

CSE 262

Lecture 9

Stencil Methods

Announcements

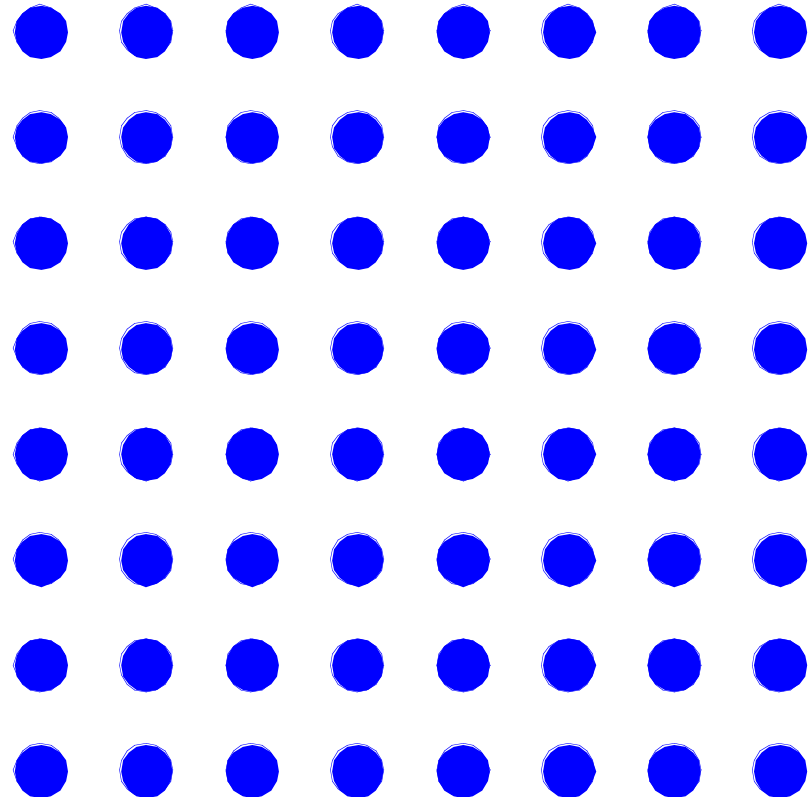
- Final presentations:
Friday March 13th, 10 AM to 12 Noon

Today's lecture

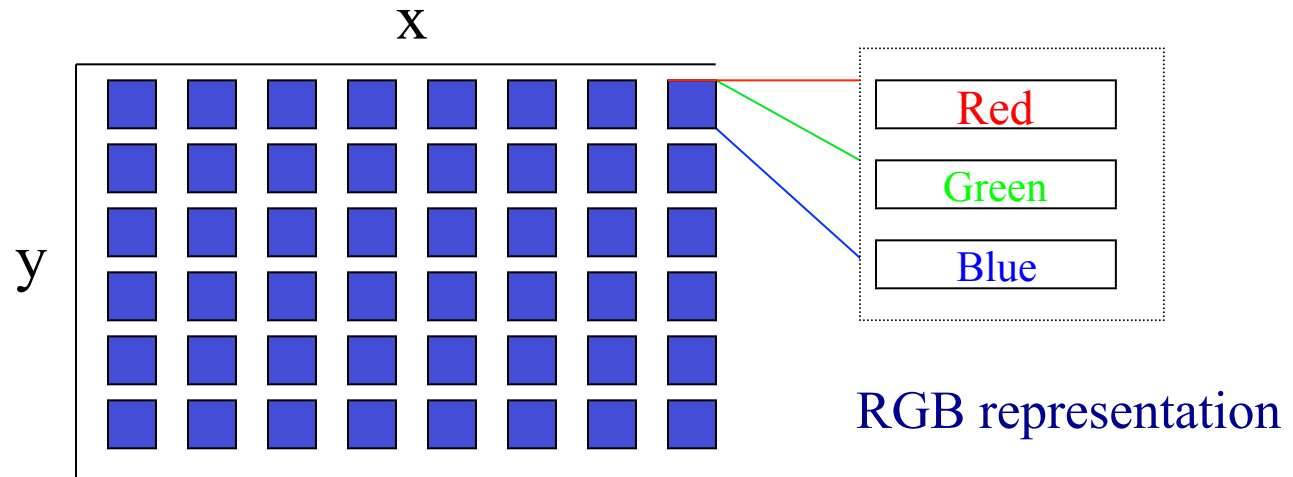
- Stencil methods
- MPI Implementation
- Under the hood of MPI
- Debugging in MPI programs
- Multigrid

Stencil methods

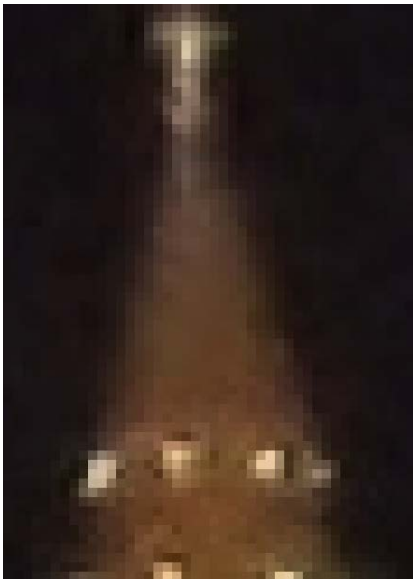
- Many physical problems are simulated on a uniform *mesh* in 1, 2 or 3 dimensions
- *Field variables* defined on a discrete set of points
- A *mapping* from ordered pairs to *physical observables* like temperature and pressure
- Important applications
 - ◆ Differential equations
 - ◆ Image processing



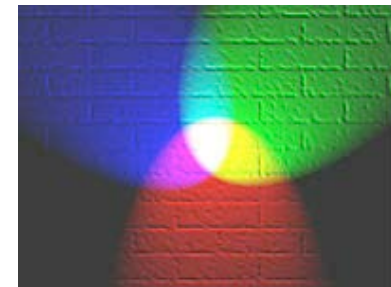
Digital Image Representation



Ryan Cuprak



Photos by Martin Juell



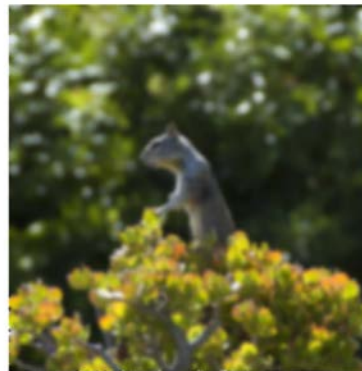
wikipedia

Canonical stencil method: image smoothing

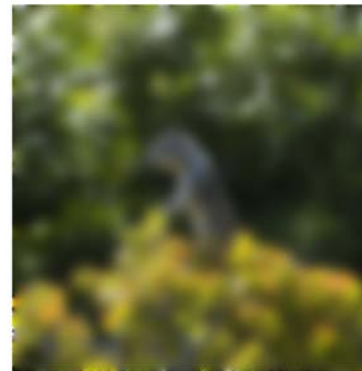
```
for iter = 1 : nSmooth
  for (i,j) in 0:N-1 x 0:N-1
     $\text{Img}^{\text{new}}[i,j] = (\text{Img}[i-1,j] + \text{Img}[i+1,j] + \text{Img}[i,j-1] + \text{Img}[i,j+1]) / 4$ 
  end
   $\text{Img} = \text{Img}^{\text{new}}$ 
end
```



