

Lecture 17

Message Passing
Programming and Performance

Announcements

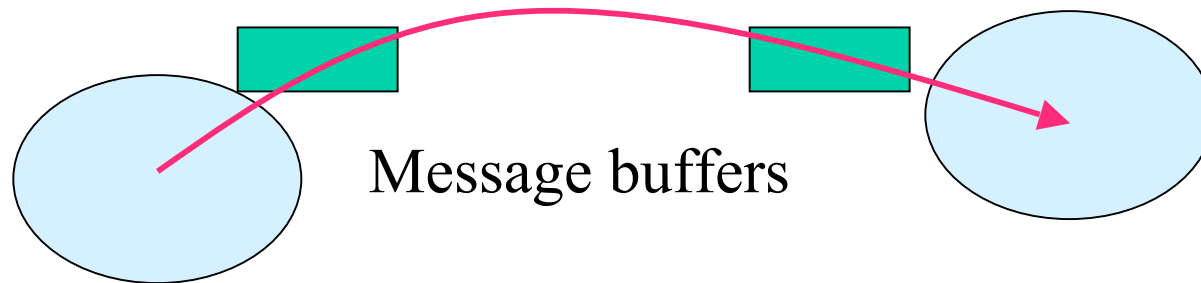
- Abe will be down on 12/1/08, 7AM to 3PM PST, so plan accordingly

Programming with Message Passing

- **The** primary model for implementing high performance applications
- Explains fundamental behavior in multi- and single processors
- Programs execute as a set of P processes
 - We specify P when we run the program
 - Assume each process is assigned a different physical processor
- Each physical process
 - is initialized with the same code, but has private state
SPMD = “Same Program Multiple Data”
 - executes instructions at its own rate
 - has an associated *rank*, a unique integer in the range $0:P-1$
- The sequence of instructions each process executes depends on its rank and the messages it sends and receives
- Program execution is often called “bulk synchronous” or “loosely synchronous”

Message Passing

- Prevailing model for high performance computing
- Requires a sender and an explicit recipient
 - Must be aware of one another
- Messages are like email; to send one we specify
 - A destination
 - A message body (can be empty)
- To receive we need a receptacle to hold the incoming data
- Message passing performs two events
 - Memory to memory block copy
 - Synchronization signal on receiving end: “Data arrived”



A minimal interface

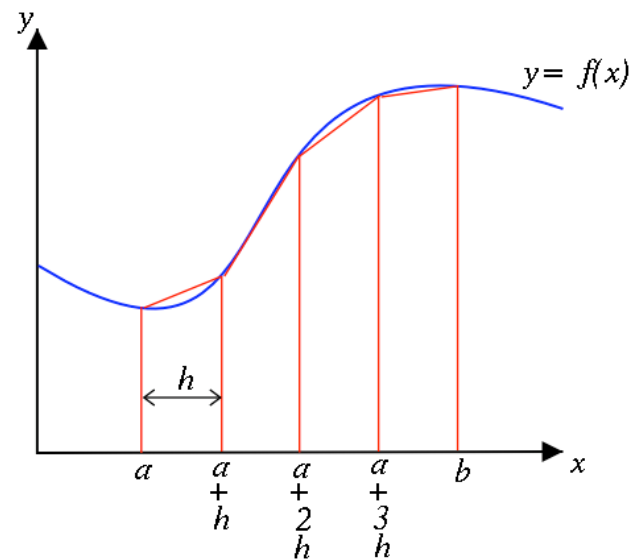
- Query functions
 - `nproc( )` = # processors
 - `myRank( )` = this process's rank
- *Point-to-point* communication
 - Simplest form of communication
 - Send a message to another process
`Send(Object, Destination process ID)`
 - Receive a message from another process
`Receive(Object)`
`Receive(Source process, Object)`
- There are also asynchronous variants (`iSend/Recv`)

A first application

The trapezoidal rule

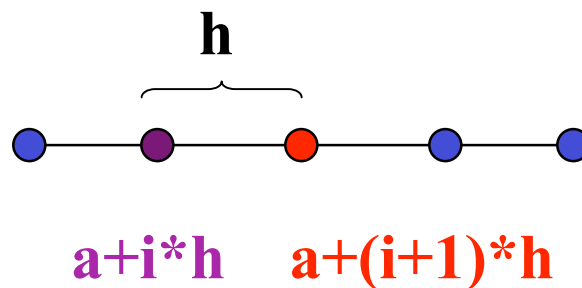
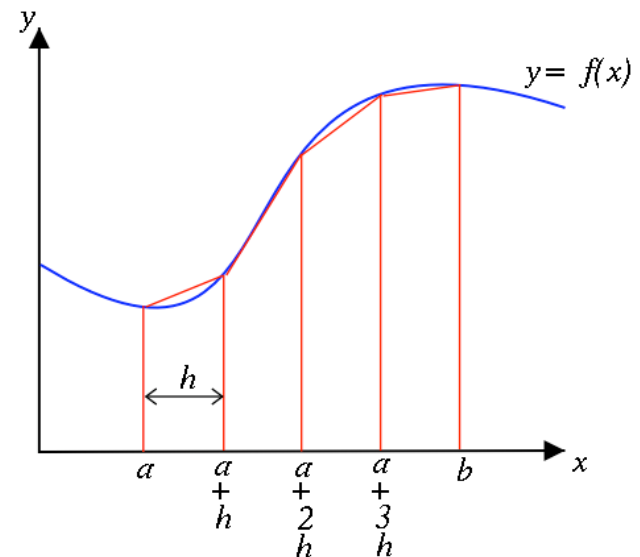
- Use the trapezoidal rule to numerically approximate the definite integral

$$\int_a^b f(x) dx$$



How the trapezoidal rule works

- Divide the interval $[a,b]$ into n segments of size $h=1/n$
- Approximate the area under an interval using a trapezoid
- Area under the i^{th} trapezoid
 $\frac{1}{2} (f(a+i \times h) + f(a+(i+1) \times h)) \times h$
- Area under the entire curve
 $\approx$ sum of all the trapezoids



