

Final Exam Review

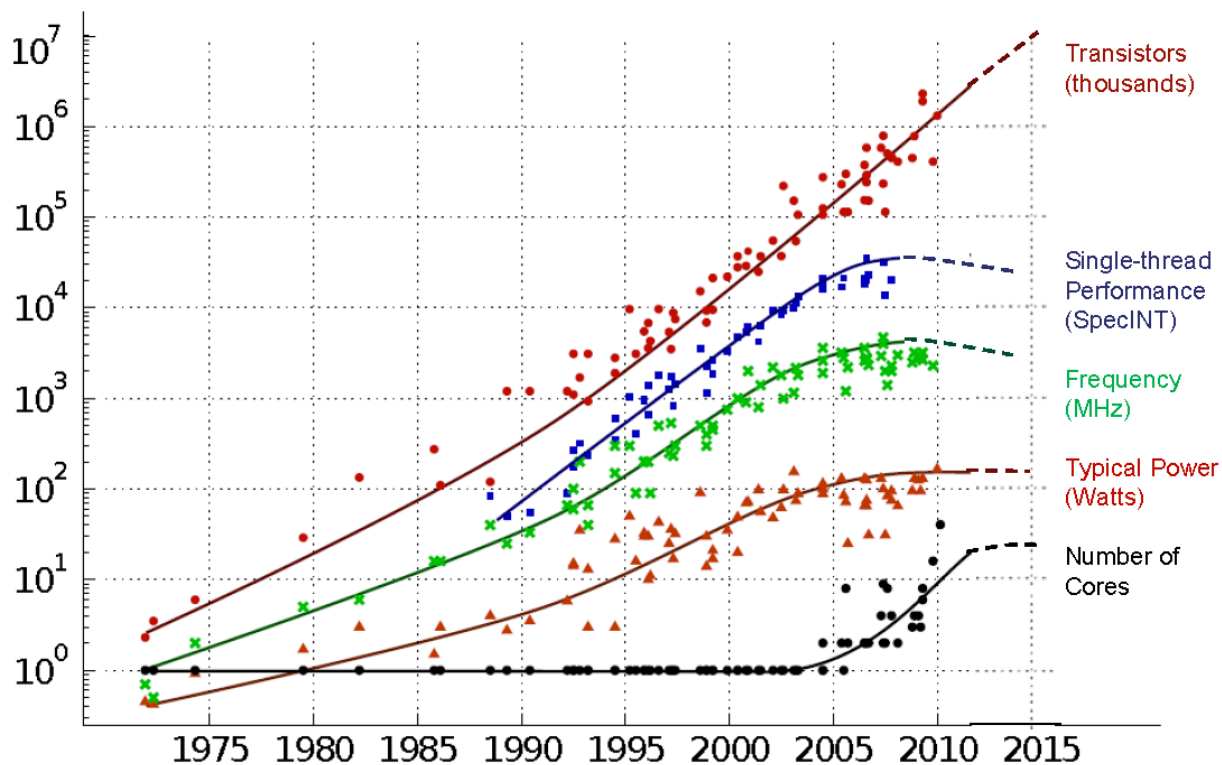
Announcements

- Final examination
 - Friday, March 22, 11.30a to 2.30p, PETER 104
 - You may bring a single sheet of notebook sized paper “8x10 inches” with notes
- Office hours during examination week
 - Monday and Wednesday usual time (2p to 3p)
 - Or by appointment

Today's lecture

- Technology trends
- Address space organization
- Interconnect
- Algorithms (applications)
- Parallel program design and implementation
- Programming models
- Performance

35 years of processor trends



Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten
Dotted line extrapolations by C. Moore

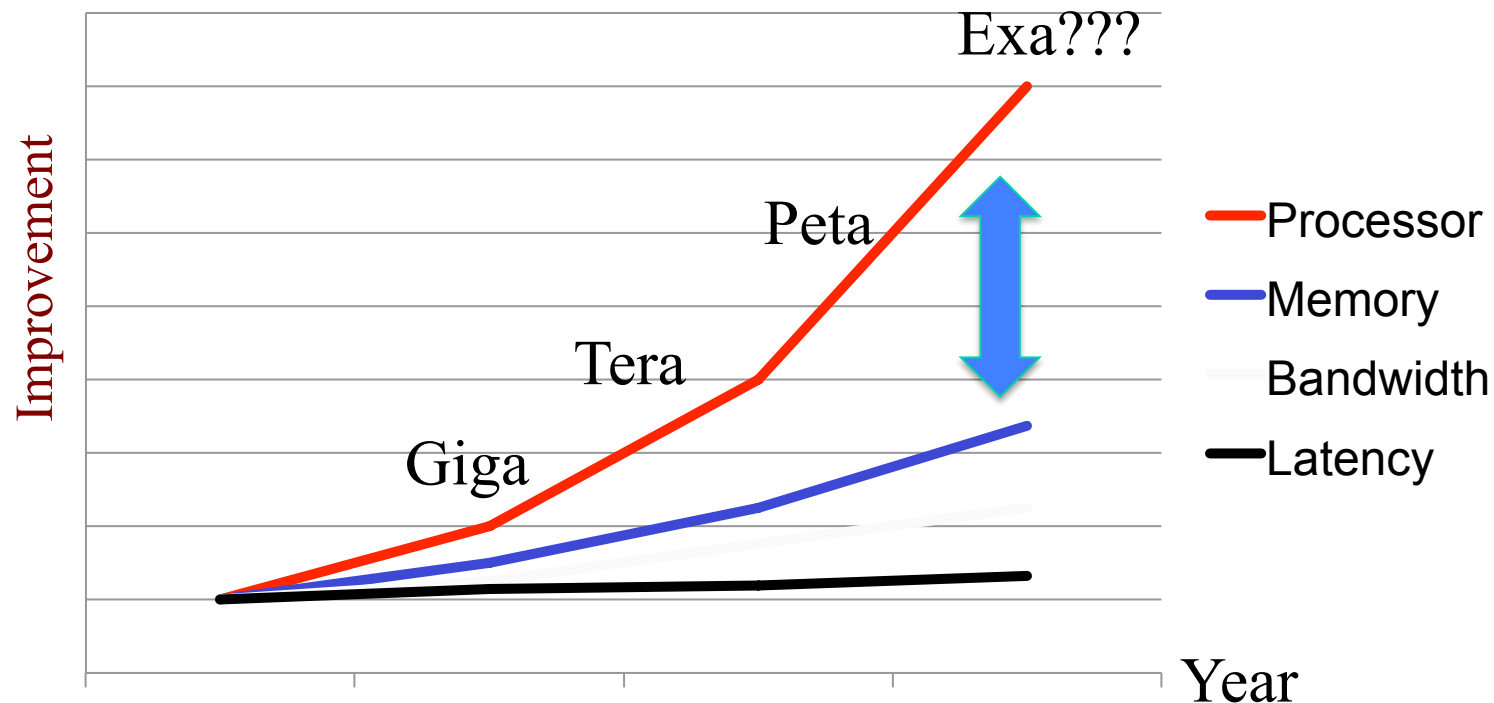
3

Data Processing in Exascale-class Computing Systems | April 27, 2011 | CRM



Trend: data motion costs are rising

- Increase amount of computation performed per unit of communication
 - Conserve locality, tolerate or avoid communication
- Many threads