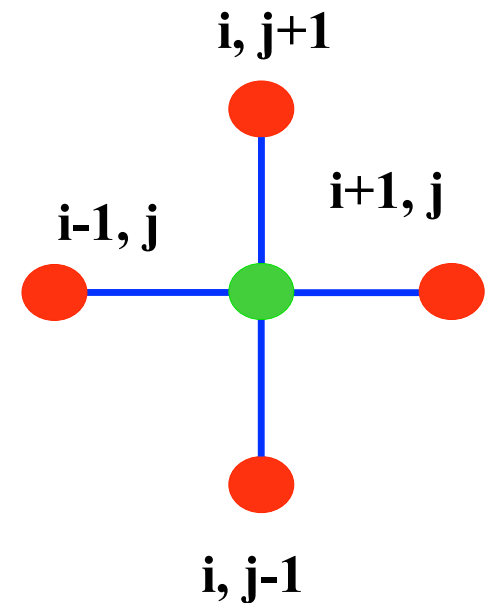
Original



100 iter

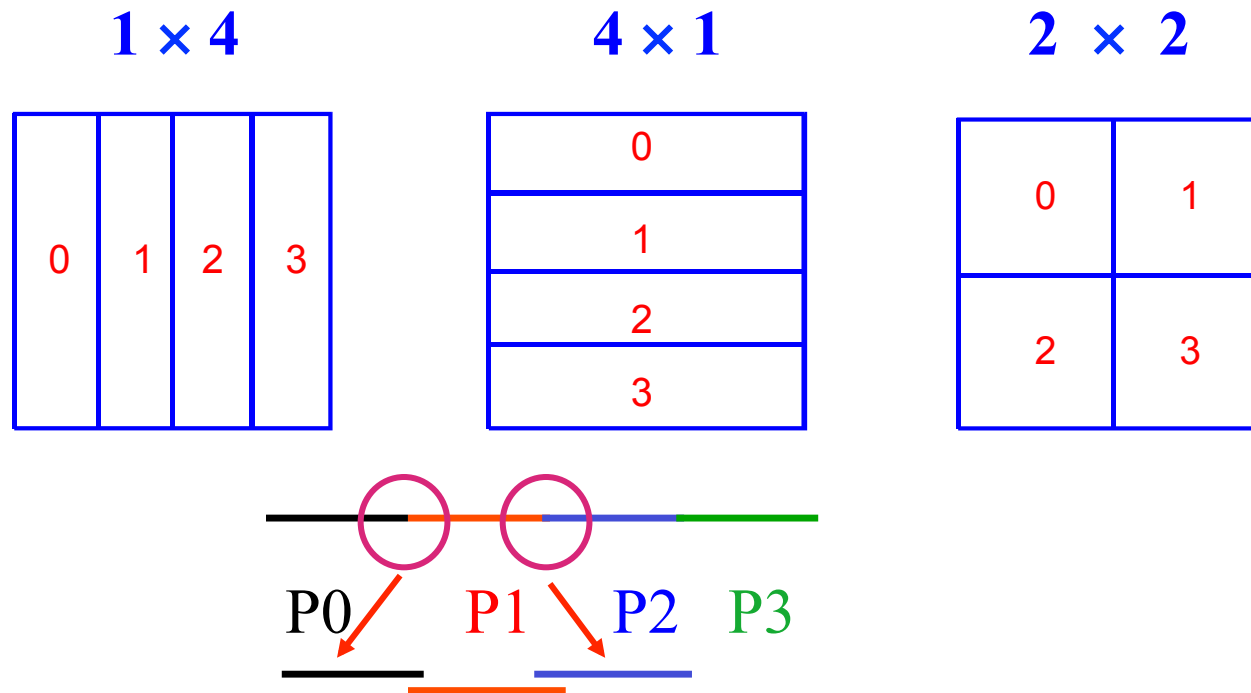


1000 iter



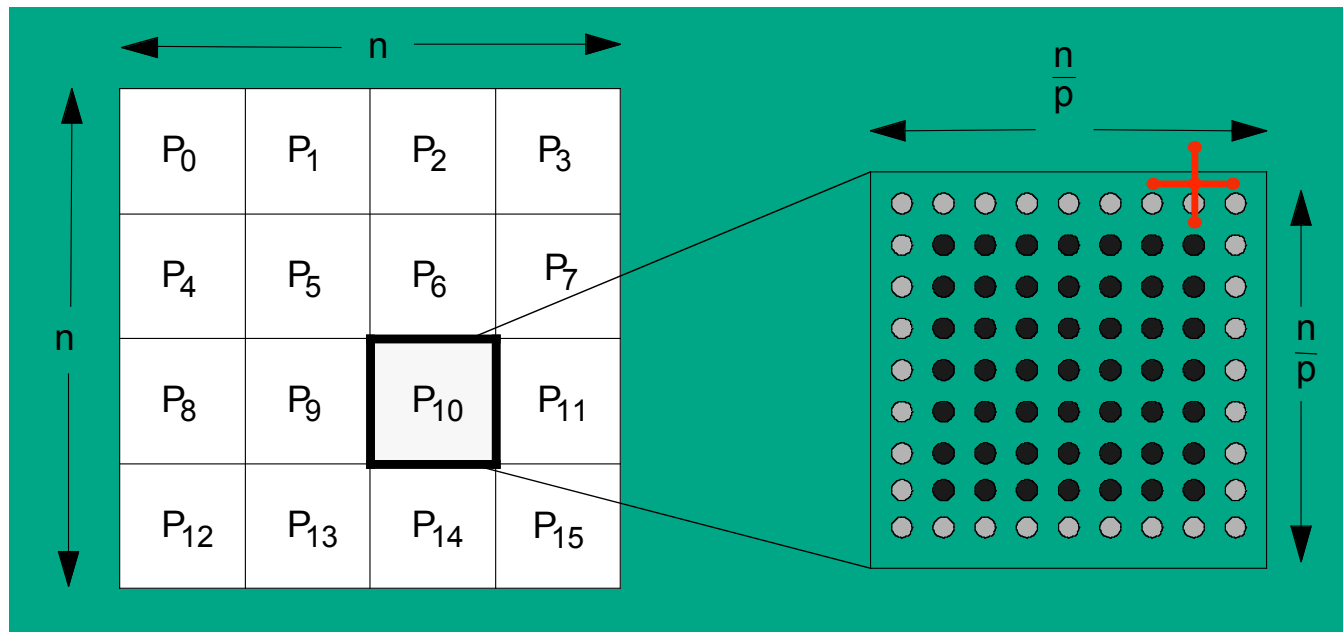
Stencil methods under message passing

- Partition computation and data, assigning each partition to a unique process: “Owner computes rule”
- Different partitionings according to the *processor geometry*
- Dependences on values found on neighboring processes
- Communicate off-processor data