Reference material

- For a discussion of the trapezoidal rule

<http://metric.ma.imperial.ac.uk/numerics/trapeziumRule/index.html>

- A applet to carry out integration

<http://www.csse.monash.edu.au/~lloyd/tildeAlgDS/Numerical/Integration>

- Code (from Pacheco hard copy text)

Serial Code

[\\$PUB/Examples/Pacheco/ppmpi_c/chap04/serial.c](#)

Parallel Code

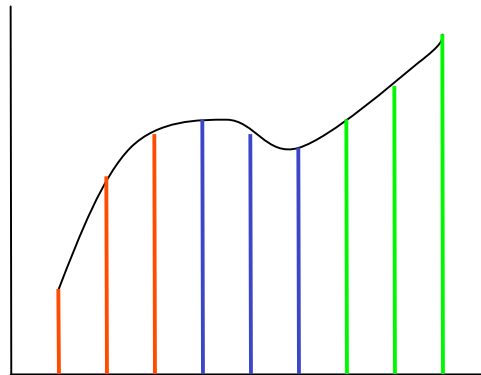
[\\$PUB/Examples/Pacheco/ppmpi_c/chap04/trap.c](#)

Serial code (Following Pacheco)

```
main() {  
    float f(float x) { return x*x; }           // Function we're integrating  
  
    float h = (b-a)/n;                          // h = trapezoid base width  
                                                // a and b: endpoints  
                                                // n = # of trapezoids  
  
    float integral = (f(a) + f(b))/2.0;  
  
    float x; int i;  
  
    for (i = 1, x=a; i <= n-1; i++) {  
        x += h;  
        integral = integral + f(x);  
    }  
    integral = integral*h;  
}
```

The parallel algorithm

- Partition the integration interval into sub-intervals, one per processor
- Each processor computes the integral on its local subdomain
- All combine their local integrals into a global one



First version of the parallel code

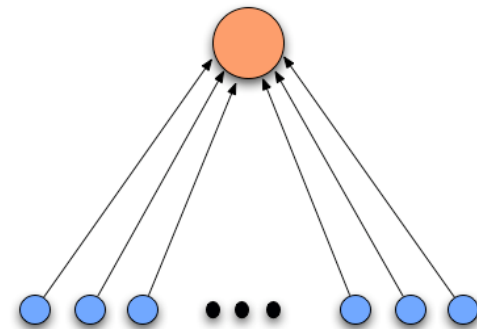
```
local_n = n/p; // Number of trapezoids; assume p divides n evenly
float local_a = a + my_rank*local_n*h,
      local_b = local_a + local_n*h,
      integral = Trap(local_a, local_b, local_n, h);

if (my_rank == ROOT) { // Sum the integrals calculated by all the
    processes
    total = integral;
    for (source = 1; source < p; source++) {
        MPI_Recv(&integral, 1, MPI_FLOAT, source, tag, WORLD, &status);
        total += integral;
    }
} else
    MPI_Send(&integral, 1, MPI_FLOAT, ROOT, tag, WORLD);
```

Improvements

- The result does not depend on the order in which the sums are taken, except to within roundoff
- We use a linear time algorithm to accumulate contributions, but there are other orderings

```
for (source = 1; source < p; source++) {  
    MPI_Recv(&integral, 1, MPI_FLOAT,  
            MPI_ANY_SOURCE, tag,  
            WORLD, &status);  
  
    total += integral;  
}
```



Improved parallel code

- We can often improve performance by taking advantage of global knowledge about communication
- Instead of using point to point communication operations to accumulate the sum, use *collective* communication **nt**

```
local_n = n/p;  
float local_a = a + my_rank*local_n*h,  
      local_b = local_a + local_n*h,  
      integral = Trap(local_a, local_b, local_n, h);  
MPI_Reduce( &integral, &total, 1,  
            MPI_FLOAT, MPI_SUM,  
            ROOT,WORLD)
```

Collective communication in MPI

- Collective operations are called by **all** processes within a communicator
- Broadcast: distribute data from a designated “root” process to all the others
`MPI_Bcast(in, count, type, root, comm)`
- Reduce: combine data from all processes and return to a designated root process
`MPI_Reduce(in, out, count, type, op, root, comm)`

Broadcast

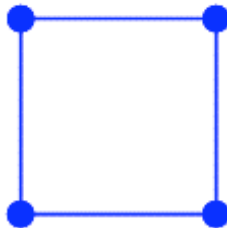
- The root process transmits of m pieces of data to all the $p-1$ other processors
- With the linear ring algorithm this processor performs $p-1$ sends of length m
 - Cost is $(p-1)(\alpha + \beta m)$
- Another approach is to use the *hypercube algorithm*, which has a logarithmic running time

What is a hypercube?

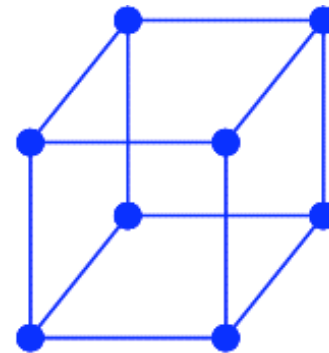
- A hypercube is a d -dimensional graph with 2^d nodes
- A 0-cube is a single node, 1-cube is a line connecting two points, 2-cube is a square, etc
- Each node has d neighbors