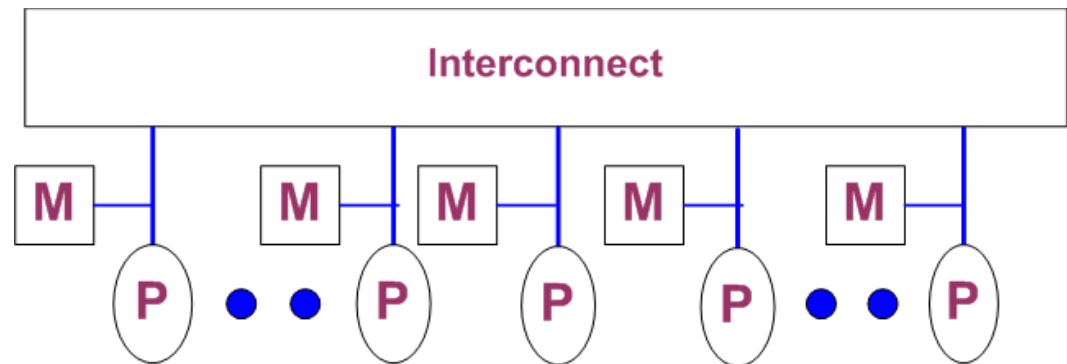
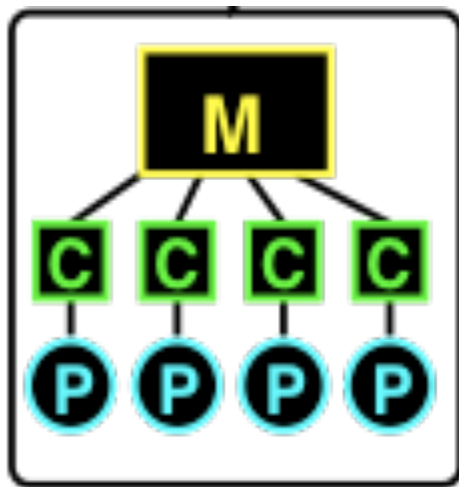


Today's lecture

- Technology trends
- Address space organization
- Interconnect
- Algorithms (applications)
- Parallel program design and implementation
- Programming models
- Performance

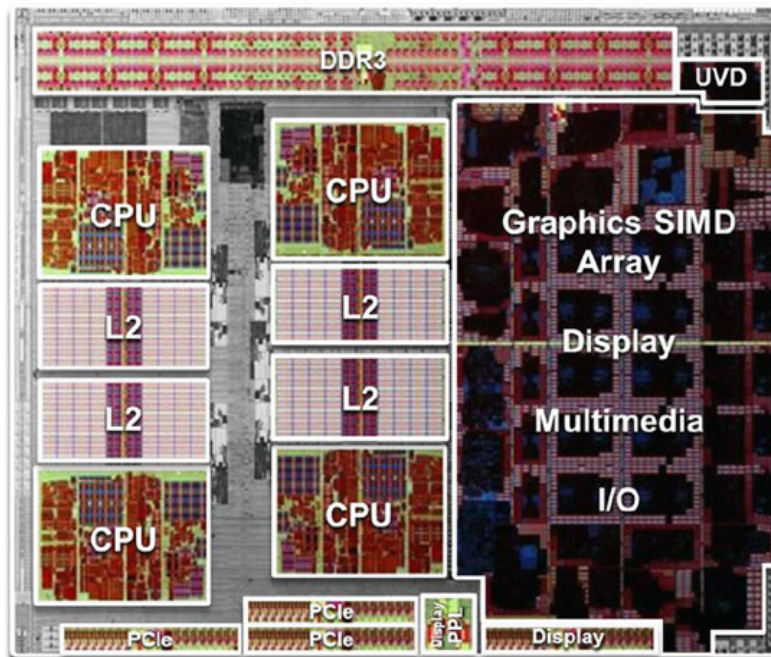
Address Organization

- Multiprocessors and multicomputers
- Shared memory, message passing



Hybrid processing

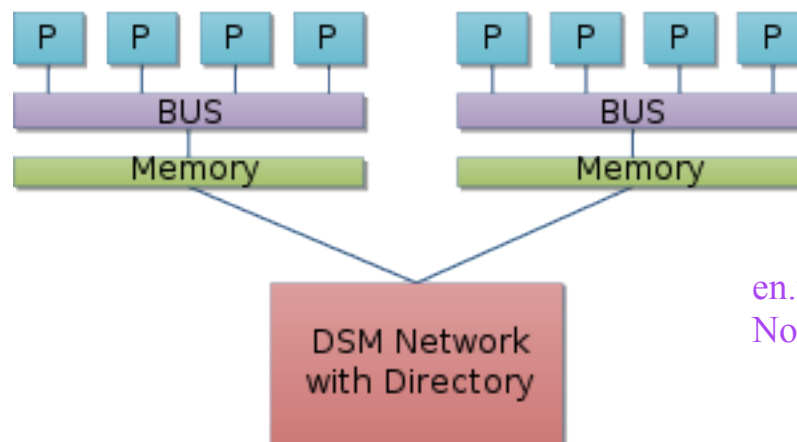
- Two types of processors: general purpose + accelerator
 - AMD fusion: 4 x86 cores + hundreds of Radeon GPU cores
- Accelerator can perform certain tasks more quickly than the conventional cores
- Accelerator amplifies relative cost of communication



Hothardware.com

NUMA Architectures

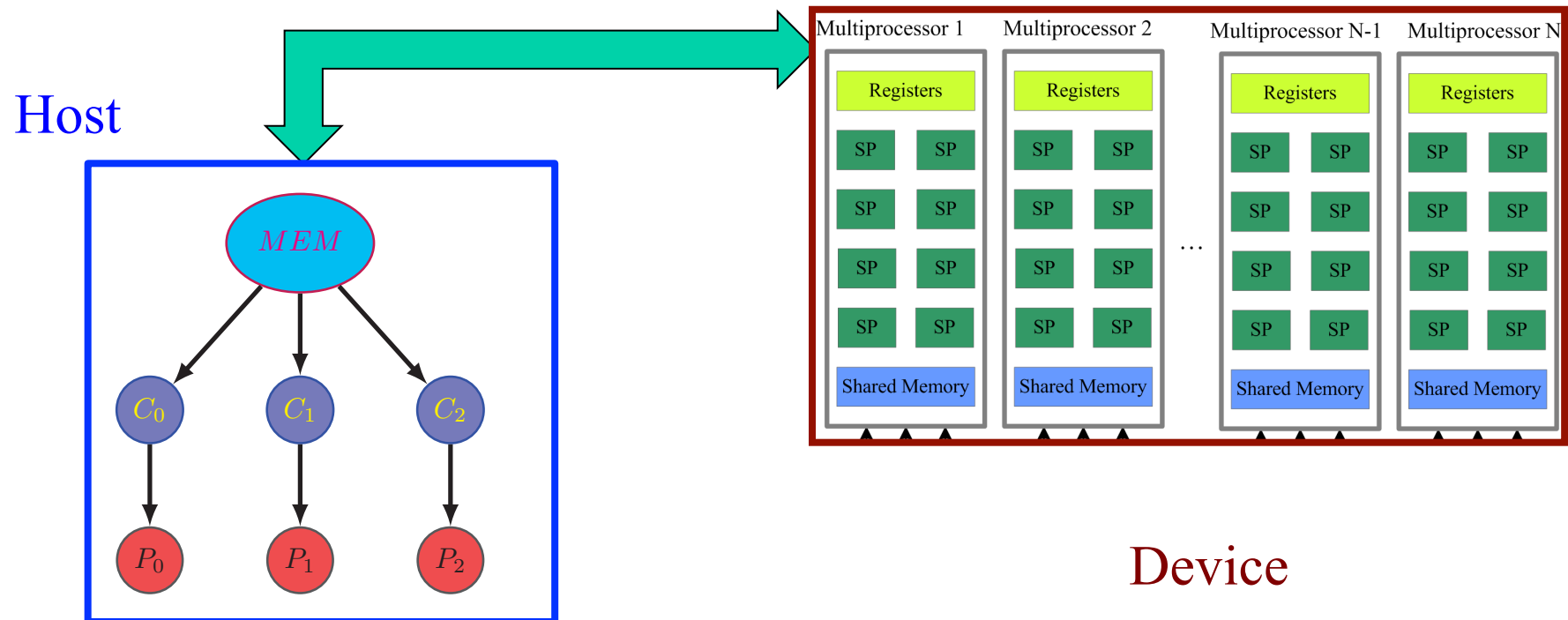
- The address space is global to all processors, but memory is physically distributed
- Point-to-point messages manage coherence
- A **directory** keeps track of sharers, one for each block of memory
- Stanford Dash; NUMA nodes of the Cray XE-6, SGI UV, Altix, Origin 2000



[en.wikipedia.org/wiki/
Non-Uniform_Memory_Access](http://en.wikipedia.org/wiki/Non-Uniform_Memory_Access)

Heterogeneous processing with Graphical Processing Units

- Specialized *many-core* processor
- Explicit data motion
 - between *host* and *device*
 - inside the device

