B/P = 1$, $i_r = m(\text{mod } k_r) = 0$
- Columns: n = 22, k_c = 4, Q = 4

  $B_c = 22/4 = 5$,

  $q = B(\text{mod } Q) = 1$, $b_c = B/Q = 1$, $i_c = n(\text{mod } k_c) = 2$

Position of (18,22)



- Each green rectangle is a 3x4 block
- Position in process mesh is (p,q) = (1,1)
- Local block position is (b_r,b_c) = (1,1)

Layout of Global Blocks: Matrix View

B_r \ B_c	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
p,q																
0	0,0	0,1	0,2	0,3	0,0	0,1	0,2	0,3	0,0	0,1	0,2	0,3	0,0	0,1	0,2	0,3
1	1,0	1,1	1,2	1,3	1,0	1,1	1,2	1,3	1,0	1,1	1,2	1,3	1,0	1,1	1,2	1,3
2	2,0	2,1	2,2	2,3	2,0	2,1	2,2	2,3	2,0	2,1	2,2	2,3	2,0	2,1	2,2	2,3
3	0,0	0,1	0,2	0,3	0,0	0,1	0,2	0,3	0,0	0,1	0,2	0,3	0,0	0,1	0,2	0,3
4	1,0	1,1	1,2	1,3	1,0	1,1	1,2	1,3	1,0	1,1	1,2	1,3	1,0	1,1	1,2	1,3
5	2,0	2,1	2,2	2,3	2,0	2,1	2,2	2,3	2,0	2,1	2,2	2,3	2,0	2,1	2,2	2,3
6	0,0	0,1	0,2	0,3	0,0	0,1	0,2	0,3	0,0	0,1	0,2	0,3	0,0	0,1	0,2	0,3
7	1,0	1,1	1,2	1,3	1,0	1,1	1,2	1,3	1,0	1,1	1,2	1,3	1,0	1,1	1,2	1,3
8	2,0	2,1	2,2	2,3	2,0	2,1	2,2	2,3	2,0	2,1	2,2	2,3	2,0	2,1	2,2	2,3
9	0,0	0,1	0,2	0,3	0,0	0,1	0,2	0,3	0,0	0,1	0,2	0,3	0,0	0,1	0,2	0,3
10	1,0	1,1	1,2	1,3	1,0	1,1	1,2	1,3	1,0	1,1	1,2	1,3	1,0	1,1	1,2	1,3
11	2,0	2,1	2,2	2,3	2,0	2,1	2,2	2,3	2,0	2,1	2,2	2,3	2,0	2,1	2,2	2,3

Layout of Global Blocks: Process Memory View

B_r, B_c	q = 0				q = 1				q = 2				q = 3			
p = 0	0,0	0,4	0,8	0,12	0,1	0,5	0,9	0,13	0,2	0,6	0,10	0,14	0,3	0,7	0,11	0,15
	3,0	3,4	3,8	3,12	3,1	3,5	3,9	3,13	3,2	3,6	3,10	3,14	3,3	3,7	3,11	3,15
	6,0	6,4	6,8	6,12	6,1	6,5	6,9	6,13	6,2	6,6	6,10	6,14	6,3	6,7	6,11	6,15
	9,0	9,4	9,8	9,12	9,1	9,5	9,9	9,13	9,2	9,6	9,10	9,14	9,3	9,7	9,11	9,15
p = 1	1,0	1,4	1,8	1,12	1,1	1,5	1,9	1,13	1,2	1,6	1,10	1,14	1,3	1,7	1,11	1,15
	4,0	4,4	4,8	4,12	4,1	4,5	4,9	4,13	4,2	4,6	4,10	4,14	4,3	4,7	4,11	4,15
	7,0	7,4	7,8	7,12	7,1	7,5	7,9	7,13	7,2	7,6	7,10	7,14	7,3	7,7	7,11	7,15
	10,0	10,4	10,8	10,12	10,1	10,5	10,9	10,13	10,2	10,6	10,10	10,14	10,3	10,7	10,11	10,15
p = 2	2,0	2,4	2,8	2,12	2,1	2,5	2,9	2,13	2,2	2,6	2,10	2,14	2,3	2,7	2,11	2,15
	5,0	5,4	5,8	5,12	5,1	5,5	5,9	5,13	5,2	5,6	5,10	5,14	5,3	5,7	5,11	5,15
	8,0	8,4	8,8	8,12	8,1	8,5	8,9	8,13	8,2	8,6	8,10	8,14	8,3	8,7	8,11	8,15
	11,0	11,4	11,8	11,12	11,1	11,5	11,9	11,13	11,2	11,6	11,10	11,14	11,3	11,7	11,11	11,15

Load Balancing Issues in a Parallel Cellular Automata Application

- This looks at an application that uses a cyclic data distribution to achieve static load balance.
- As in WaTor, data inconsistency in how updates are performed is an issue

CA for Surface Reactions

- A cellular automaton is used to model the reaction of carbon monoxide and oxygen to form carbon dioxide

$$CO + O \longleftrightarrow CO_2$$

- Reactions take place on surface of a crystal which serves as a catalyst.

The Problem Domain

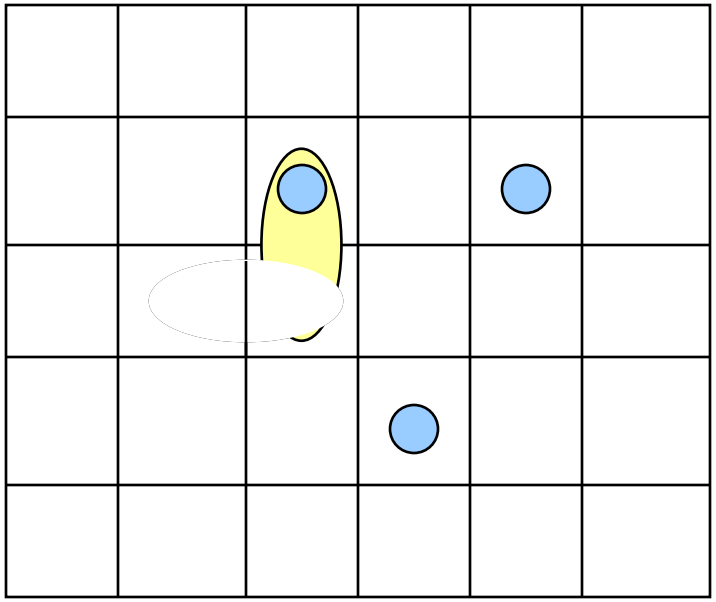
- The problem domain is a periodic square lattice representing the crystal surface.
- CO and O_2 are adsorbed onto the crystal surface from the gas phase.
- Parameter y is the fraction of CO and 1-y is the fraction of O_2.