1D



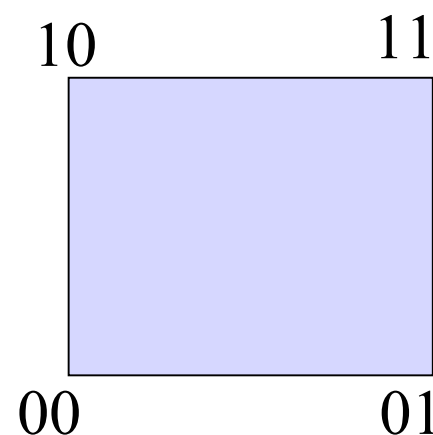
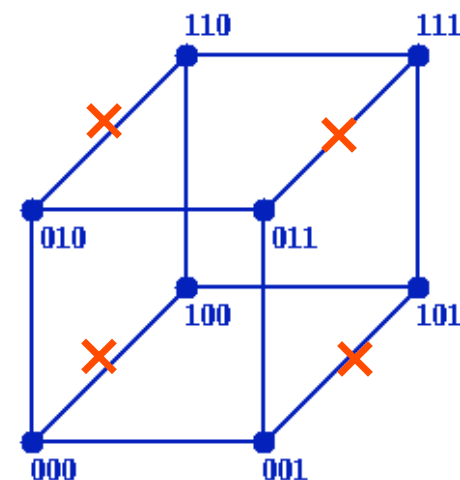
2D



3D

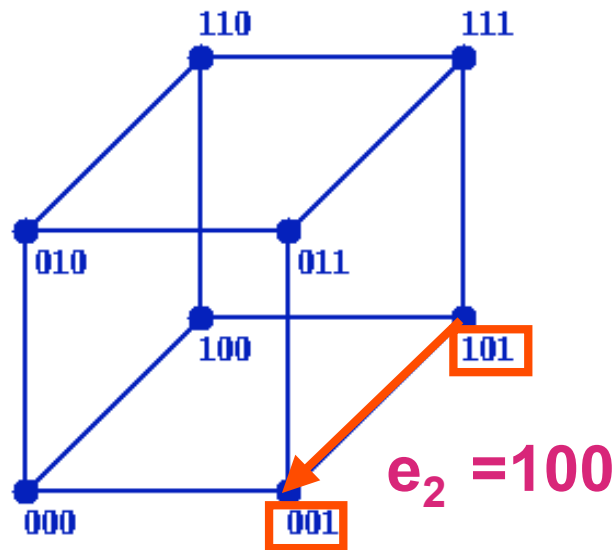
Properties of hypercubes

- A hypercube with p nodes has $\lg(p)$ dimensions
- *Inductive construction*: we may construct a d -cube from two $(d-1)$ dimensional cubes
- **Diameter**: What is the maximum distance between any 2 nodes?
- **Bisection bandwidth**: How many cut edges (mincut)



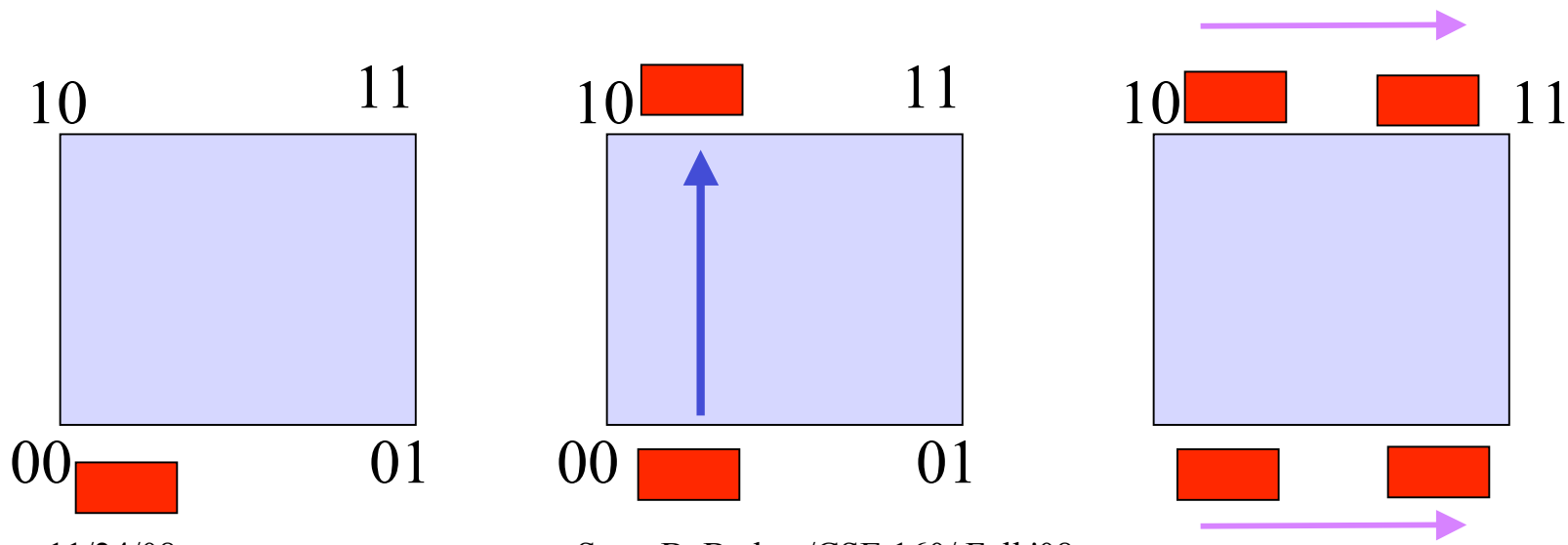
Bookkeeping

- Label nodes with a binary reflected grey code
<http://www.nist.gov/dads/HTML/graycode.html>
- Neighboring labels differ in exactly one bit position $001 = 101 \otimes e_2$, $e_2 = 100$



Hypercube broadcast algorithm with $p=4$

- Processor 0 is the root, sends its data to its hypercube “buddy” on processor 2 (10)
- Proc 0 & 2 send data to respective buddies



Reduction

- We may use the hypercube algorithm to perform reductions as well as broadcasts
- Another variant of reduction provides all processes with a copy of the reduced result

Allreduce()

- Equivalent to a *Reduce* + *Bcast*
- A clever algorithm performs an *Allreduce* in one phase rather than having perform separate reduce and broadcast phases

Final version

```
int local_n = n/p;  
  
float local_a = a + my_rank*local_n*h,  
      local_b = local_a + local_n*h,  
      integral = Trap(local_a, local_b, local_n, h);  
  
MPI_Allreduce( &integral, &total, 1,  
              MPI_FLOAT, MPI_SUM, WORLD)
```