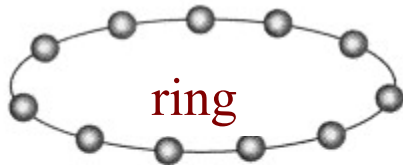


Today's lecture

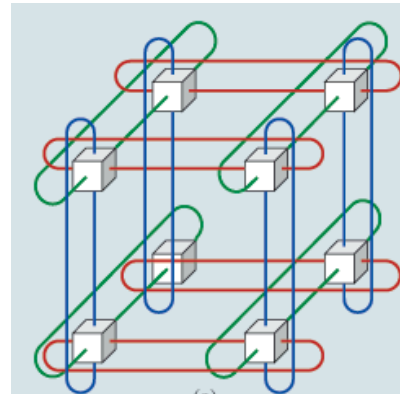
- Technology trends
- Address space organization
- Interconnect
- Algorithms (applications)
- Parallel program design and implementation
- Programming models
- Performance

Interconnect

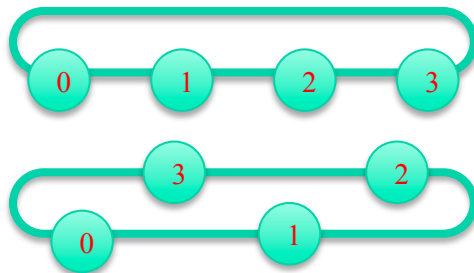
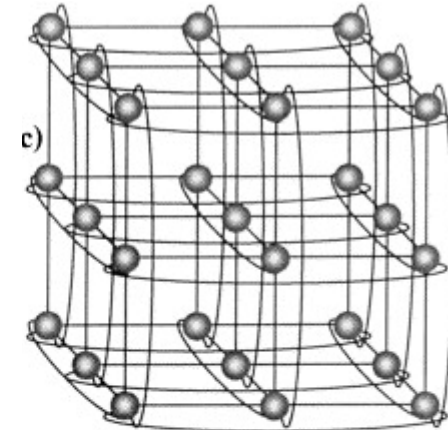
- Toroidal mesh (end around), ring
- K-ary n-cube (K^n nodes)
- Hypercube
- Diameter and bisection bandwidth



3d hypercube

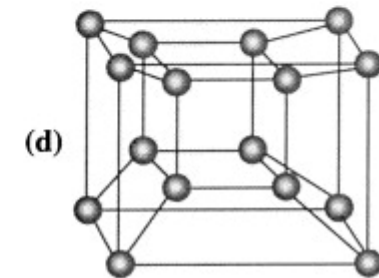
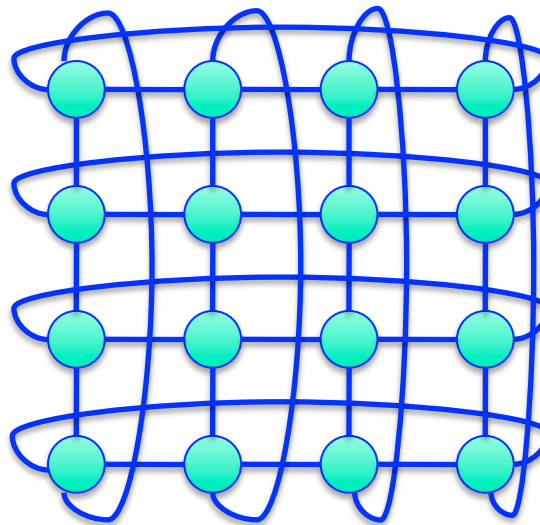


3d toroidal mesh



Mapping a ring to avoid
long wiring

Natalie EnrightJerger



4d hypercube

Sciencedirect.com

Today's lecture

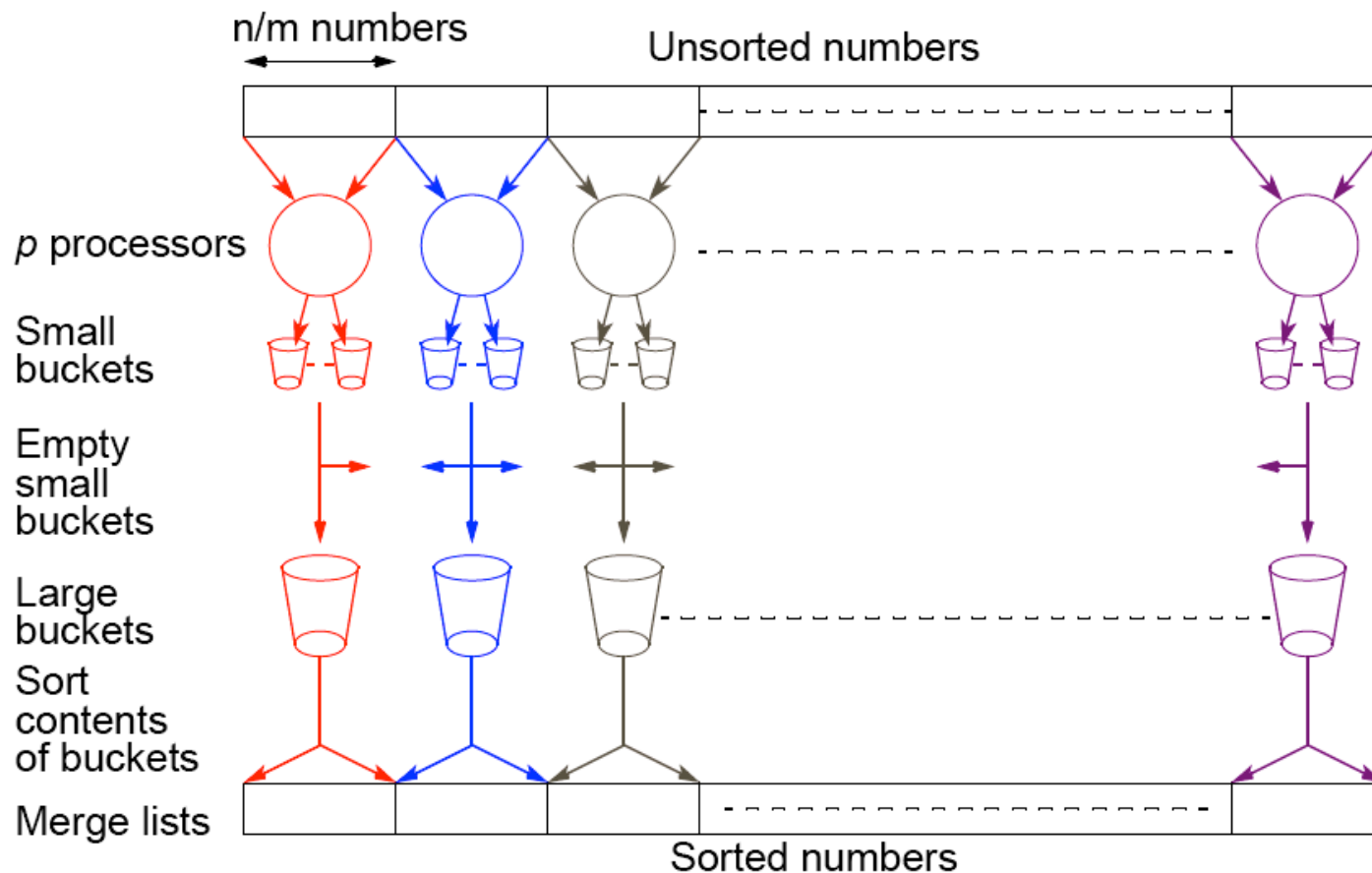
- Technology trends
- Address space organization
- Interconnect
- **Algorithms (applications)**
- Parallel program design and implementation
- Programming models
- Performance

Algorithms

- Sorting
 - Odd-even sort, bucket sort, sample sort
- Numerical integration (trapezoidal rule)
- Mandelbrot set
- Stencil methods
 - Ghost cells, partitioning
- Matrix multiplication: Cannon's Algorithm

Bucket sort

- Each process has 2 large buckets for input & output
- P small buckets for routing
- Worst and best cases?



B. Wilkinson

Why numerically intensive applications?