Interaction Rules

- Choose a lattice site at random and attempt to place a CO or an O_2 there with probabilities y and 1-y, respectively.
- If site is occupied then the CO or O_2 bounces off, and a new trial begins.
- O_2 disassociates so we have to find 2 adjacent sites for these.
- The following rules determine what happens next.

Interaction Rules for CO

1. CO adsorbed
2. Check 4 neighbors for O
3. CO and O react
4. CO_2 desorbs

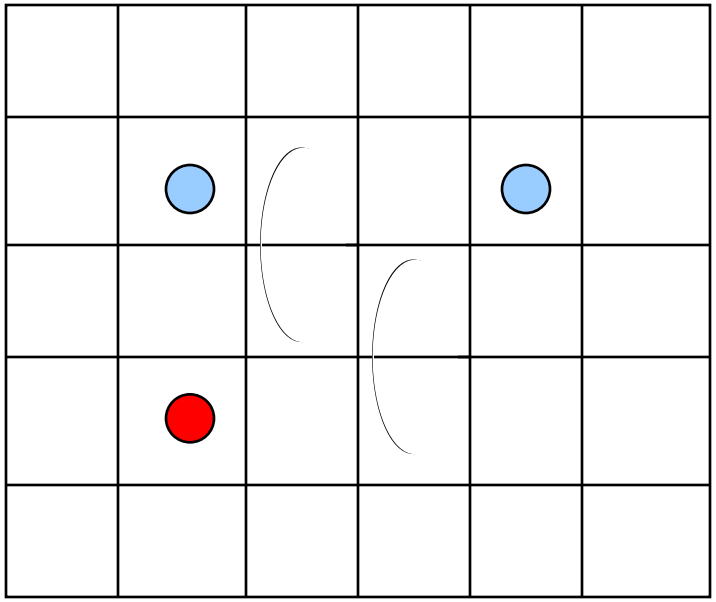


oxygen

CO

Interaction Rules for O

1. O_2 adsorbed
2. O_2 disassociates
3. Check 6 neighbors for CO
4. O and CO react
5. CO_2 desorbs



oxygen
CO
O_2

Parallel Version of Code

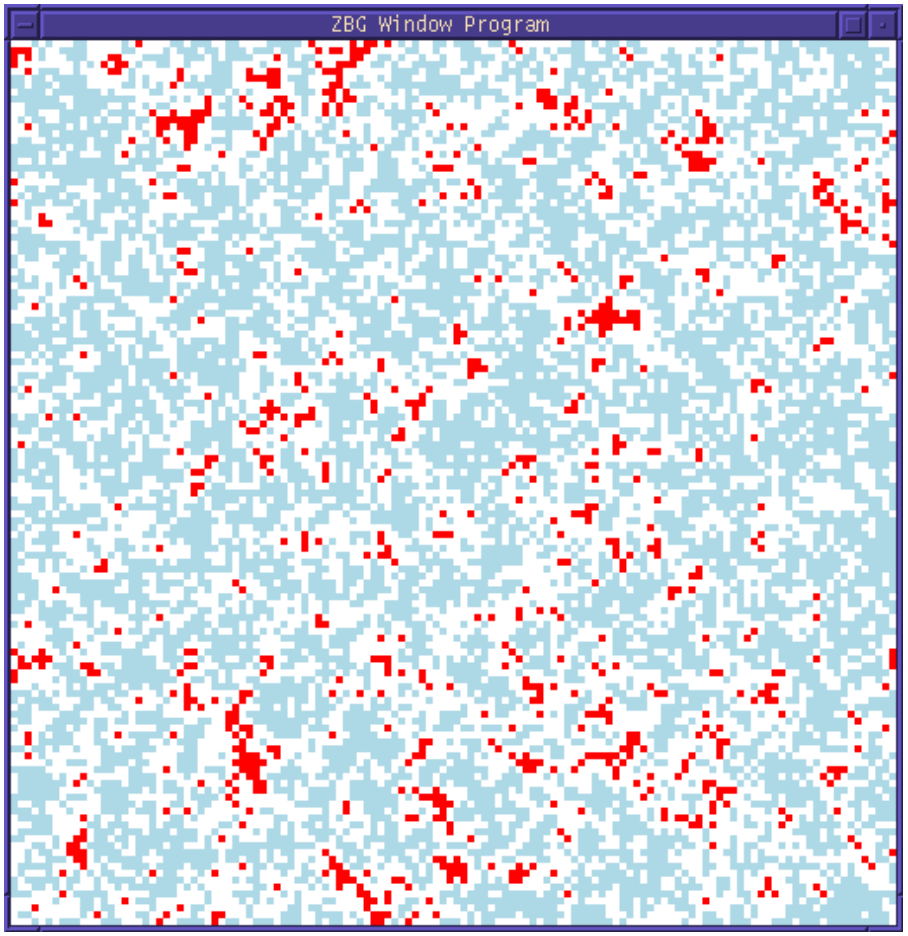
- As simulation evolves the distribution of molecules may become very uneven.
- This results in load imbalance.
- Use a 2-D block cyclic data distribution for the lattice.
- This will give statistical load balance, but smaller block sizes will result in more communication.