Relevance to multicore computing

- Recall how we accumulated quantities with threads: $O(P)$ algorithm
Critical section:
Global value += local value
- Similar to the slow version of reduction used in the trapezoidal rule
- In the future, multicore processors may be NUMA
- Apply the techniques used in message passing to reduce the running time to $O(\lg(P))$

Image Smoothing Performance

Image smoothing



Original



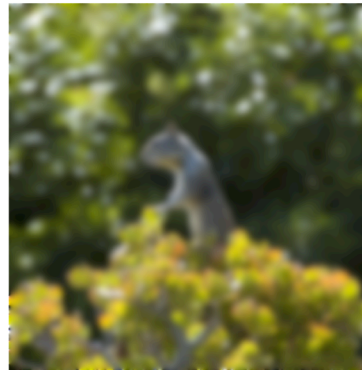
15 iter



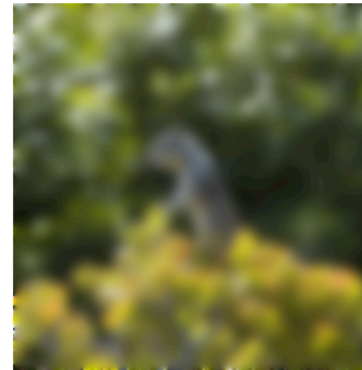
50 iter



100 iter



300 iter



1000 iter

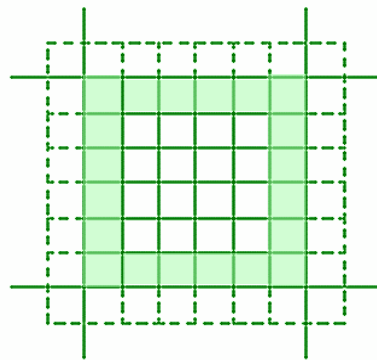
Image smoothing algorithm

- Repeat as many times as needed

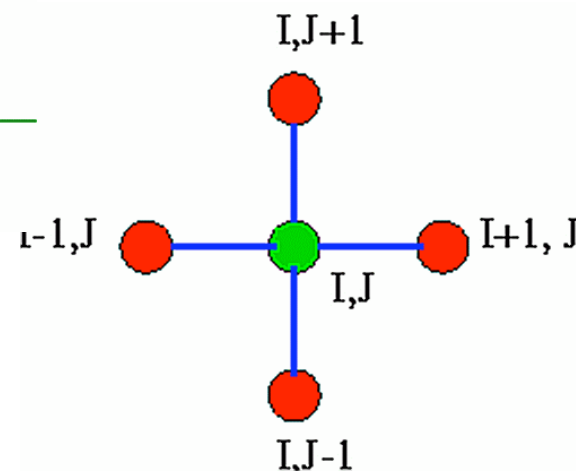
for (i,j) in 0:N-1 x 0:N-1

$$u'[i,j] = (u[i-1,j] + u[i+1,j] + u[i,j-1] + u[i,j+1])/4$$

$u = u'$

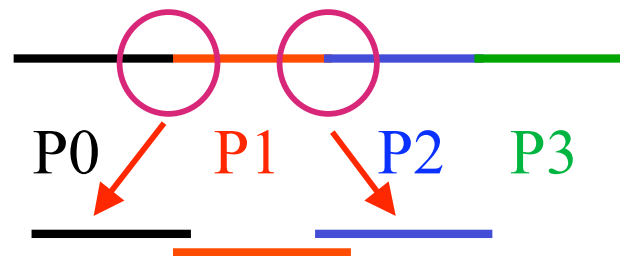
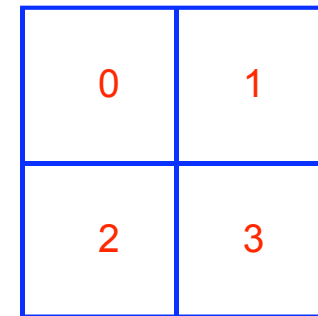
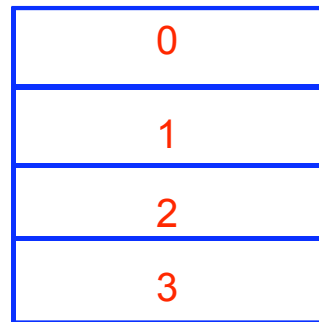
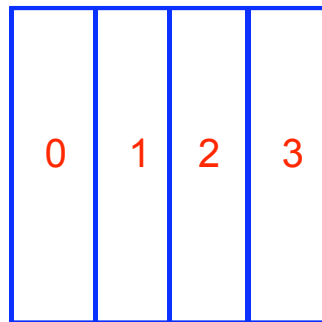