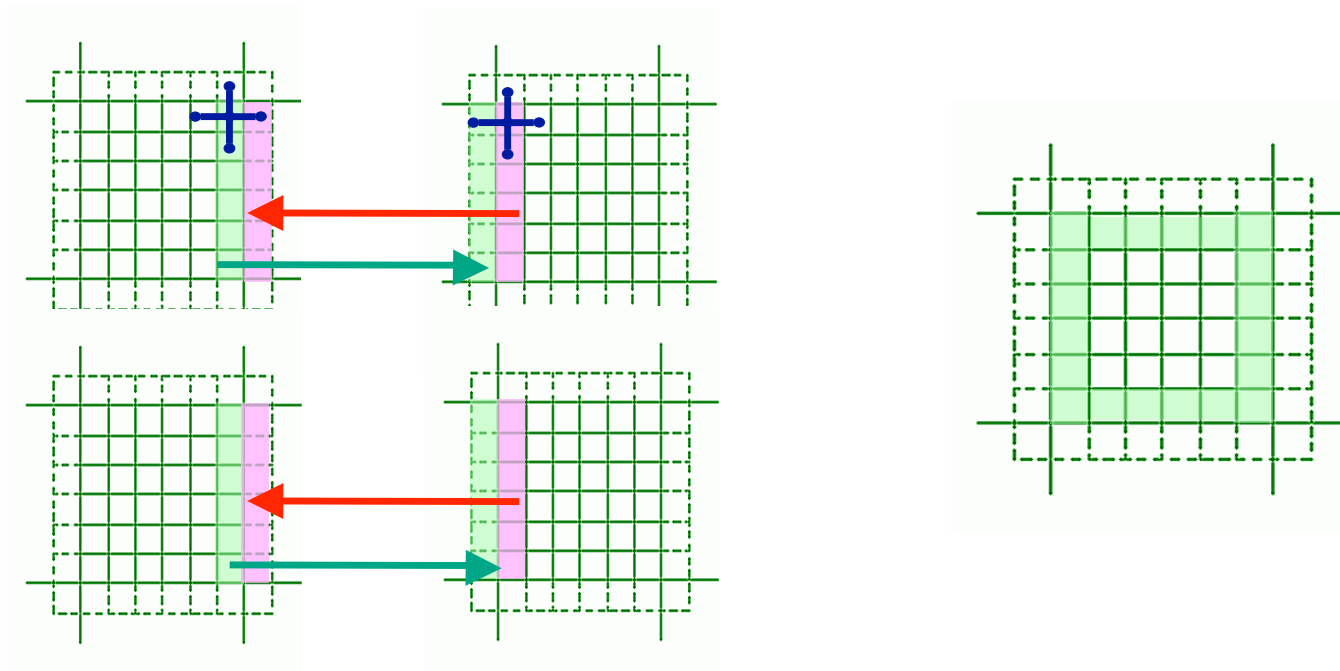
Communication

- Expensive to communicate each mesh box individually
- Send/Recv data *en masse* into *ghost regions* on neighboring processors
- Packing and unpacking of non-contiguous data on some boundaries



Managing ghost cells

- Post **IReceive** () for all neighbors
- **Send** data to neighbors
- **Wait** for completion



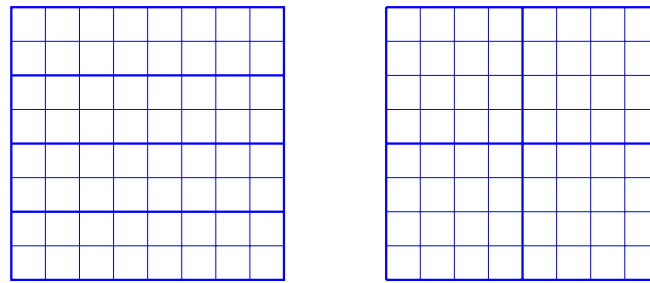
Performance is sensitive to processor geometry

- Aliev- Panfilov method running on triton.sdsc.edu (Nehalem Cluster)
- 256 cores, n=2047, t=10 (8932 iterations)

Geometry	GFlops	Gflops w/o Communication
32 x 8	573	660
8 x 32	572	662
16 x 16	554	665
2 x 128	508	658
4 x 64	503	668
128 x 2	448	658
256 x 1	401	638

Modeling the parallel running time

- The model has two parts
 - ♦ Local computation
 - ♦ Communication
- We may ignore the convergence test (check infrequently)
- Communication overheads are due to ghost cell updates
- 2 kinds of geometries: strips vs boxes



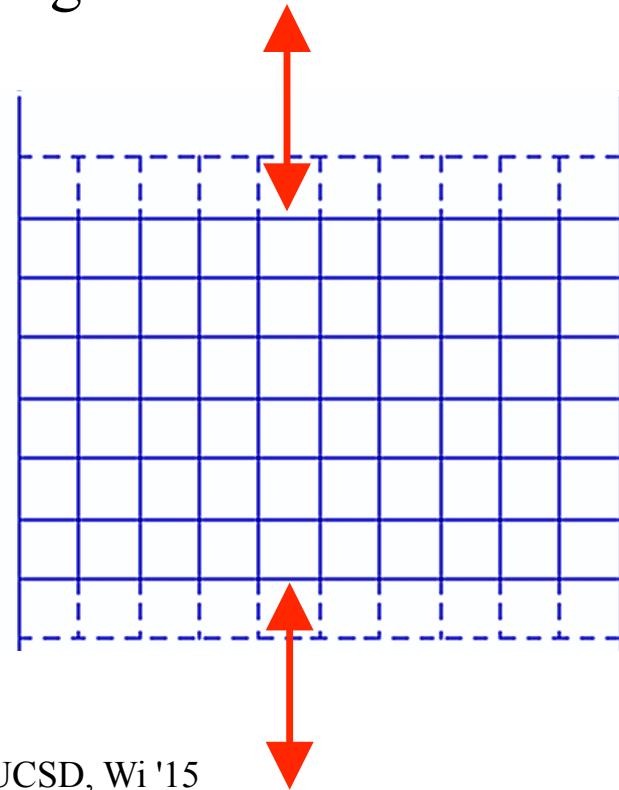
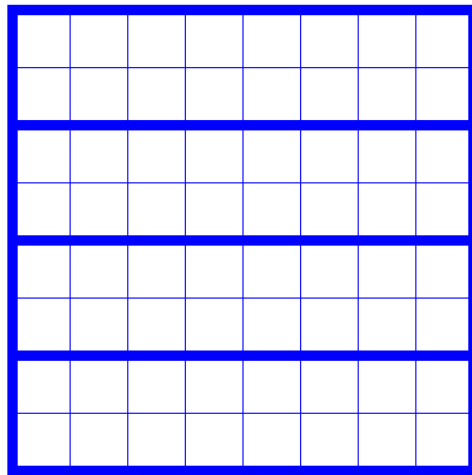
Model assumptions and definitions