Steady State Reaction

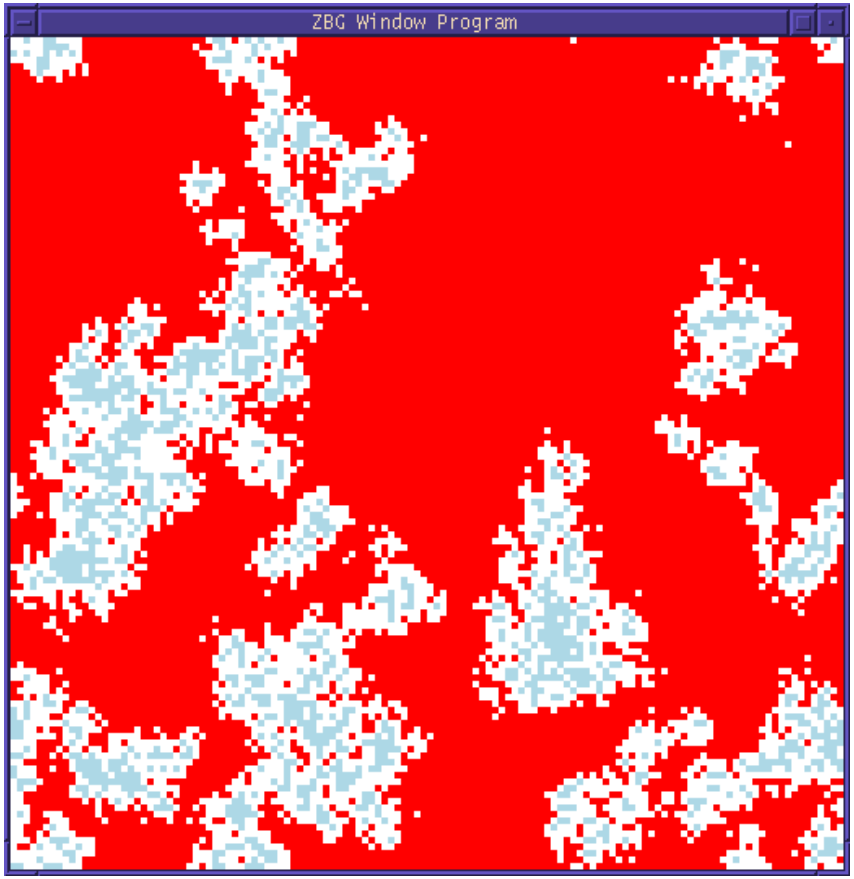
For $y_1<y<y_2$ we get a steady state.

$y_1 \approx 0.39$

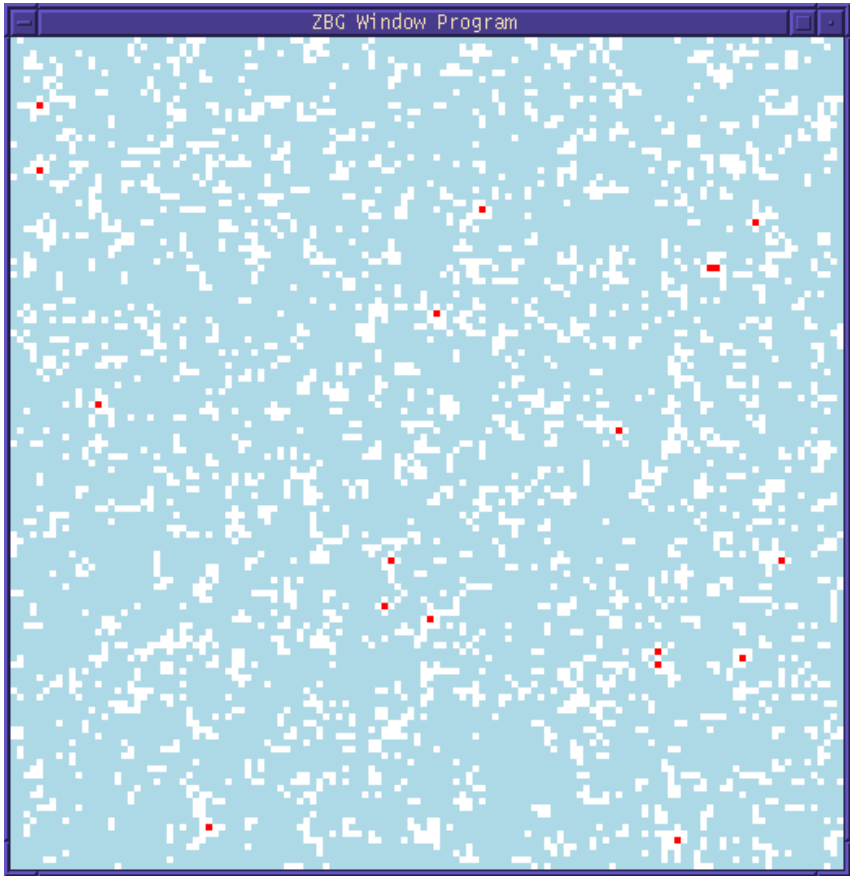
$y_2 \approx 0.53$



CO Poisoning: $y > y_2$



Oxygen Poisoning: $y < y_1$



Main Issues

- MPI used – user-defined datatypes were important in performing communication.
- There is a trade-off between load imbalance and communication.
- A block-cyclic data distribution is used.
- Performance can be modelled.

Block-Cyclic Data Distribution

Block-cyclic data distribution improves load balance by scattering processes over the lattice in a regular way.