$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$



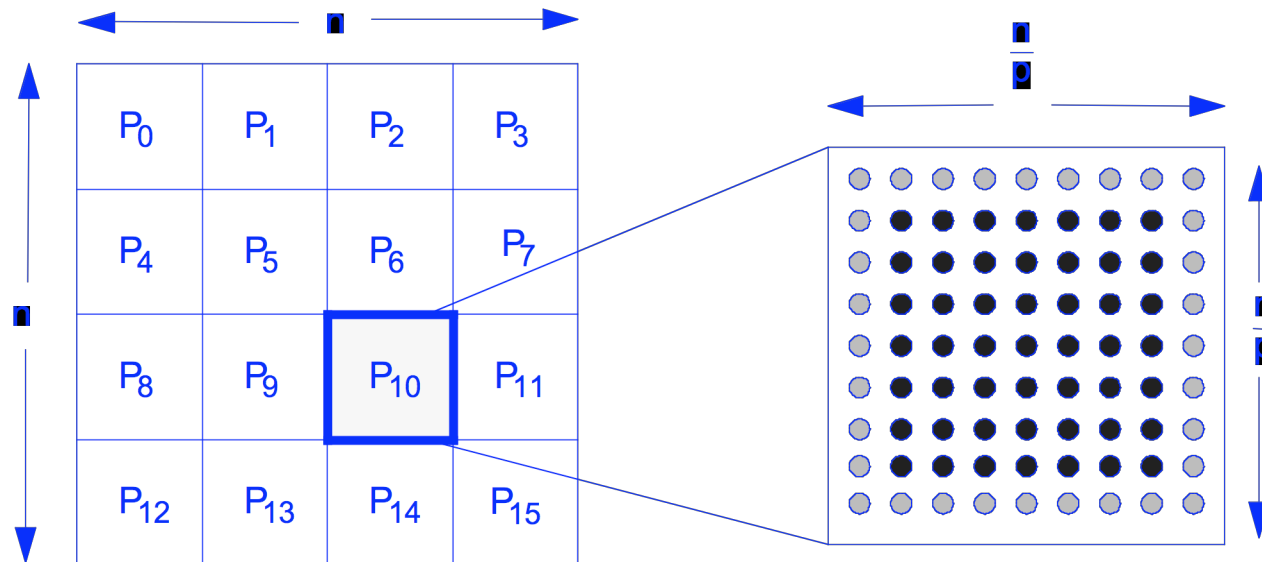
Parallel Implementation

- Partition data into parts, assigning each to a unique processor
- Dependences on values found on neighboring processes
- “Overlap” or “ghost” cells hold a copies off-processor values

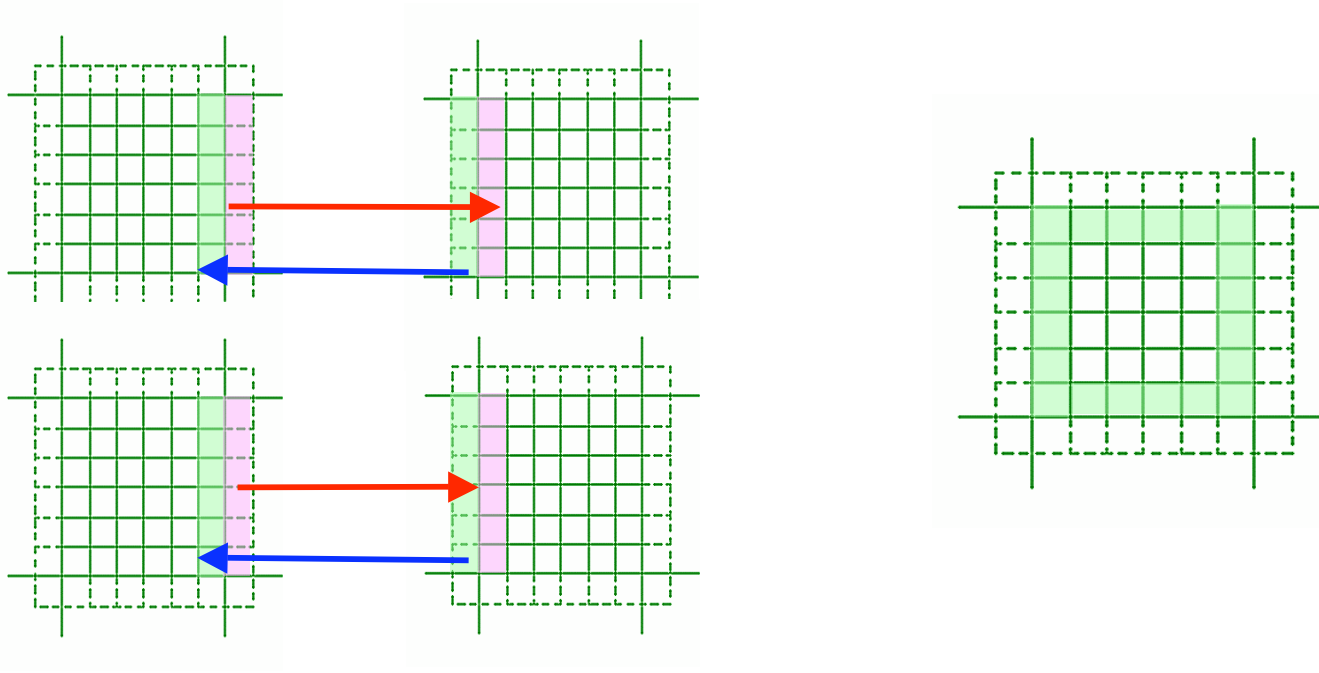


Ghost cells high dimensions

- Ghost cells surround each local subproblem
- Non-contiguous data
- Inefficient to communicate individual values

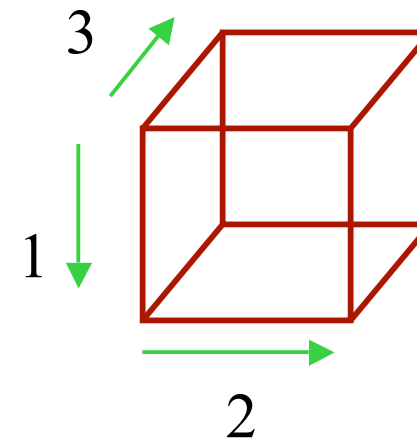


Periodically refresh the ghost cells



Performance is sensitive to processor geometry

	Total			Without communication			
Geometry	MF/s	Time	Comm	MF/s	Time		
16x 1x1	510	15.80	2.41	601	13.39		3D smoother on an IBM SP2 (16 processors)
1x16x 1	655	12.29	2.43	816	9.87		
1x 1x 16	547	14.72	8.33	1260	6.39	57% communication	
4x 4x 1	611	13.18	0.90	656	12.27	6.8% communication	
4x 1x 4	674	11.94	2.71	872	9.23		
1x 4x 4	621	12.96	2.77	790	10.19		
8x 2x 1	554	14.54	1.43	614	13.11		
8x 1x 2	619	13.00	1.21	683	11.79		
4x 2x 2	629	12.81	1.10	688	11.71		
2x 4x 2	629	12.81	1.07	686	11.74		
2x 2x 4	671	12.00	2.48	846	9.52		
2x 8x 1	669	12.03	1.39	757	10.64		
2x 1x 8	617	13.06	4.40	931	8.65		
1x 2x 8	560	14.37	4.43	810	9.95		
1x 8x 2	693	11.63	1.84	823	9.79	15.8% communication	