- $T(1,(m,n))$ = running time of the **best serial algorithm** on a problem of size $m \times n$
- $T(P,(m,n))$ = running time on P processors
- $T_\gamma(P,(m,n))$ = **grind time** on P processors
 - ♦ $T_\gamma(P,(m,n)) = T(P,(m,n))/(m \cdot n \cdot \text{Niter})$
 - ♦ Ideally T_γ is independent of m , n , and P
- Processor geometry is $p \times q$
 - ♦ Strips or box –like partitions
- $T(P,(N,N)) = T(1,(m,n)) + T_{\text{comm}}, m = N/p, n = N/q$
- $T_{\text{comm}} = \alpha + \beta^{-1}_\infty n$ [oversimplified!]
- Following analysis applies to a Poisson Solver

$$u'[i,j] = (u[i-1, j] + u[i+1, j] + u[i, j-1] + u[i, j+1] - h^2 * b[i-1, j-1]) / 4$$

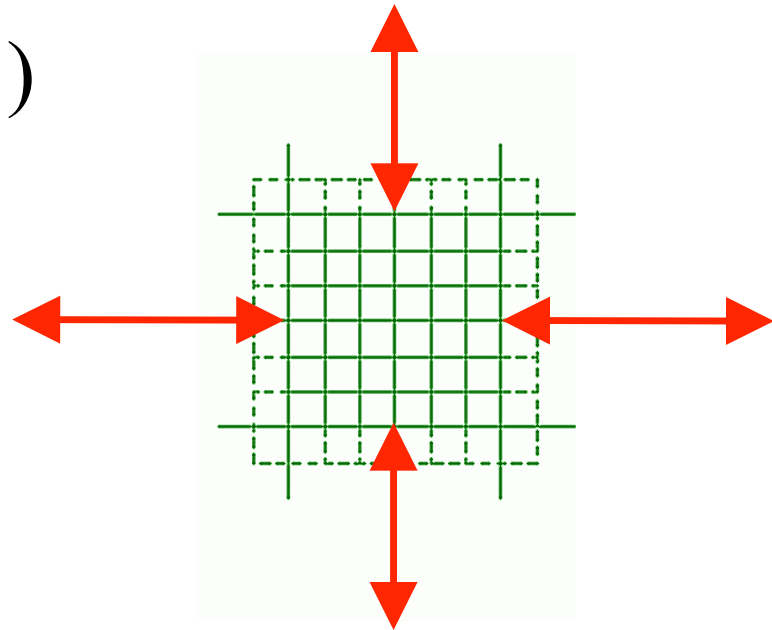
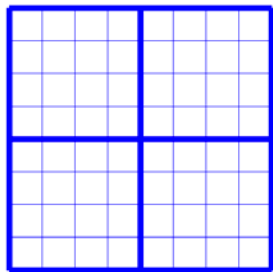
Communication costs for 1D geometries

- Assumptions
 - ♦ P divides N evenly
 - ♦ $N/P > 2$
 - ♦ 1 word = double precision floating point = 8 bytes
- For horizontal strips, data are contiguous
 $T_{\text{comm}} = 2(\alpha + 8\beta N)$



2D Processor geometry

- Assumptions
 - ♦ $\sqrt{P}$ divides N evenly
 - ♦ $N/\sqrt{P} > 2$
 - ♦ 1 word = double precision floating pt. = 8 bytes
- Ignore the cost of packing message buffers
- $T_{\text{comm}} = 4(\alpha + 8\beta N/\sqrt{P})$



Summing up communication costs

- Substituting $T_\gamma \approx 16 \beta$
- 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 Bang
 - $\alpha = 1.2 \text{ us}, \beta = 1/(1.4 \text{ GB/sec}),$
 - ♦ $N < 210(\sqrt{P}/(\sqrt{P} - 2))$
 - ♦ For $P = 16$, strips are preferable when $N < 280$
- 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