Block size is k_r x k_c

(0,0)	(0,1)	(0,2)	(0,3)	(0,0)	(0,1)	(0,2)	(0,3)	(0,0)	(0,1)	(0,2)	(0,3)
(1,0)	(1,1)	(1,2)	(1,3)	(1,0)	(1,1)	(1,2)	(1,3)	(1,0)	(1,1)	(1,2)	(1,3)
(2,0)	(2,1)	(2,2)	(2,3)	(2,0)	(2,1)	(2,2)	(2,3)	(2,0)	(2,1)	(2,2)	(2,3)
(3,0)	(3,1)	(3,2)	(3,3)	(3,0)	(3,1)	(3,2)	(3,3)	(3,0)	(3,1)	(3,2)	(3,3)
(0,0)	(0,1)	(0,2)	(0,3)	(0,0)	(0,1)	(0,2)	(0,3)	(0,0)	(0,1)	(0,2)	(0,3)
(1,0)	(1,1)	(1,2)	(1,3)	(1,0)	(1,1)	(1,2)	(1,3)	(1,0)	(1,1)	(1,2)	(1,3)
(2,0)	(2,1)	(2,2)	(2,3)	(2,0)	(2,1)	(2,2)	(2,3)	(2,0)	(2,1)	(2,2)	(2,3)
(3,0)	(3,1)	(3,2)	(3,3)	(3,0)	(3,1)	(3,2)	(3,3)	(3,0)	(3,1)	(3,2)	(3,3)
(0,0)	(0,1)	(0,2)	(0,3)	(0,0)	(0,1)	(0,2)	(0,3)	(0,0)	(0,1)	(0,2)	(0,3)
(1,0)	(1,1)	(1,2)	(1,3)	(1,0)	(1,1)	(1,2)	(1,3)	(1,0)	(1,1)	(1,2)	(1,3)
(2,0)	(2,1)	(2,2)	(2,3)	(2,0)	(2,1)	(2,2)	(2,3)	(2,0)	(2,1)	(2,2)	(2,3)
(3,0)	(3,1)	(3,2)	(3,3)	(3,0)	(3,1)	(3,2)	(3,3)	(3,0)	(3,1)	(3,2)	(3,3)

Parallel Implementation

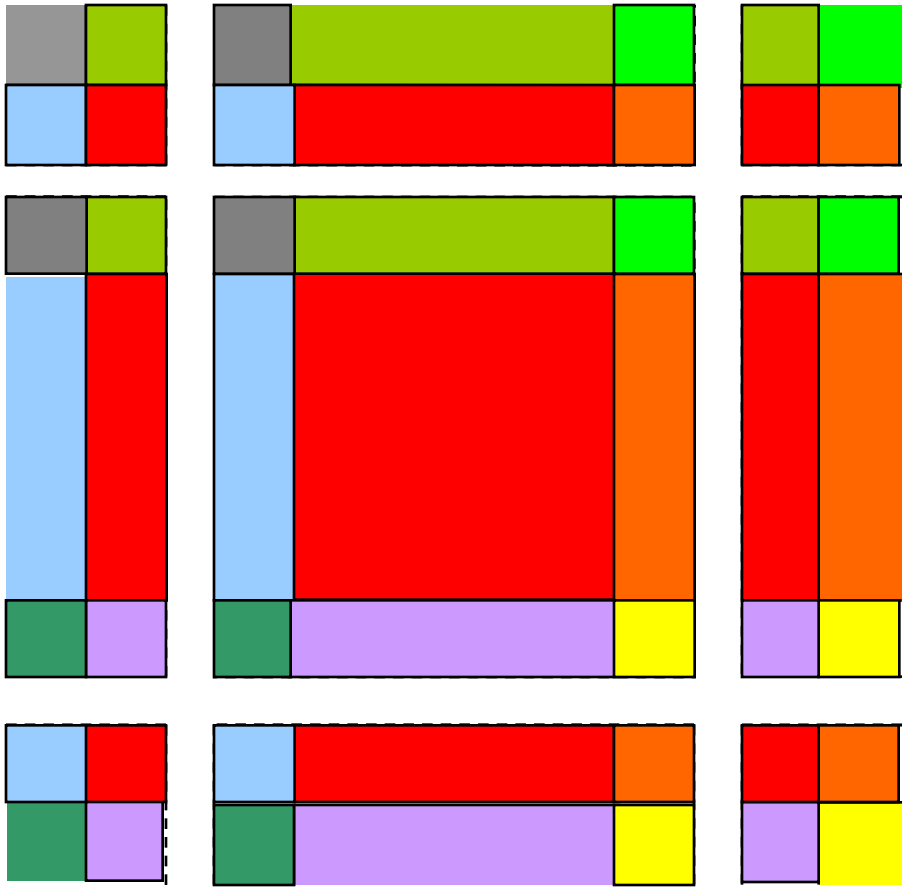
- Processes need to communicate their boundary data to neighboring processes.
- Sites within two sites from the boundary must be communicated.
- Each process can generate random numbers independently.

A Communication Strategy

- Do a left shift: send leftmost 2 columns left while receiving from the right.
- Do a right shift: send rightmost 2 columns right while receiving from the left.
- Similarly for up shifts and down shifts.
- After these 4 shifts have been done each process can update all its lattice sites.