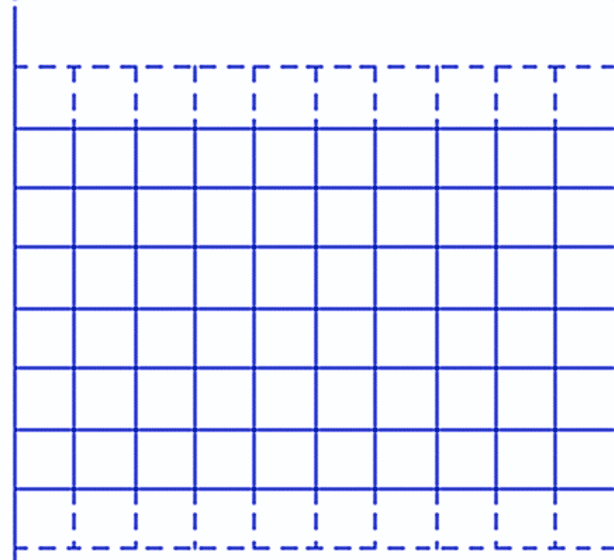
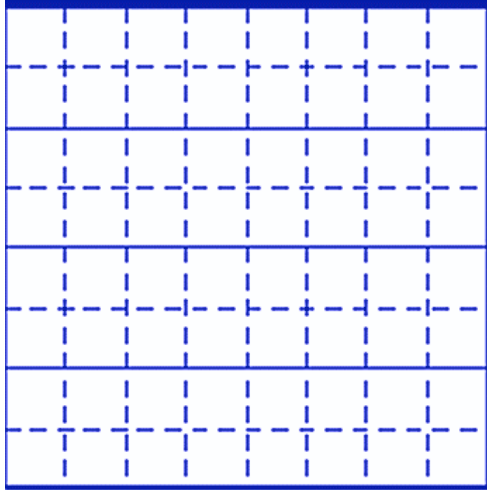


Modeling the parallel running time

- The model has two parts
 - Local computation
 - Communication: ghost cell updates
- Communication overheads are due to ghost cell updates
- Overhead and computation rate depend on the geometry

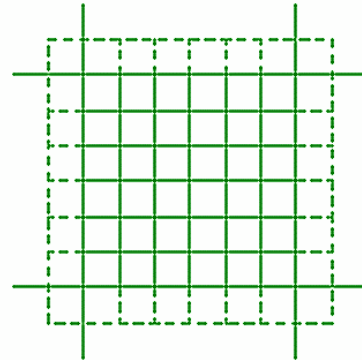
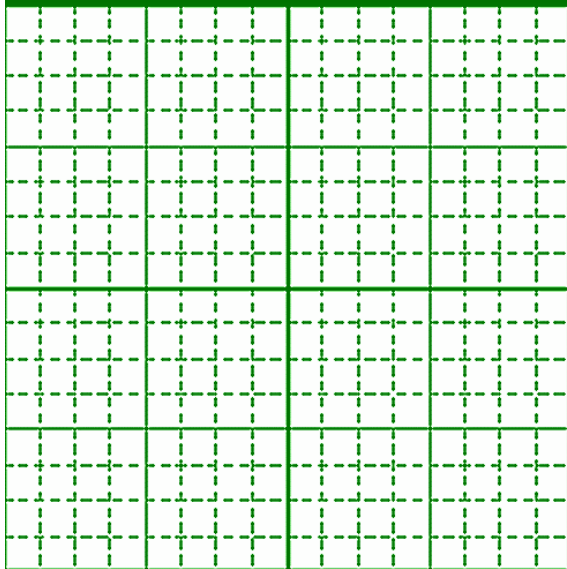
Communication cost for 1D geometry

- P divides N evenly
- $N/P > 2$
- For horizontal strips, data are contiguous
 $T_{\text{comm}} = 2(\alpha + 8\beta N)$



2D Processor geometry

- Assume $\sqrt{P}$ divides N evenly and $N/\sqrt{P} > 2$
- Ignore the cost of packing message buffers
- $T_{\text{comm}} = 4(\alpha + 8\beta N/\sqrt{P})$



Summing up the performance models

- 1-D decomposition

$$(16N^2 \beta/P) + 2(\alpha + 8\beta N)$$

- 2-D decomposition

$$(16N^2 \beta/P) + 4(\alpha + 8\beta N/\sqrt{P})$$

Comparative performance

- Strip decomposition will outperform box decomposition—resulting in lower communication times—when $2(\alpha + 8\beta N) < 4(\alpha + 8\beta N/\sqrt{P})$
- Assuming $P \geq 2$: $N < (\sqrt{P}/(\sqrt{P} - 2))(\alpha/8\beta)$
- On SDSC's IBM SP3 system “Blue Horizon”
 $\alpha = 24 \text{ us}$
 $\beta = 1/(390 \text{ MB/sec})$
- $N < 1170 (\sqrt{P}/(\sqrt{P} - 2))$
- For $P = 16$, strips are preferable when $N < 2340$,

Parallel speedup and efficiency

- 1-D decomposition

$$S_p = T_1/T_p = 16N^2\beta / (16N^2\beta/P + 2(\alpha+8\beta N))$$

$$\begin{aligned} E_p = S_p / P &= 16N^2\beta / (16N^2\beta + 2P(\alpha+8\beta N)) \\ &= 1 / (1 + (\alpha+8\beta N)P / (8N^2\beta)) \end{aligned}$$