- Highly repetitive computations are prime candidates for parallel implementation
 - Improve quality of life, economically and technologically important
 - Data Mining
 - Image processing
 - Simulations – financial modeling, weather, biomedical
 - We can classify applications according to Patterns of *communication* and computation that persist over time and across implementations
- Phillip Colella's 7 Dwarfs

Courtesy of Randy Bank

Classifying the application domains

- ▶ Patterns of *communication* and computation that persist over time and across implementations

- ▶ Structured grids

- Panfilov method

- ▶ Dense linear algebra

- Matrix multiply, Vector-Mtx Mpy
Gaussian elimination

- ▶ N-body methods

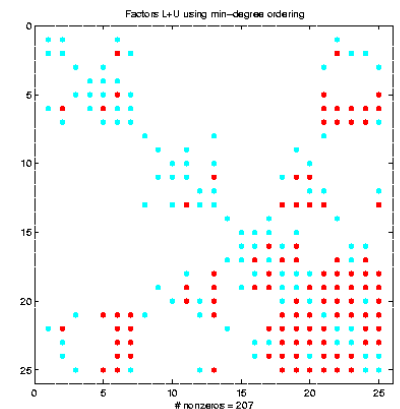
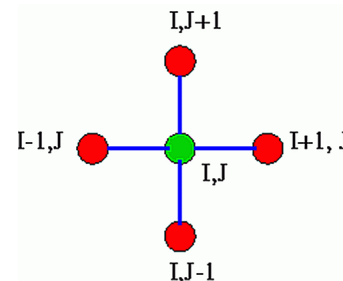
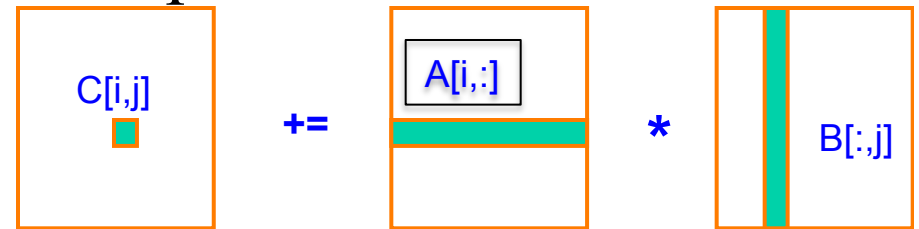
- ▶ Sparse linear algebra

- In a sparse matrix, we take advantage of knowledge about the locations of non-zeros, improving some aspect of performance

- ▶ Unstructured Grids

- ▶ Spectral methods (FFT)

- ▶ Monte Carlo



Courtesy of Randy Bank

Application-specific knowledge is important

- There currently exists no tool that can convert a serial program into an efficient parallel program

... for all applications ... all of the time ... on all hardware

- The **more we know** about the application...
 - ... specific problem ... math/physics ... initial data ...
 - ... context for analyzing the output...
 - ... **the more we can improve productivity**
- Issues
 - Data motion and locality
 - Load balancing
 - Serial sections

Today's lecture

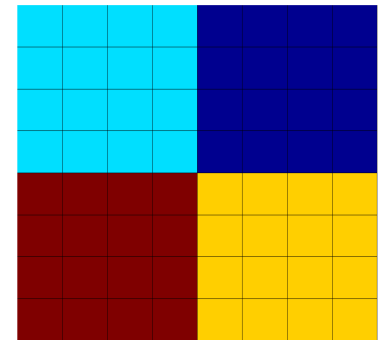
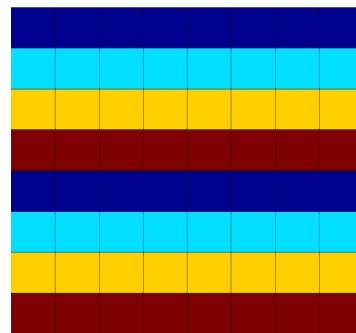
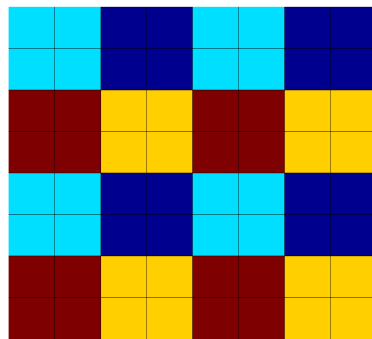
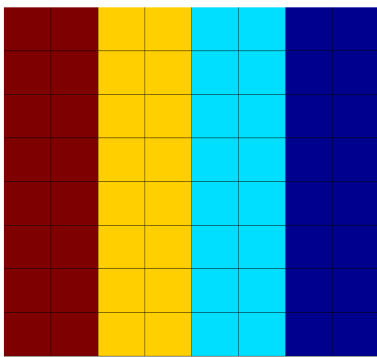
- Technology trends
- Address space organization
- Interconnect
- Algorithms (applications)
- **Parallel program design and implementation**
- Programming models
- Performance

Parallel program design and implementation

- Code organization and re-use
- Parallelizing serial code
(code reorganization and restructuring)
- Message passing, shared memory, hybrid
- How to design with parallelism in mind
- Performance

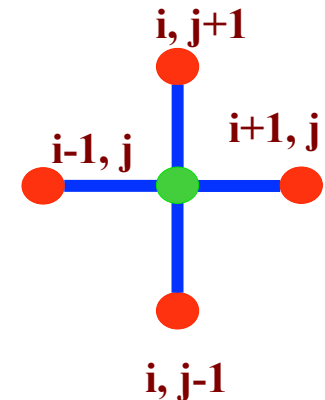
SPMD Implementation techniques

- Pthreads and OpenMP
- MPI
- Processor geometry and data distribution
 - BLOCK
 - Block Cyclic
- Owner computes rule

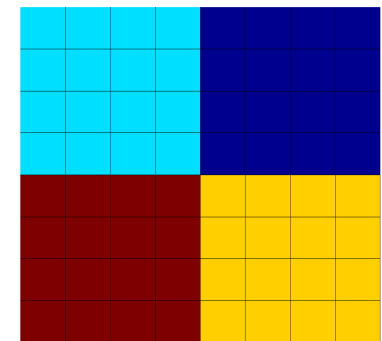
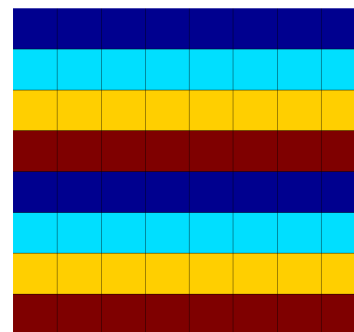
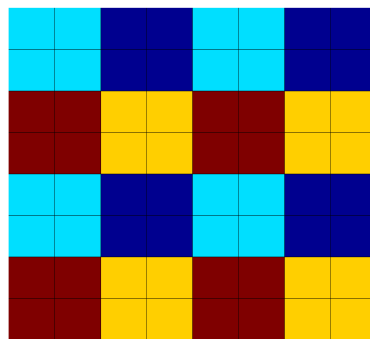
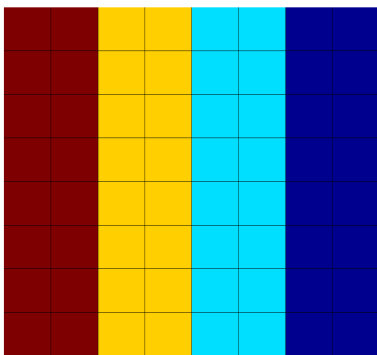


Owner computes rule

```
#pragma omp parallel for schedule(static,CHUNK)
for (int j=1; j<=m+1; j++)
    for (int i=1; i<=n+1; i++)
        E[j][i] = E_prev[j][i]+alpha*(E_prev[j][i+1]+E_prev[j][i-1]-
            4*E_prev[j][i]+E_prev[j+1][i]
            +E_prev[j-1][i])
```



```
#define E' [i,j] E_prev[(j+1)*(m+3) + (i+1)]
I = blockIdx.y*blockDim.y + threadIdx.y;
J = blockIdx.x*blockDim.x + threadIdx.x;
if ((I <= n) && (J <= m) )
    E[I]=E' [I,J] +  $\alpha$ *(E'[I-1,J] + E'[I+1,J] +E'[I,J-1]+E' [I,J+1] - 4*E' [I,J]);
```



Today's lecture

- Technology trends
- Address space organization
- Interconnect
- Algorithms (applications)
- Parallel program design and implementation
- **Programming models**
- **Performance**