Communication Shifts

1. Left shift.
2. Right shift.
3. Up shift.
4. Down shift.

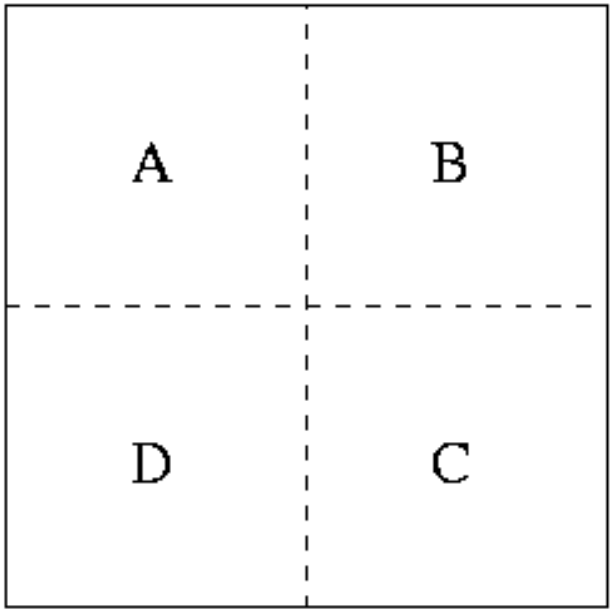


Update Conflicts

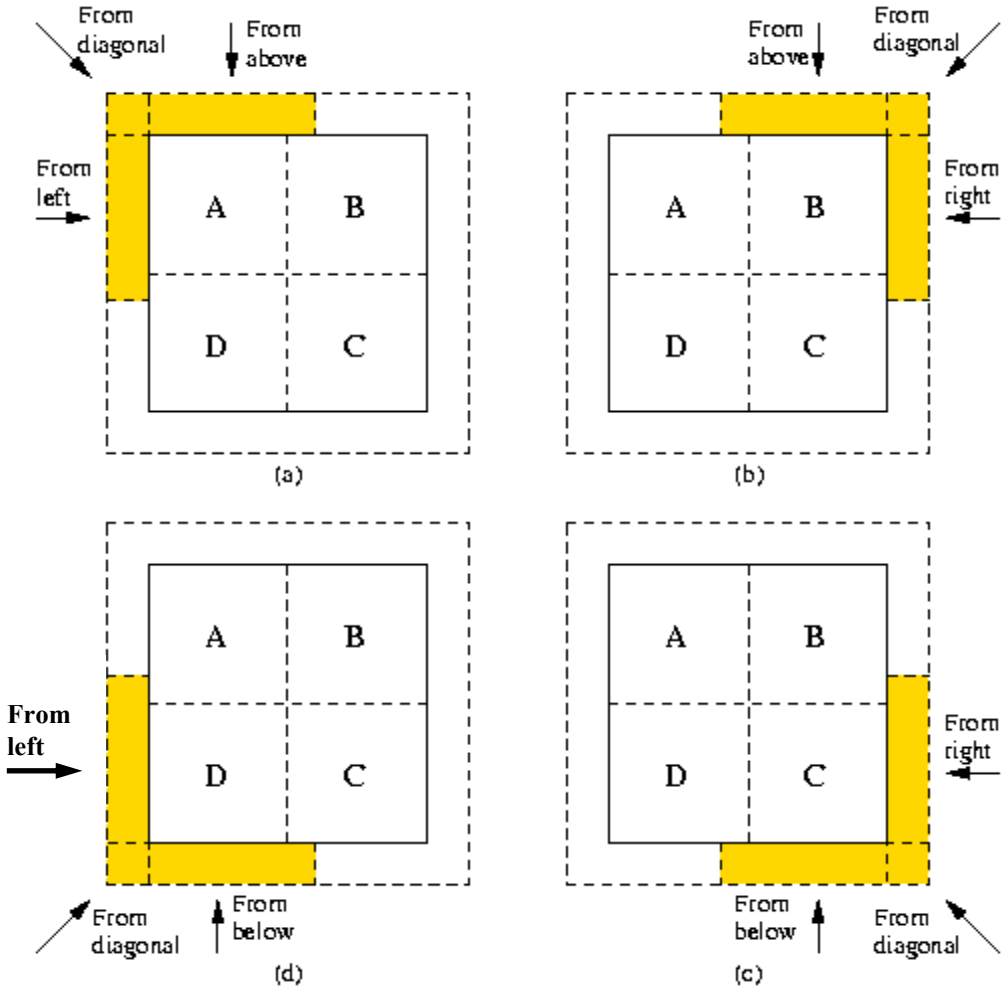
- Two adjacent processes can concurrently update the same lattice site close to their common boundary.
- This is an *update conflict*.
- Avoid conflicts by never updating adjacent areas in processes concurrently.
- Use sub-partitioning to do this.

Sub-partitioning

- First each process updates A, then B, C, and D.
- Before updating a sub-partition communication is needed to ensure each process has all the data to update its points.
- After updating a sub-partition the data is sent back to the process it came from.