- 2-D decomposition

$$S_p = T_1/T_p = 16N^2\beta / (16N^2\beta/P + 4(\alpha+8\beta N/\sqrt{P}))$$

$$\begin{aligned} E_p = S_p / P &= 16N^2\beta / ((16N^2\beta) + 4(\alpha P + 8\beta N\sqrt{P})) \\ &= 1 / (1 + (\alpha P + 8\beta N\sqrt{P}) / (4N^2\beta)) \end{aligned}$$

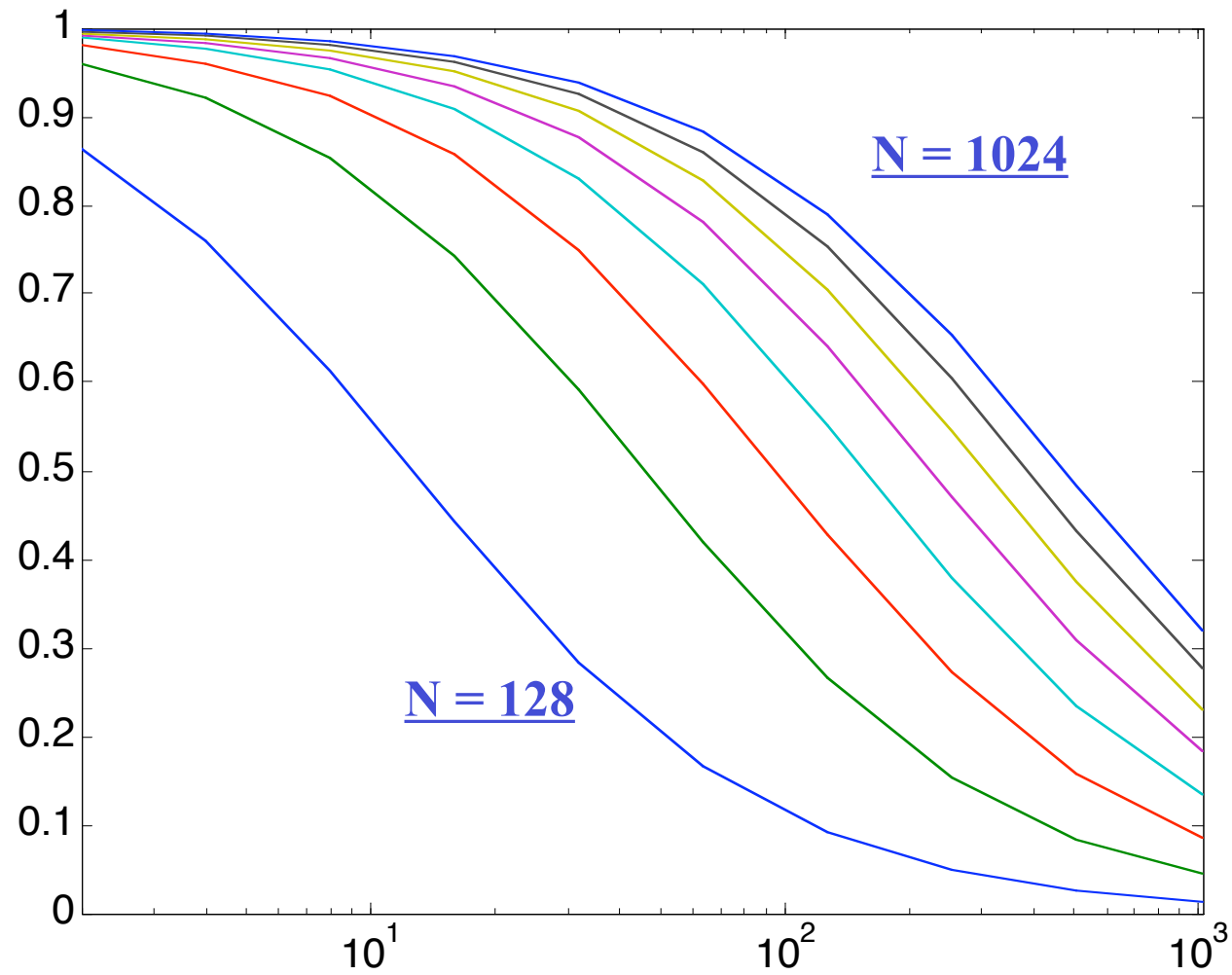
Putting these formulas to work

- 1-D decomposition
- Let's plot E_p as a function of N , varying P as a parameter