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

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

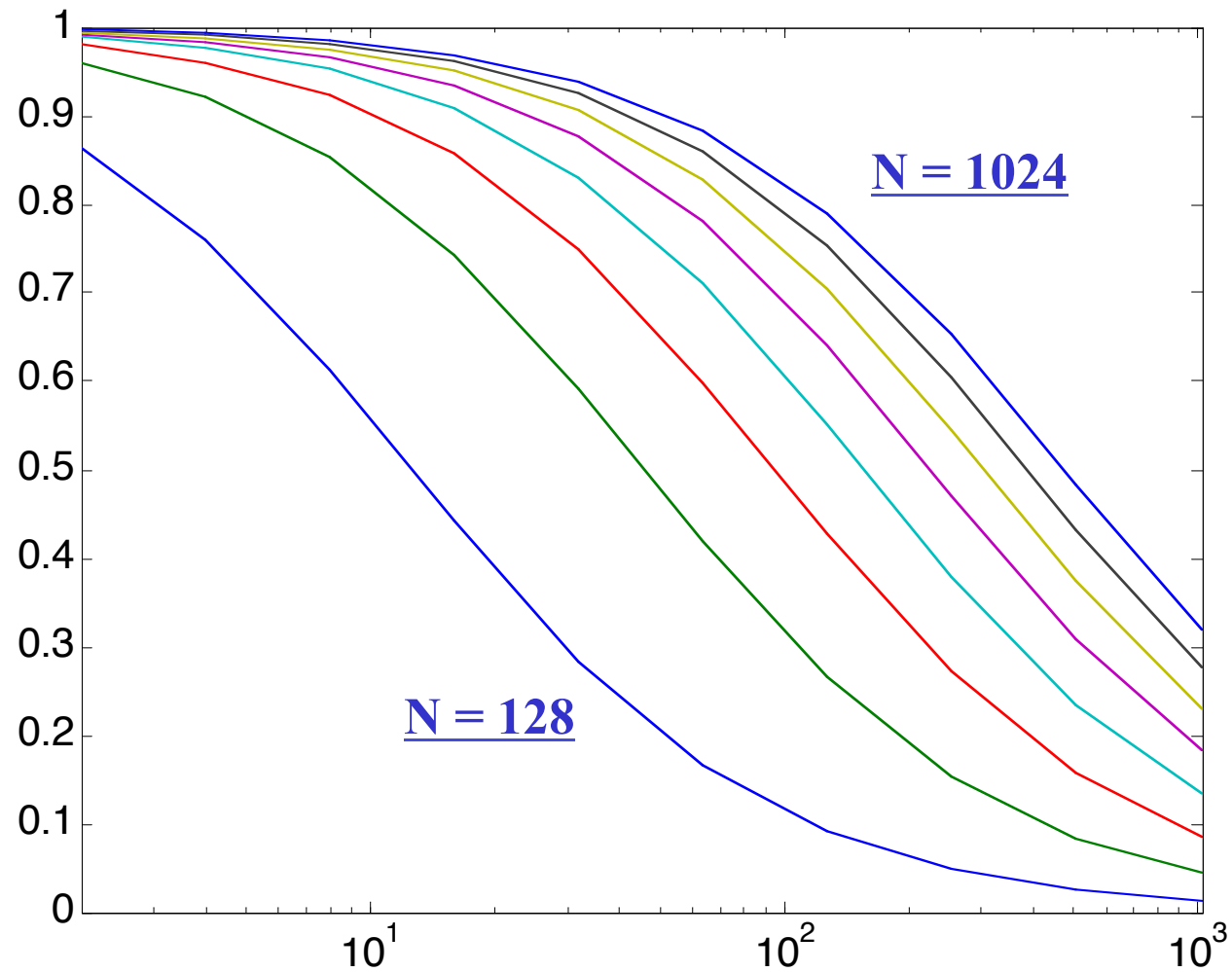
Putting these formulas to work

- 1-D decomposition
- Plot E_p as a function of N , varying P as a parameter

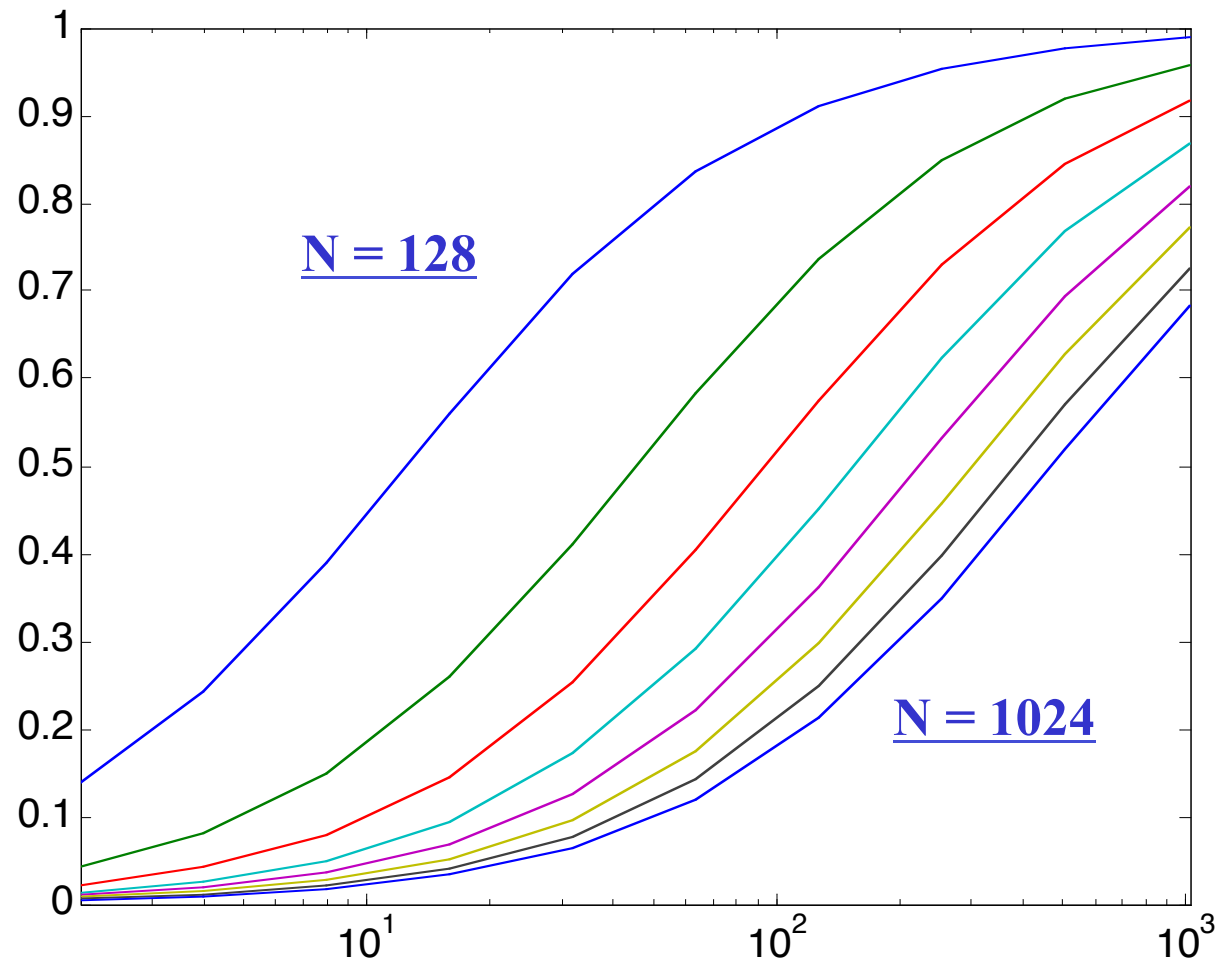
$$E_p = 1 / (1 + (\alpha + 8\beta N)P / (8N^2\beta))$$