Multithreading

- Primitives
 - Thread launching and joining
 - Atomic variables, mutual exclusions, barriers
- New storage class: shared
- Work scheduling
 - Static vs dynamic

Multithreading in perspective

- Benefits
 - Harness parallelism to improve performance
 - Ability to multitask to realize concurrency, e.g. display
- Pitfalls
 - Program complexity
 - Partitioning, synchronization, parallel control flow
 - Data dependencies
 - Shared vs. local state (globals like errno)
 - Thread-safety
 - New aspects of debugging
 - Race conditions
 - Deadlock

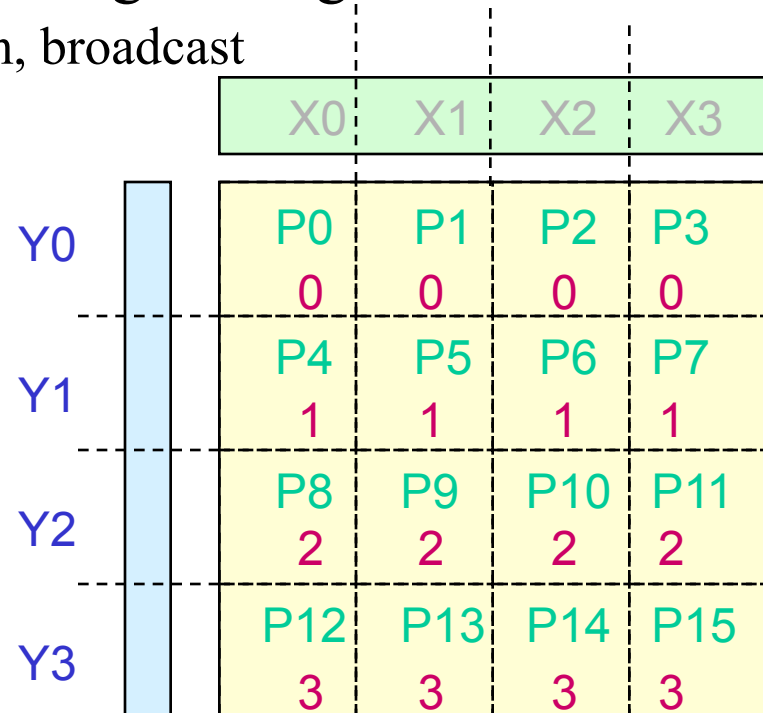
Message passing

- No globally accessible storage
- MPI
- Blocking vs non-blocking communication
- Point to point vs collective
- Collectives distinguish short and long messages
 - Gather/scatter, All-to-all, reduction, broadcast
 - Allreduce, Gatherv, Alltoallv
- Message filtering
- Communicators

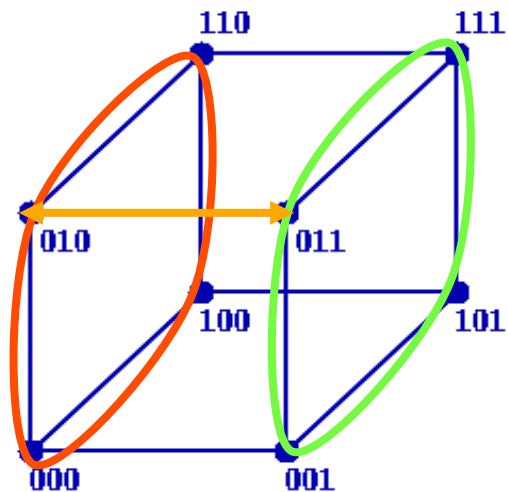
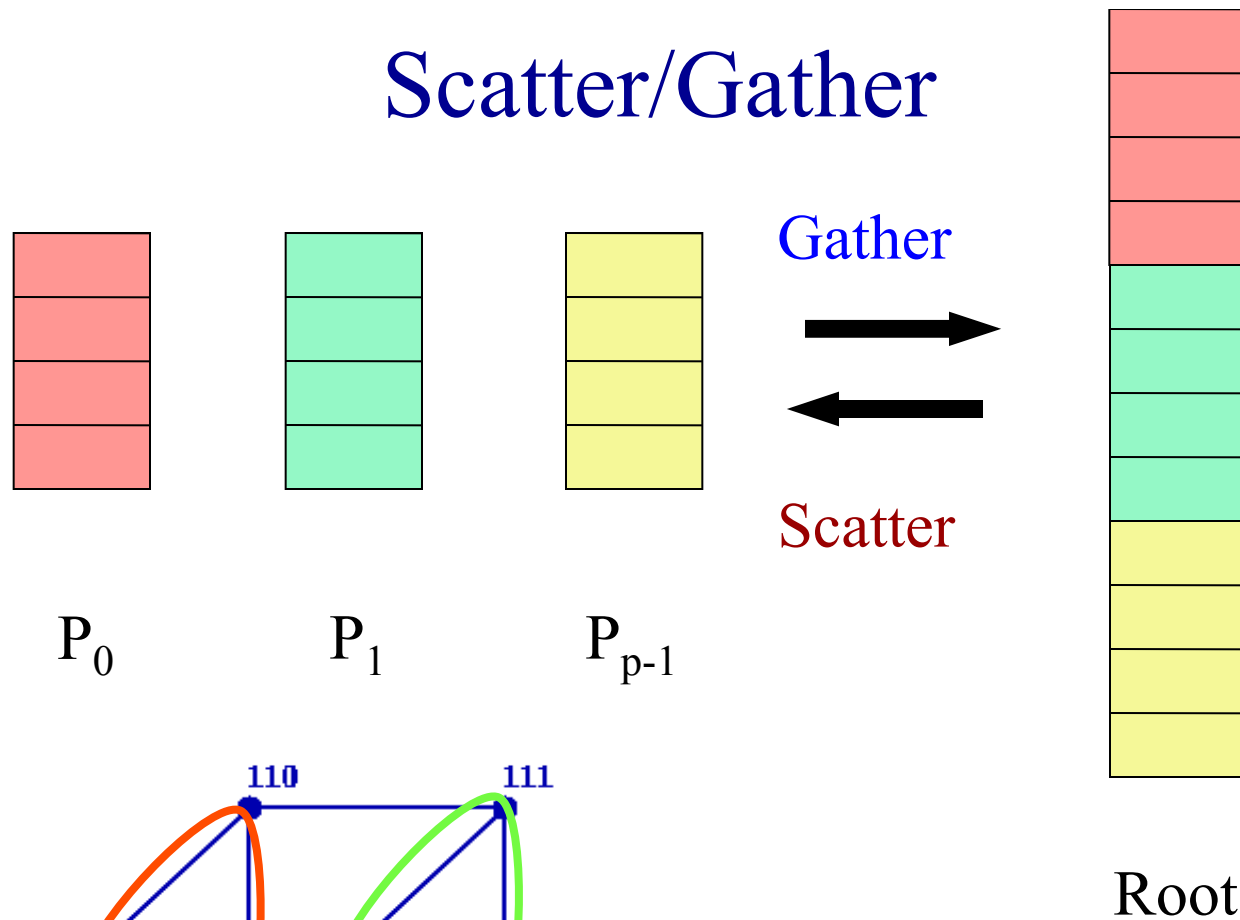
MPI_Comm rowComm;

MPI_Comm_split(MPI_COMM_WORLD,
myRank / $\sqrt{P}$, myRank, &rowComm);

MPI_Comm_rank(rowComm,&myRow);



Scatter/Gather



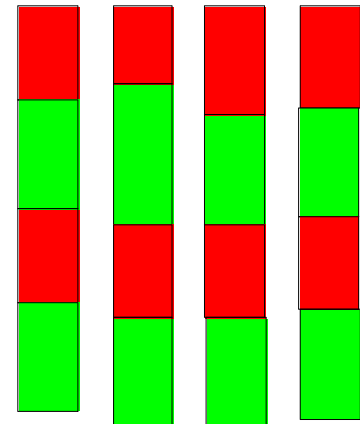
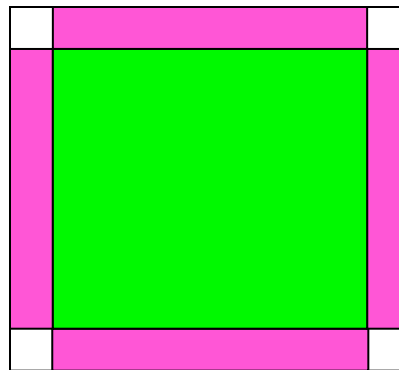
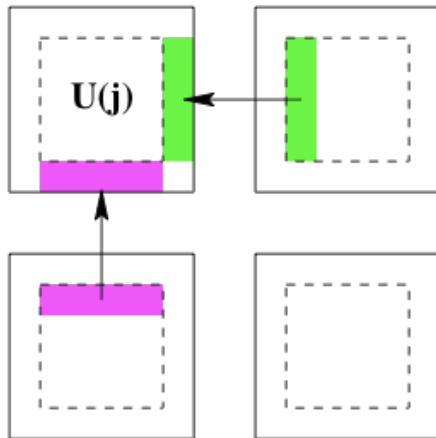
Short message algorithm