Communication Before Update

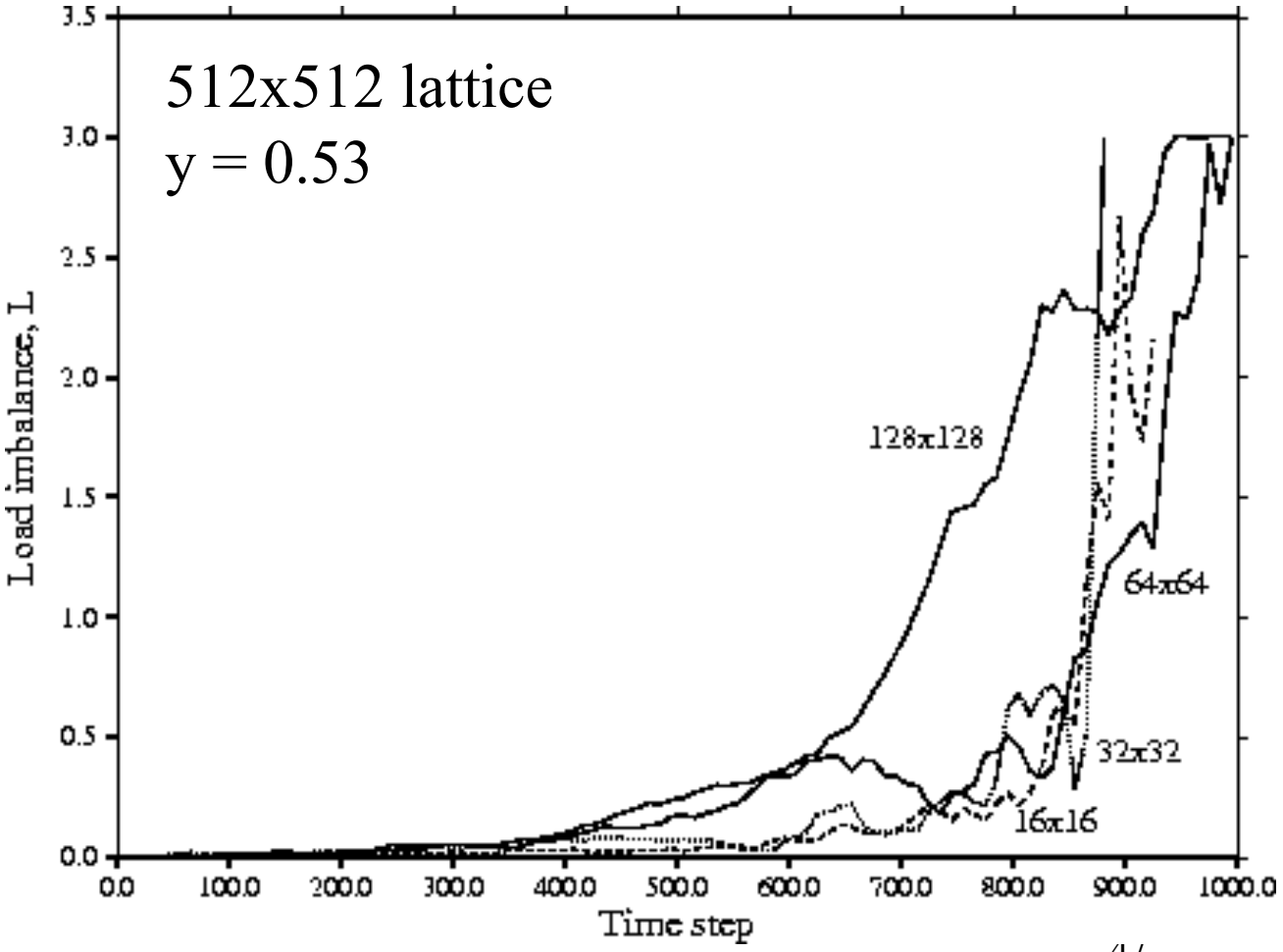


Load Imbalance

2x2 process mesh used.

Load imbalance is smaller for smaller block sizes.

Load imbalance is large as CO poisoning occurs.

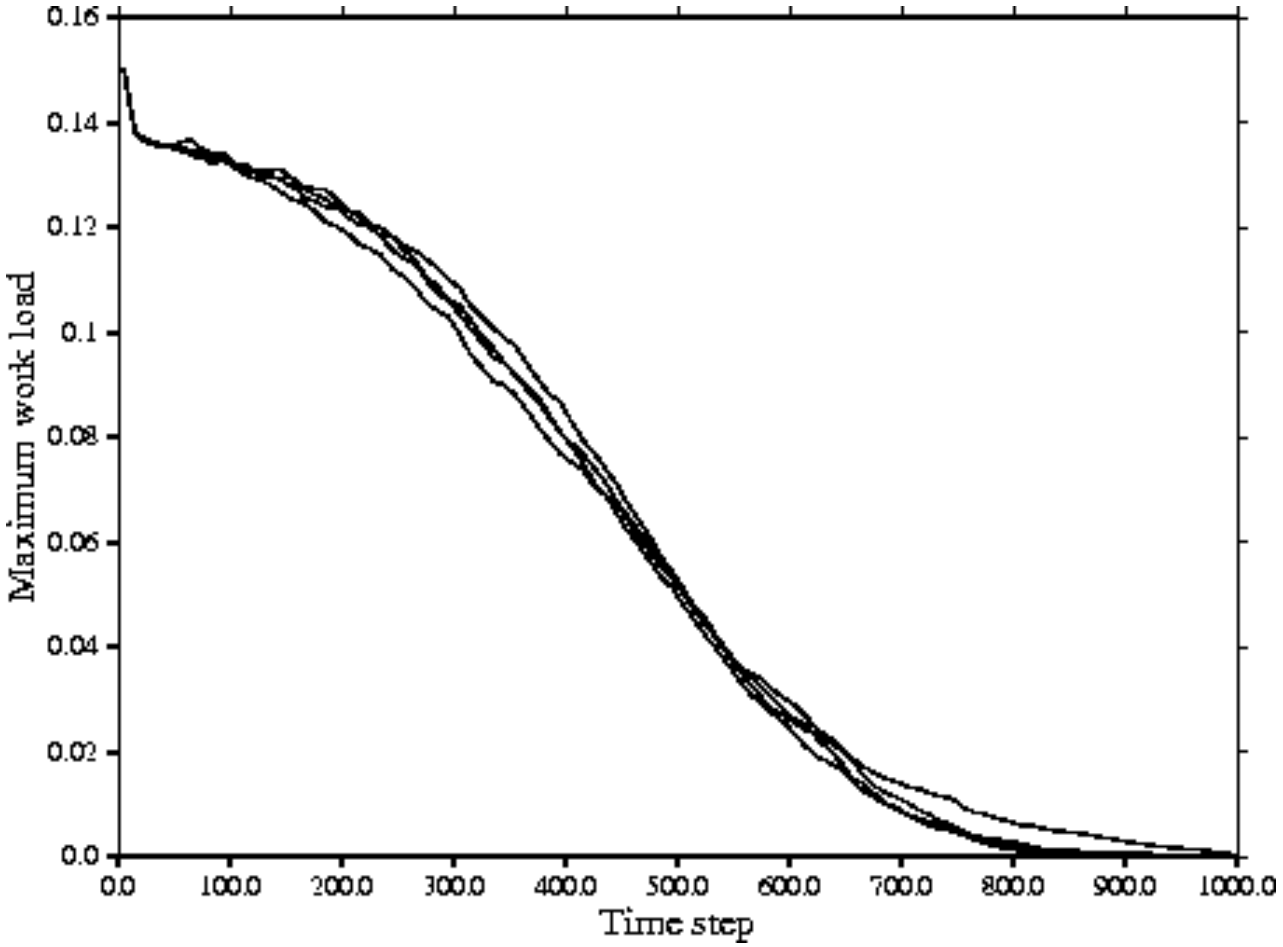


Maximum Work Load

Maximum work load is similar for different block size, except after step 700.

Not much work available at this time.

Load imbalance not very important!