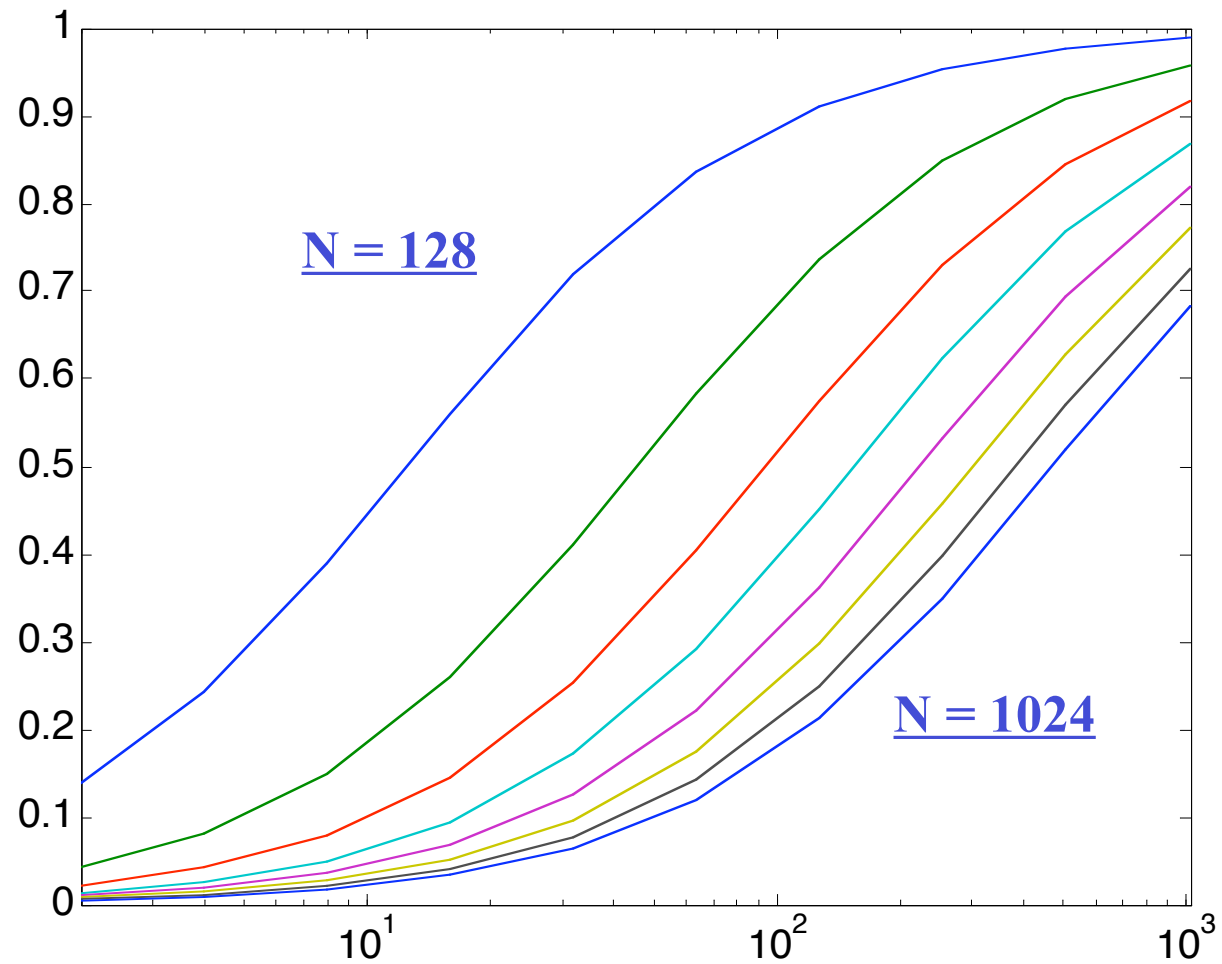
$$E_p = 1 / (1 + (\alpha + 8\beta N)P / (8N^2\beta))$$

- Let's also plot the fraction of time spent communicating

Parallel efficiency



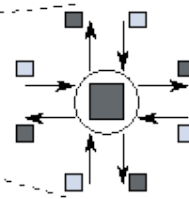
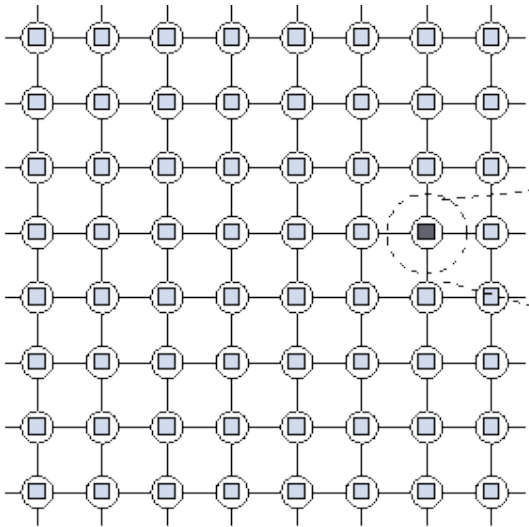
Communication fraction



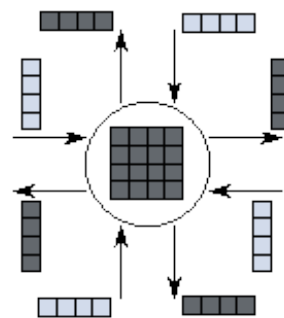
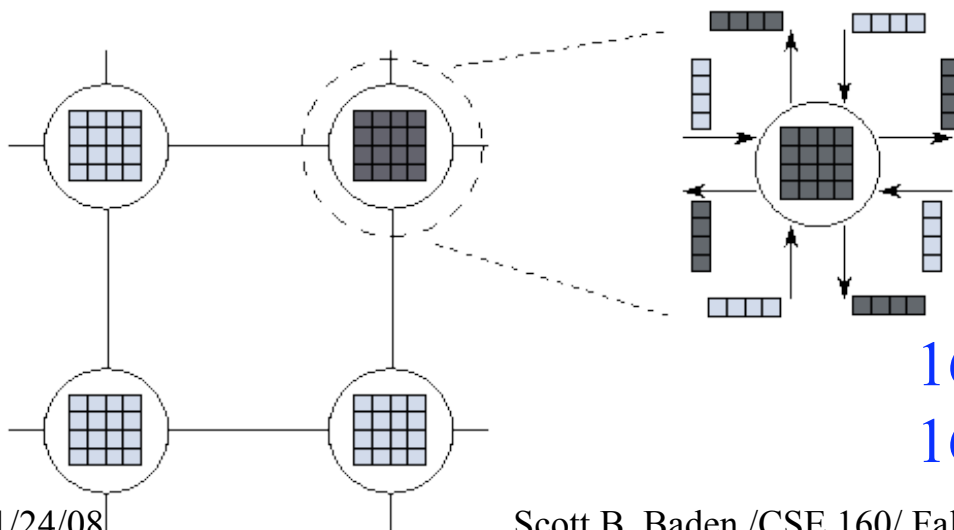
Surface to volume ratio affects performance

- The *surface to volume ratio* of a geometry is the maximum number of points on the surface (perimeter) over all partitions divided by the volume
- As we increase N while leaving P fixed, we decrease the surface to volume ratio, which gives us a measure of the relative cost of communication
- As volume increases, S/V drops

Surface to volume ratio



1 unit of work
4 units of communication



16 units of work
16 units of communication

Refinements to the performance model

- Transmitting non-contiguous data is more expensive than transmitting contiguous data, strides larger in 3D than 2D
- The grind time is sensitive to the aspect ratio of the local grid