Message passing in perspective

- Benefits
 - Processes communicate explicitly, no anonymous updates
 - Message arrival provides synchronization
- Pitfalls
 - Must replace synchronization with explicit data motion ($\pm$)
 - Can't rely on the cache to move data between processes living on separate processing nodes
 - Ghost cells, collectives
 - Harder to incrementally parallelize code than with threads

A Classic message passing implementation of a stencil method

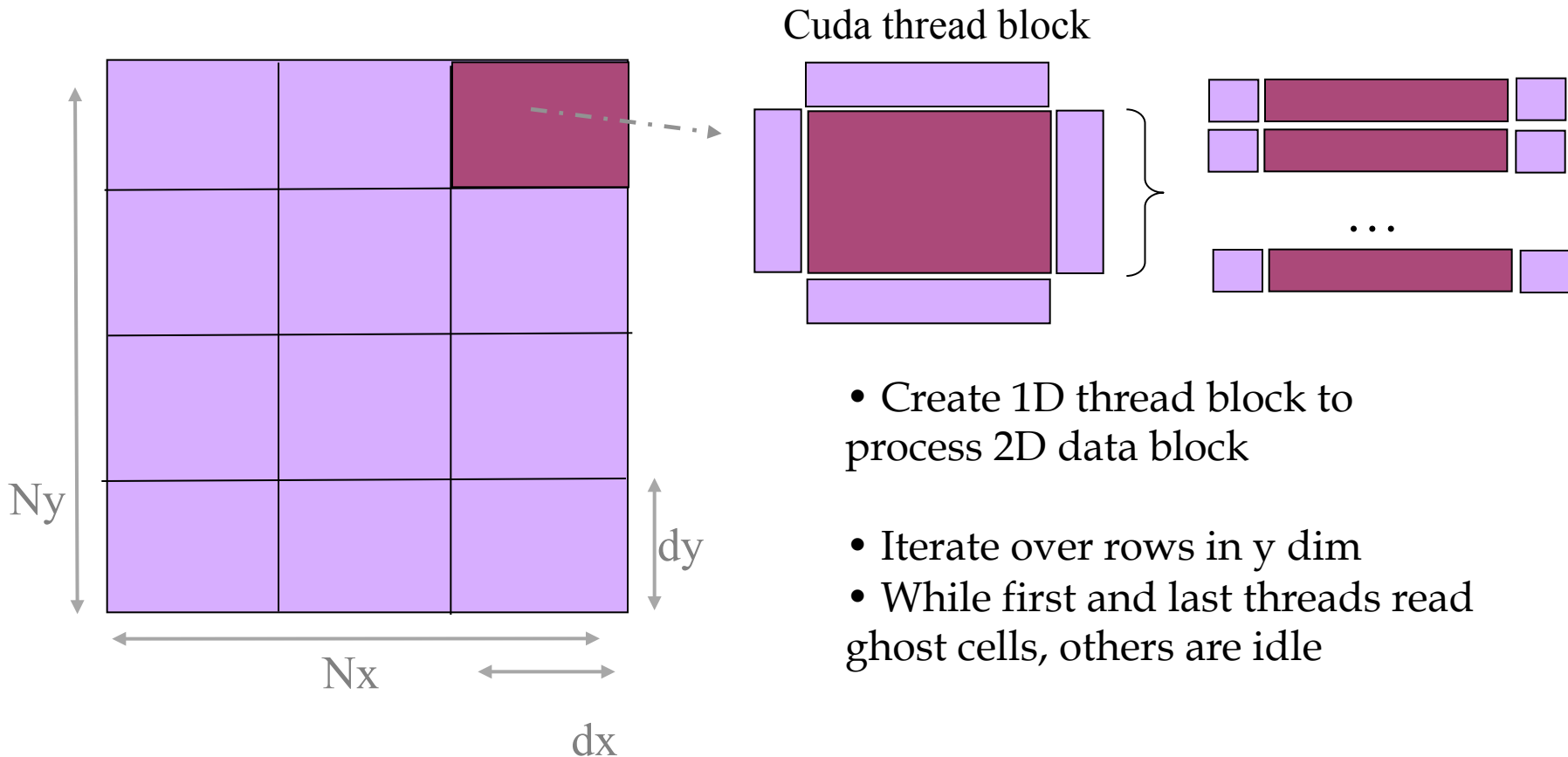
- Decompose domain into sub-regions, one per process
 - Transmit **halo regions** between processes
 - Compute **inner region** after communication completes
- Loop carried dependences impose a strict ordering on communication and computation



Programming with GPUs

- Benefits: simplified processor design enables higher performance for targeted workloads, lower power consumption per flop
- Pitfalls
 - Manage on-chip memory explicitly (shared memory) though caches on some designs highly effective
 - Branches are costly (OK within a warp)
 - More complicated programming model

Using Shared Memory



Compared to a 2D thread blocking, 1D thread blocks provide a 12% improvement in double precision and 64% improvement in single precision

Didem Unat

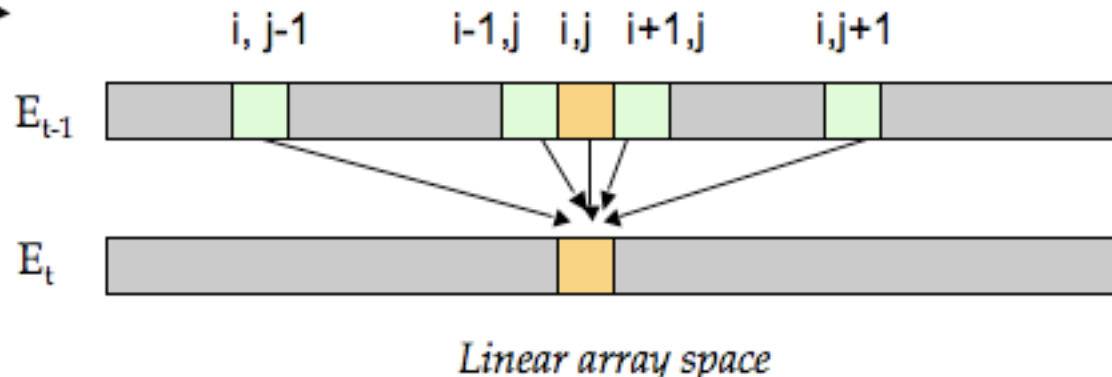
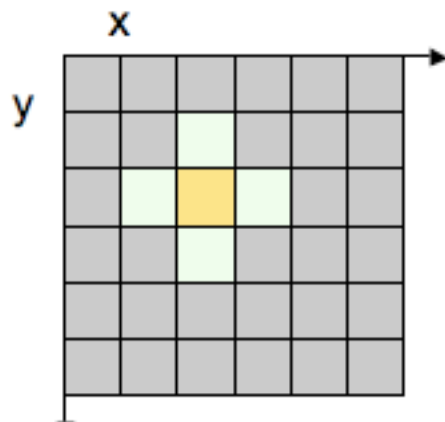
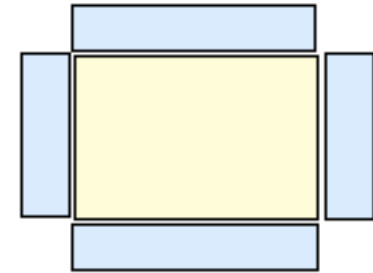
CUDA Code for computational loop

```
__shared__ float block[DIM_Y + 2 ][DIM_X + 2 ];  
int idx = threadIdx.x, idy = threadIdx.y ; //local indices  
//global indices  
int x = blockIdx.x * (DIM_X) + idx;  
int y = blockIdx.y * (DIM_Y) + idy;  
idy++; idx++;  
unsigned int index = y * N + x ;  
  
//interior points  
float center = E_prev[index] ;  
block[idy][idx] = center;  
  
__syncthreads();
```