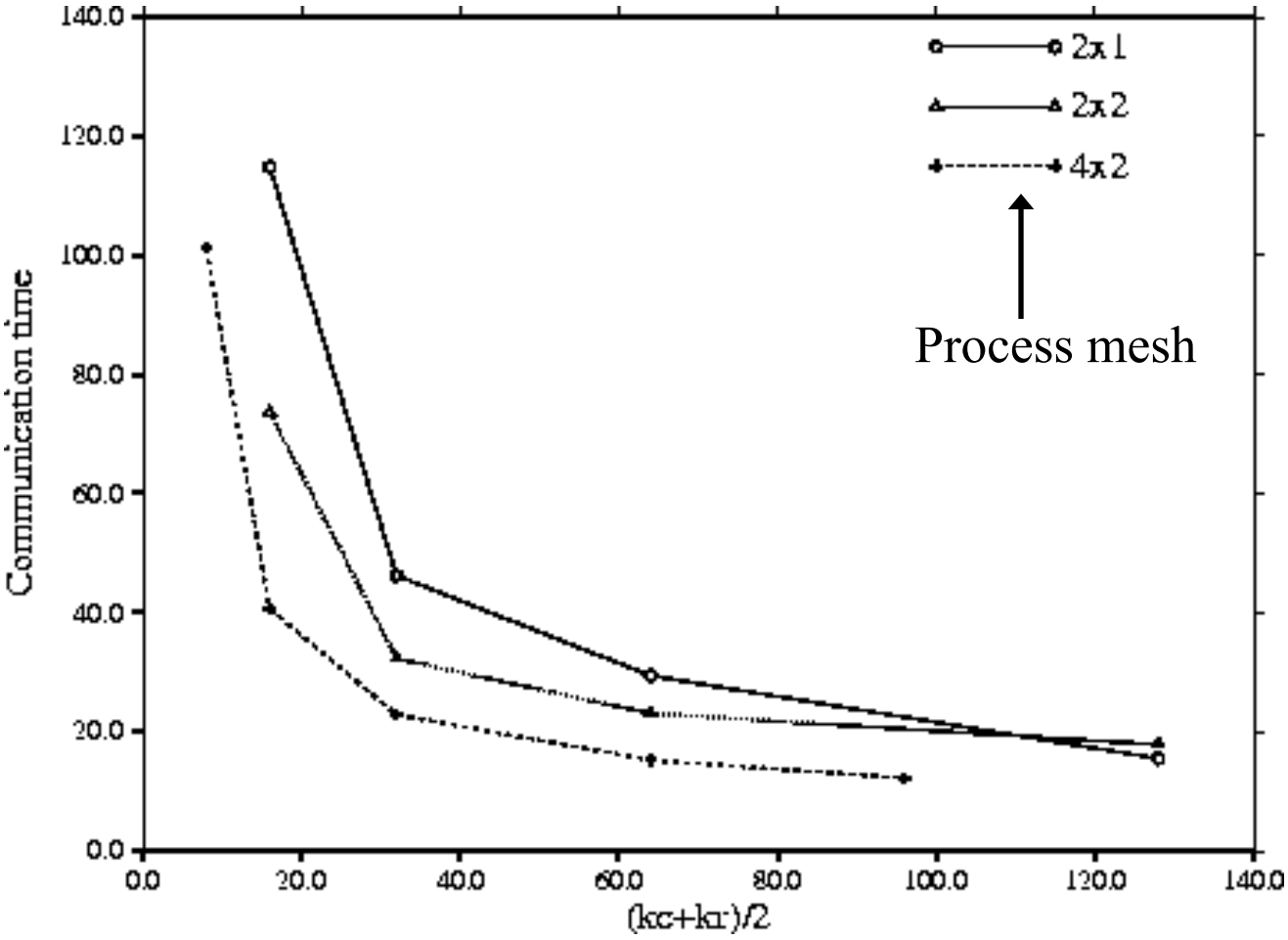


Communication Time

512x512 lattice,
y = 0.53

For given problem communication is smaller for more processes - less data per process.

Smaller blocks require more communication.



Performance Model

- Amount of communication and computation both depend linearly on problem size.
- Speed-up is independent of problem size and is given by:

$$T_{seq} = CM^2 t_{calc}$$

$$T_{par} = \left(\frac{CM^2}{N}\right) t_{calc} + \mathrm{B}\left(\frac{M^2}{k_r k_c}\right)\frac{1}{N}\left(2(k_r + k_c)\right) t_{comm}$$