- Plot the fraction of time spent communicating

Parallel efficiency



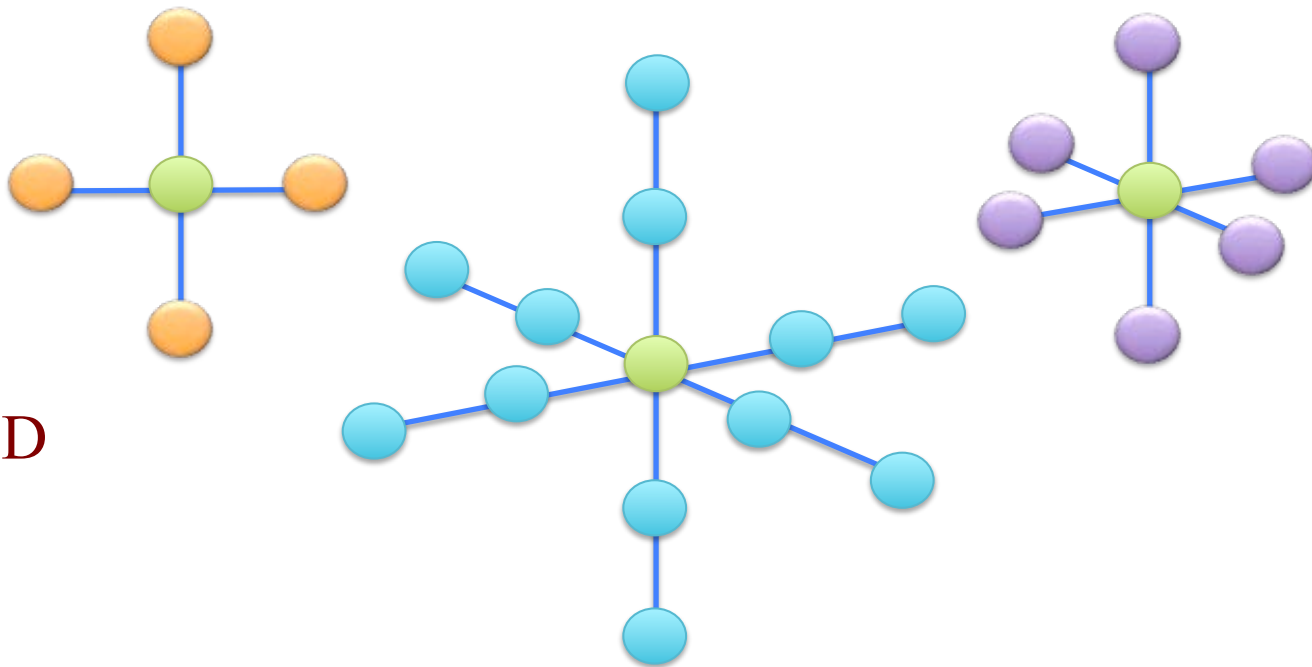
Communication fraction



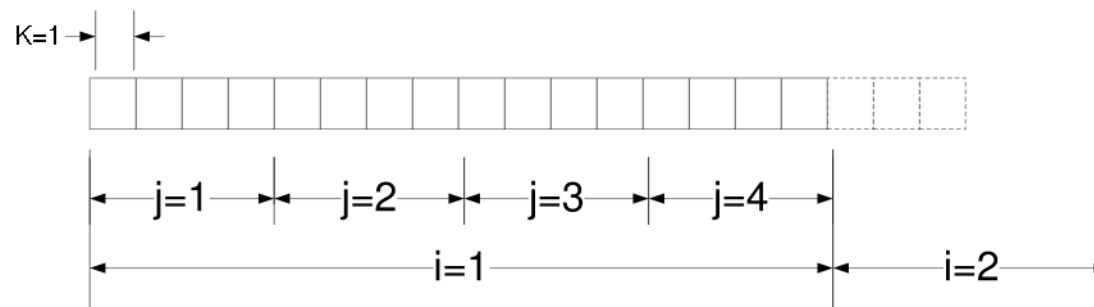
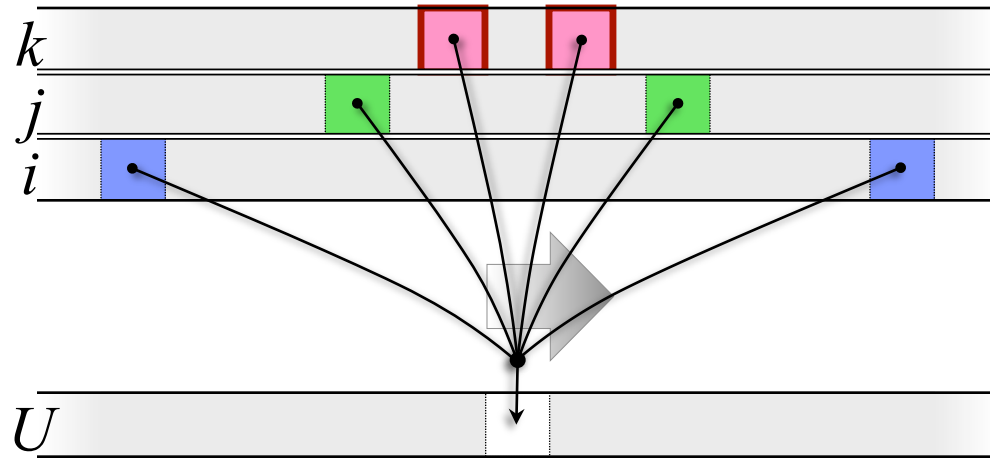
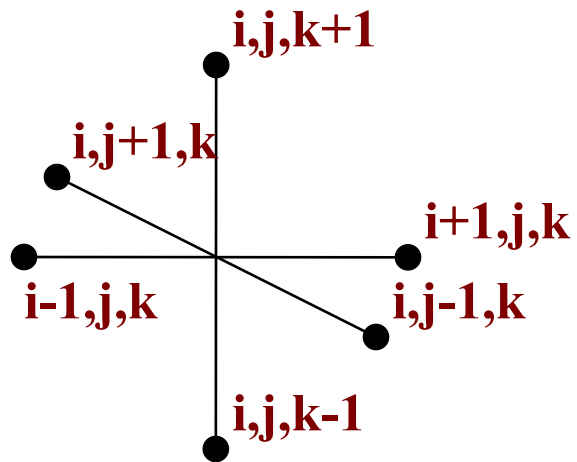
3D Stencils

- More demanding
 - ♦ Large strides
 - ♦ Curse of dimensionality

2D



Memory Strides



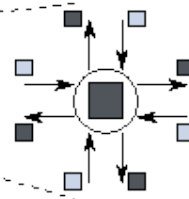
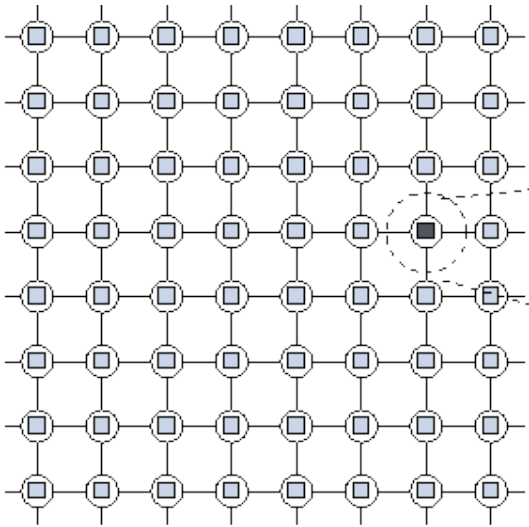
Sam Williams et al.