Ghost cell copying - thread divergence

When loading ghost cells, only some of the threads are active, some are idle

```
if (idy == 1 && y > 0 )  
    block[0][idx] = E_prev[index - N];  
else if(idy == DIM_Y && y < N-1)  
    block[DIM_Y+1][idx] = E_prev[index + N];  
if ( idx==1 && x > 0 )  
    block[idy][0] = E_prev[index - 1];  
else if( idx== DIM_X && x < N-1 )  
    block[idy][DIM_X + 1] = E_prev[index + 1];  
__syncthreads();
```



Today's lecture

- Technology trends
- Address space organization
- Interconnect
- Algorithms (applications)
- Parallel program design and implementation
- Programming models
- **Performance**

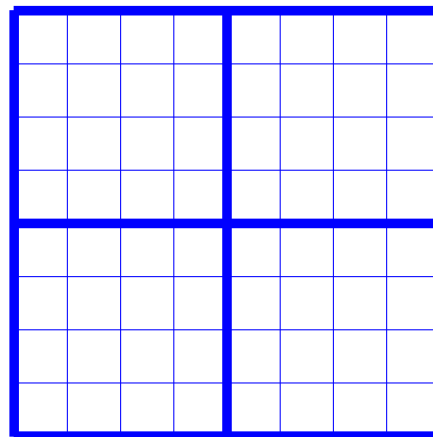
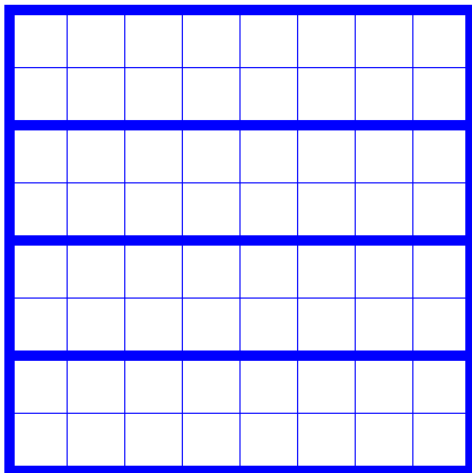
Communication costs -stencil method

- 1-D decomposition

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

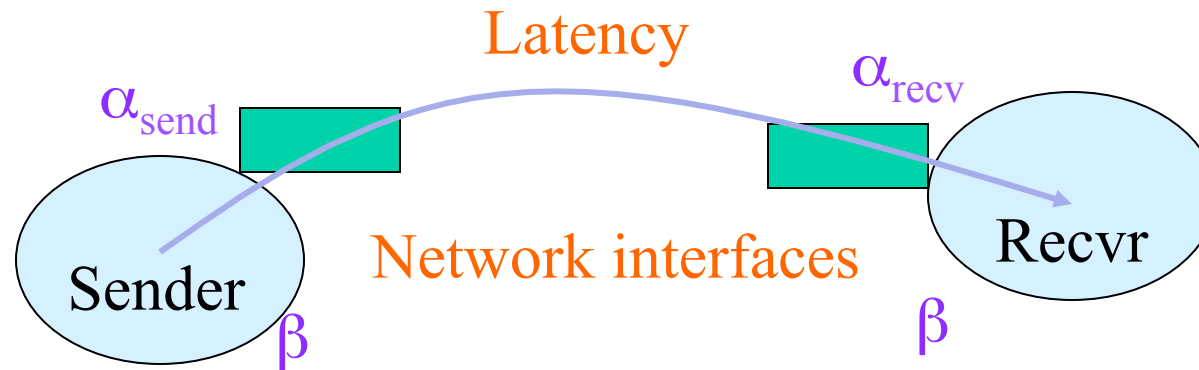
- 2-D decomposition

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



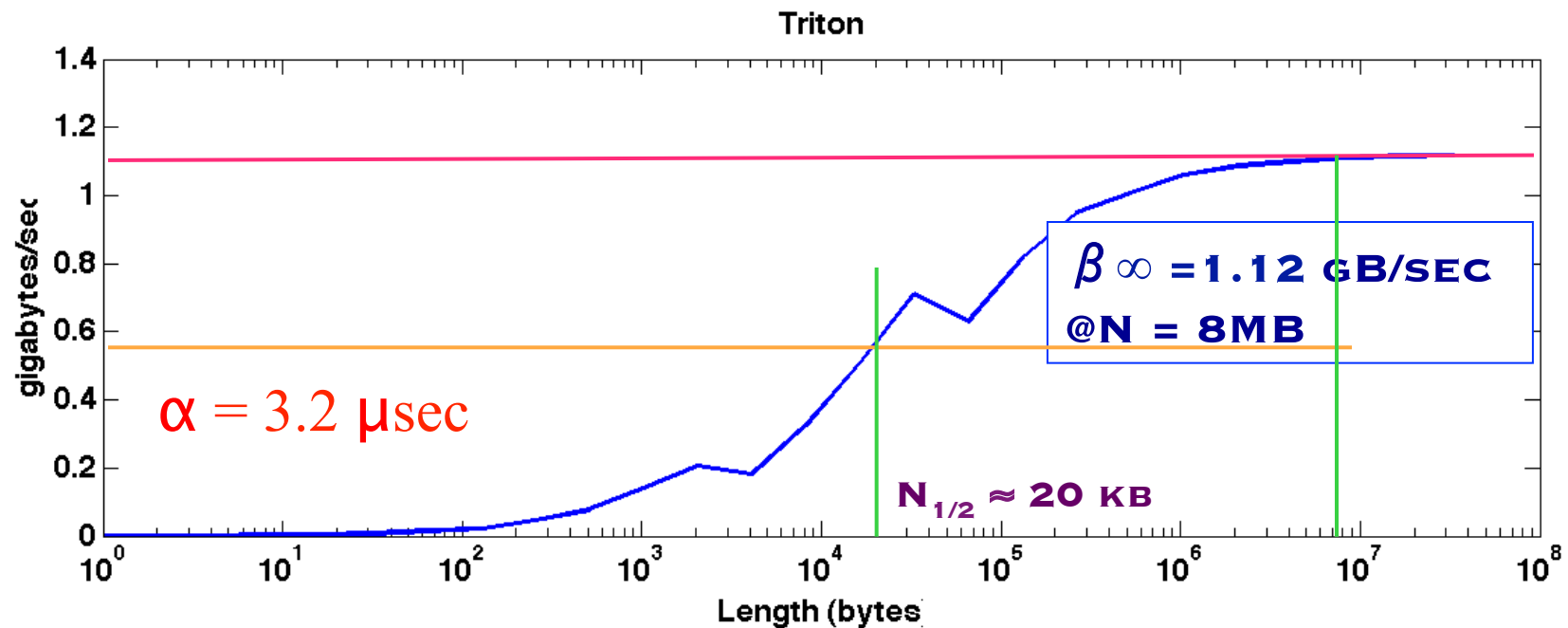
Messaging system design and implementation

- Short vs long messages, half power point
- Eager vs. rendezvous communication
- Hypercube algorithms for collectives
- Message buffering, completion



α - β model of communication time

- Triton.sdsc.edu



Questions

1. Performance
2. Cache coherence
3. Ghost Cells

1. Performance (Q1)

- This is too hard. The 2nd phase should consume a fraction f on 2 processor
- We run on 16 processors
- An application with 2 phases
 - Phase 1: perfectly parallelizable, $E_{16} = 100\%$
 - Phase 2: $E_{16} = 25\%$ efficiency on 16 processors
Serial section: f of the overall running time on *one* processor (best serial algorithm): fT_1
- Express T_{16} in terms of f , as a fraction of the form x/y

2. Cache coherence

```
const int BIG_NUMBER = ...
```

```
//shared
```

```
int sharedVar = 0;
```

```
while(sharedVar < BIG_NUMBER){
```

```
    CRITICAL SECTION:
```

```
    sharedVar++;
```