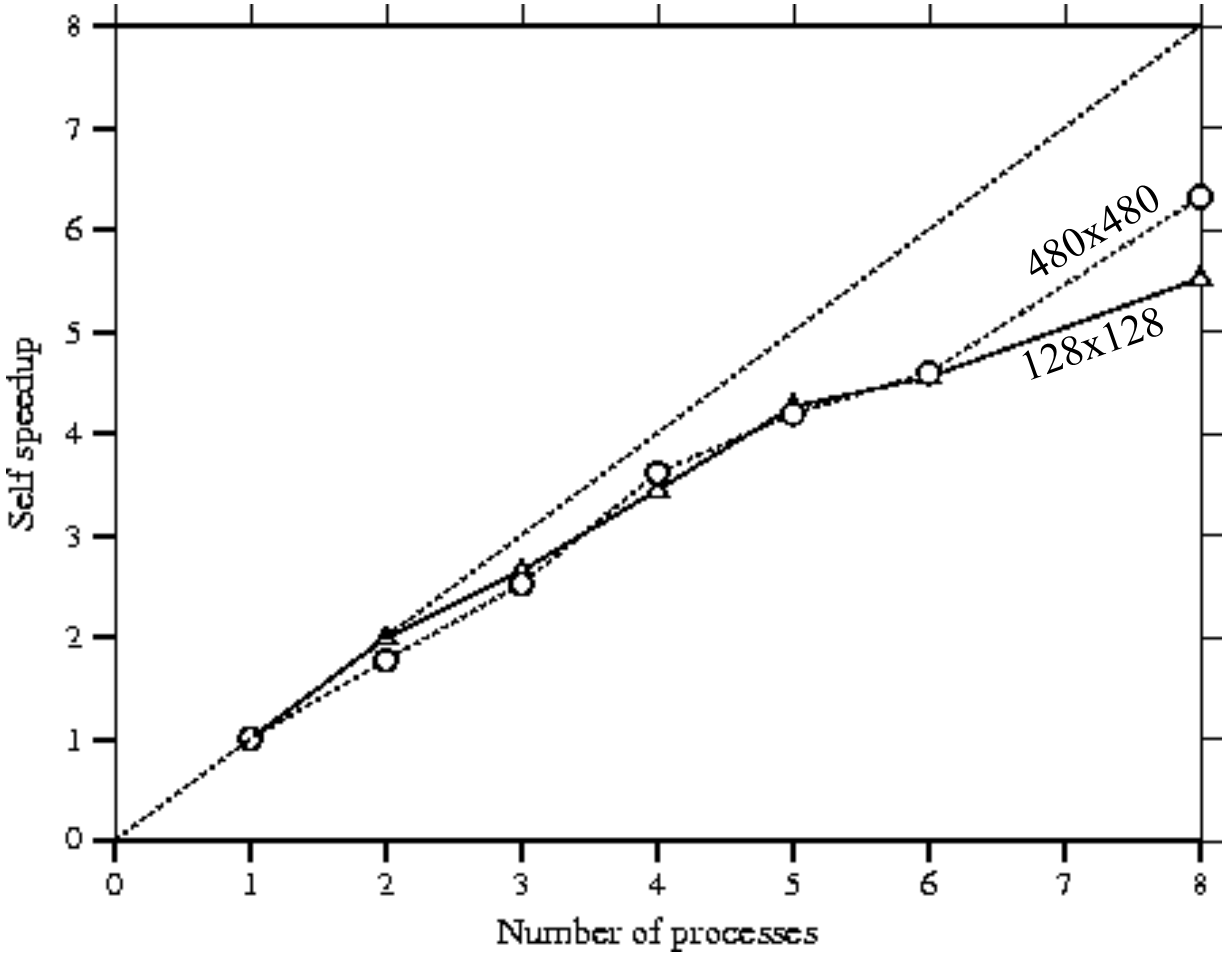
$$\mathrm{S} = \frac{\mathrm{N}}{1 + \mathrm{A}\,\tau\,(\mathrm{k_r}+\mathrm{k_c})/(\mathrm{k_r}\,\mathrm{k_c})}$$

Number of blocks per process

Perimeter of block

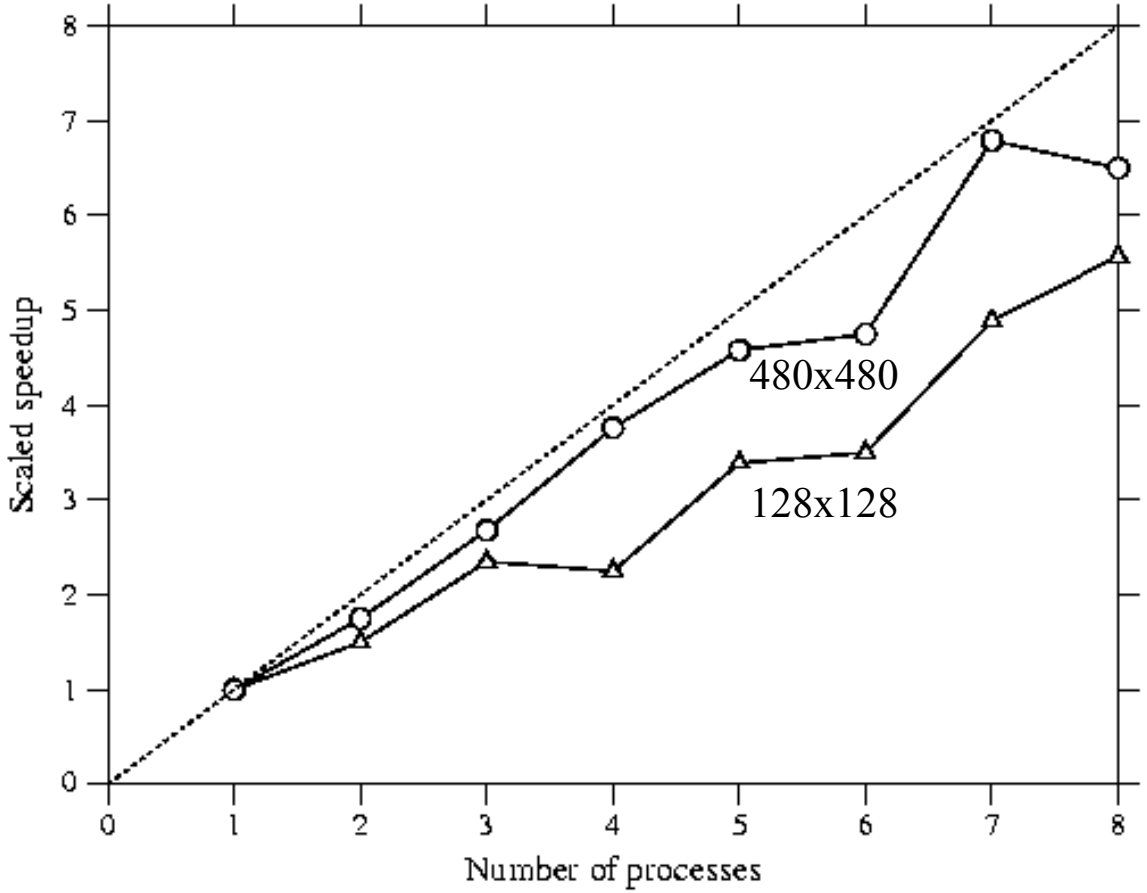
Self Speed-Up

As expected, speed-up is independent of problem size (except at 8!)



Scaled Speed-Up

Fixed problem size per process



Summary

- It turns out that load imbalance is not very important in this problem.
- Load imbalance will be important in cellular automata with more complex geometries.
- Easy to modify code for other CA problems.
- Speed-up independent of problem size.