H. Das, S. Pan, L. Chen

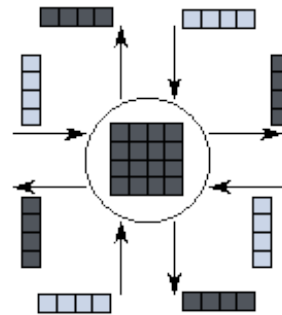
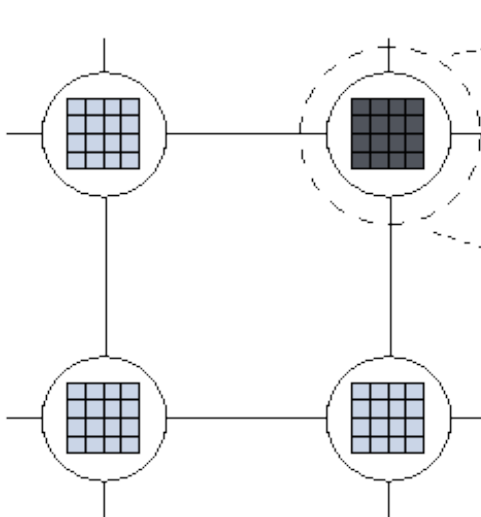
The curse of dimensionality

- As we move to higher dimensional spaces, communication becomes relatively more costly
 - ▶ In 2D: $4N / N^2 = 4/N$
 - ▶ In 3D: $6N^2 / N^3 = 6/N$
- A surface to volume effect
 - ◆ Communication is proportional to surface area of computation is proportional to the volume
 - ◆ Increasing N/P decreases the surface volume ratio, lowering relative communication cost

Surface to volume ratio



1 unit of work
4 units of communication



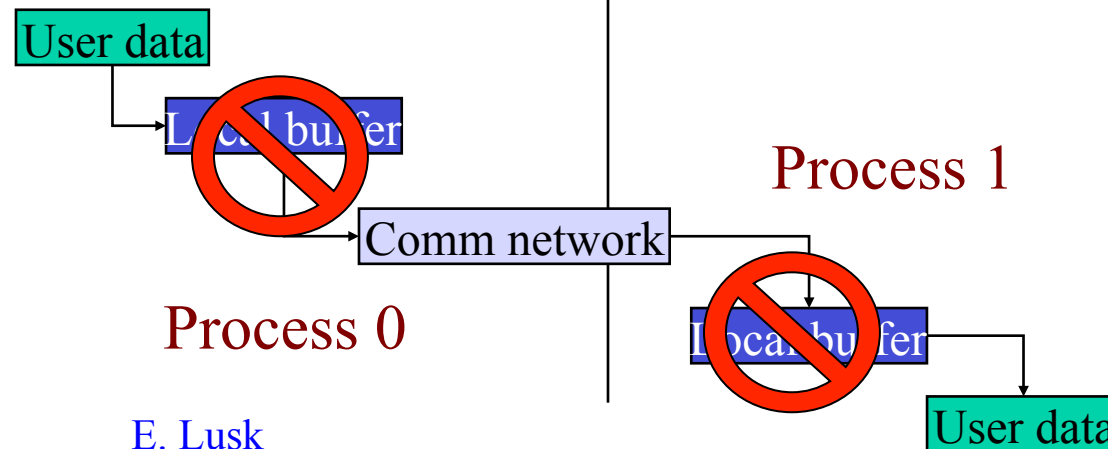
16 units of work
16 units of communication

Today's lecture

- Stencil methods
- MPI Implementation
- **Under the hood of MPI**
- Debugging in MPI programs
- Multigrid

Buffering

- Where does the message go when you send it?
- If there's not a pending receive for an incoming message, it's placed in an anonymous system buffer
- When the receive gets posted, the message is moved into the user specified buffer
- Double copying reduces communication performance
- Non-blocking communication can help avoid it



E. Lusk

Rendezvous

- When a long message is to be sent, can MPI just send the message?
- For “short” message, it can. This is *eager mode*
- *Eager limit*: longest message that can be sent in eager mode
- See M. Banikazemi et al., IEEE TPDS, 2001, “MPI-LAPI: An Efficient Implementation of MPI for IBM RS/6000 SP Systems”
- For long messages, MPI first sends a scout to get permission to send the message
- This is called *rendezvous* mode

Today's lecture

- Stencil methods
- MPI Implementation
- Under the hood of MPI
- **Debugging in MPI programs**
- Multigrid

Debugging tips

- Bugs?! Not in my code!
- The seg fault went away when I added a print statement
- Garbled output
- 2D partitioning is much more involved than 1D
- MPI is a library, not a language
- Gdb not as useful when you have many processes
- Try ddt on Stampede
<https://portal.tacc.utexas.edu/software/ddt>
- Parallel print function

Parallel print function

- Problem: how to sort out all the output on the screen
- Many messages say the same thing
 - Process 0 is alive!
 - Process 1 is alive!
 - ...
 - Process 15 is alive!
- Compare with
 - Processes[0–15] are alive!
- Parallel print facility
 - <http://www.llnl.gov/CASC/ppf>

Summary of capabilities

- Compact format list sets of nodes with common output
`PPF_Print( MPI_COMM_WORLD, "Hello world" );`
0-3: Hello world
- `%N` specifier generates process ID information
`PPF_Print( MPI_COMM_WORLD, "Message from %N\n" );`
Message from 0-3
- Lists of nodes
`PPF_Print(MPI_COMM_WORLD,`
`(myrank % 2)`
`? "[%N] Hello from the odd numbered nodes!\n"`
`: "[%N] Hello from the even numbered nodes!\n")`
[0,2] Hello from the even numbered nodes!
[1,3] Hello from the odd numbered nodes!