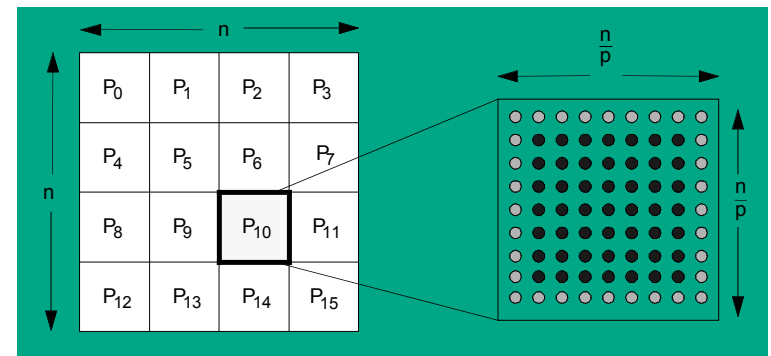
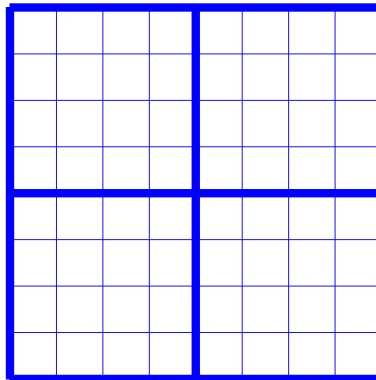
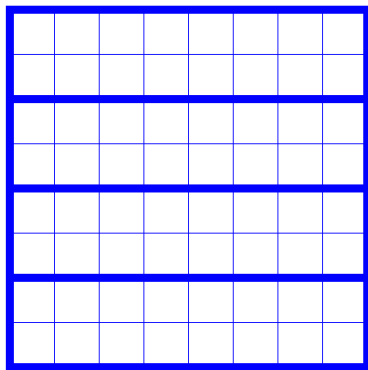
```
}
```

- We run with 2 threads
- We run with 100 threads
- What can prevent threads from updating the shared variable?

3. Ghost cells

- **Image smoother**
repeat until done:
 for $i = 1 : N-2$
 for $j = 1 : N-2$
 $U^{\text{new}}[i,j] = (U[i+1,j] + U[i-1,j] + U[i,j+1] + U[i,j-1])/4;$
 Swap U^{new} and U
- What is the max number of ghost cells that a processor will need to transmit when P does not divide N evenly?

$$N/\sqrt{P}$$



Fin

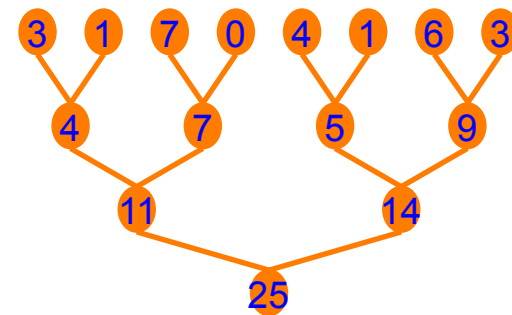
5. Tree Summation

- Input: an array $x[]$, length $N \gg P$
- Output: Sum of the elements of $x[]$
- Goal: Compute the sum in $\lg P$ time

```
sum = 0;  
for i=0 to N-1  
    sum += x[i]
```

- Assume P is a power of 2, $K = \lg P$
- Starter code
for $m = 0$ to $K-1$ {

}
}



Correctness

Global Change, $I[:,:]$, $I^{\text{new}}[:,:]$

Local $\text{mymin} = 1 + (\$TID * n / \$NT)$,

$\text{mymax} = \text{mymin} + n / \$NT - 1$;

Local $\text{done} = \text{FALSE}$;

while (!done) do

Local $\text{myChange} = 0$;

BARRIER

Only on thread 0: $\text{Change} = 0$;

BARRIER

update I^{new} and myChange

CRITICAL SEC: $\text{Change} += \text{myChange}$

BARRIER

if ($\text{myChange} < \text{Tolerance}$) $\text{done} = \text{TRUE}$;

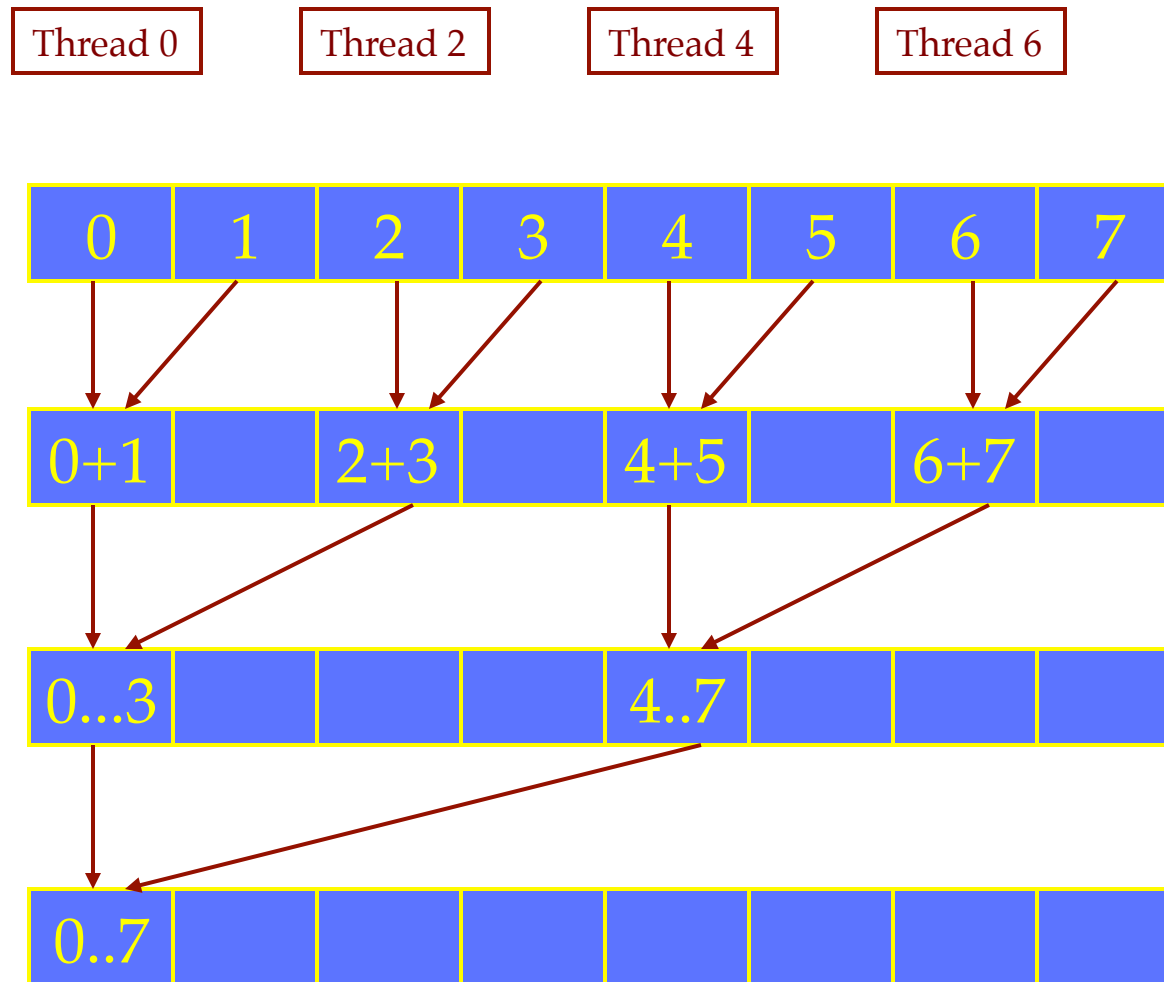
Only on thread 0: Swap pointers: $I \leftrightarrow I^{\text{new}}$

end while

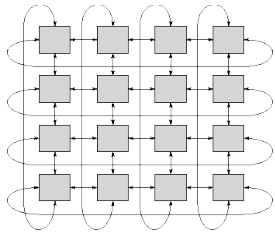
0
1
2
3

Does this code use minimal synchronization?

Visualizing the Summation



Extra



Ian Foster

<http://www.mcs.anl.gov/~itf/dbpp/text/node33.html>