Practical matters

- Installed in `$(PUB)/lib/PPF`
- Specify `ppf=1` and `mpi=1` on the “make” line or in the Makefile
 - ◆ Defined in `arch.gnu-4.7_c++11.generic`
 - ◆ Each module that uses the facility must

`#include “ptools_ppf_cpp.h”`

- Look in `$(PUB)/Examples/MPI/PPF` for example programs `ppfexample_cpp.C` and `test_print.c`
- Uses `MPI_Gather()`
- Installed on Bang, will install on Stampede upon request

Today's lecture

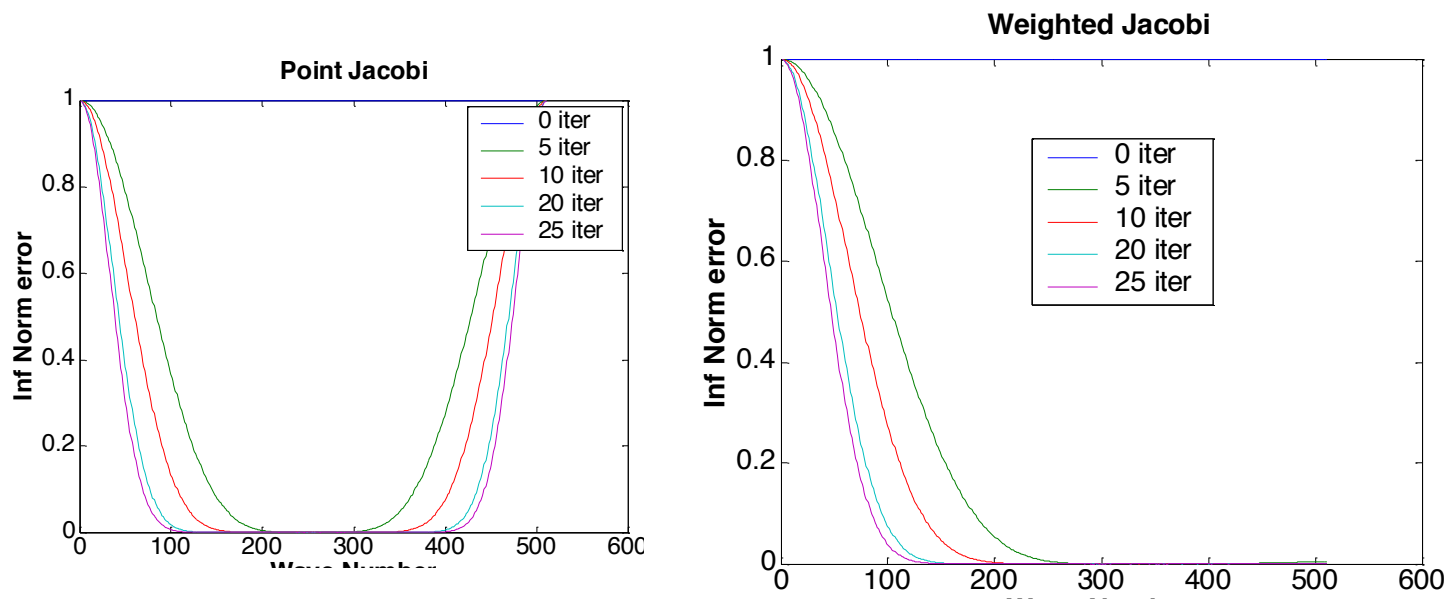
- Stencil methods
- MPI Implementation
- Under the hood of MPI
- Debugging in MPI programs
- **Multigrid**

Multigrid

- The image smoother converges slowly: $O(n^2)$ iterations for a mesh with n unknowns, $n=m^2$
- Slow numerical communication, by nearest neighbors
- $O(1)$ convergence: multigrid
- Multi-scale
 - ▶ Coarsen the mesh, solve a smaller problem in less time
 - ▶ Do this recursively

The effects of smoothing on the error

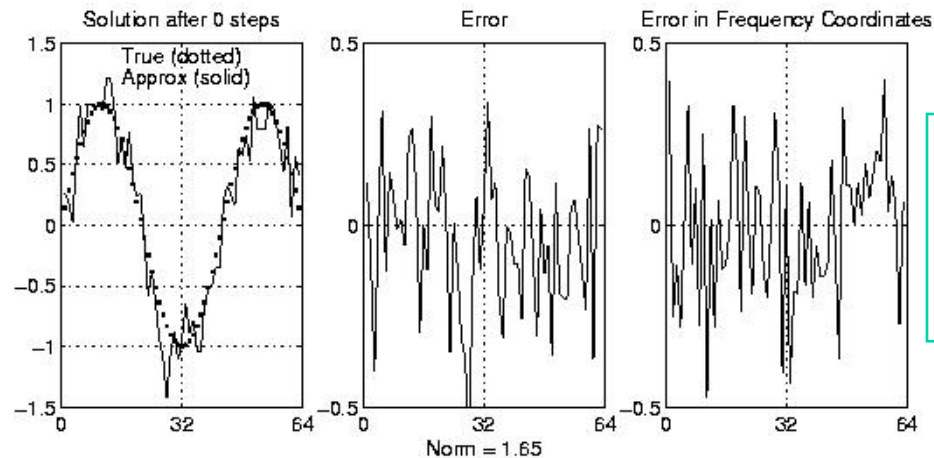
- Let's express the error in the solution in frequency space
- The smoother suppresses the higher frequency components in just a few steps



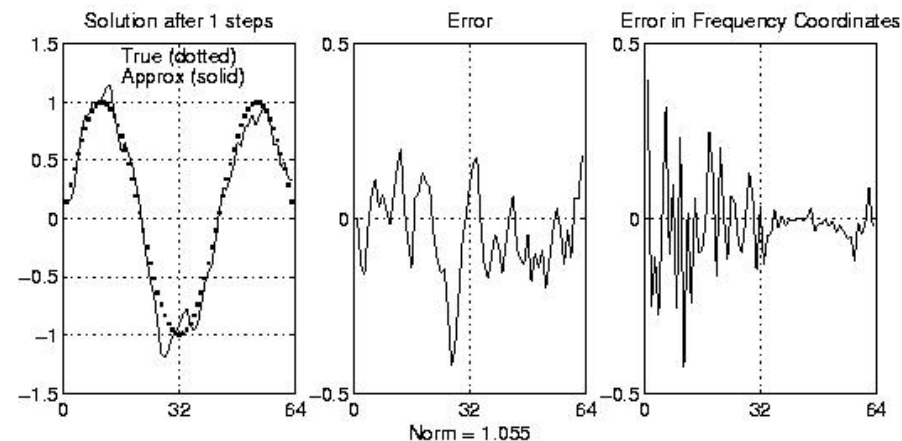
The idea behind multigrid

- If we can make low frequencies appear to be high frequencies, we can speed convergence
- Coarsening the mesh uses half as many points, doubling the frequency
- Cancel out the fine mesh errors using coarse mesh information
- Multigrid provides the glue between levels
- Another angle: numerical information communicated at multiple length scales

Smoothing damps high frequency error



Initial error
"Rough"
Lots of high frequency components
Norm = 1.65



Error after 1 Jacobi step
"Smoother"
Fewer high frequency components
Norm = 1.055