Parallelism for the Masses

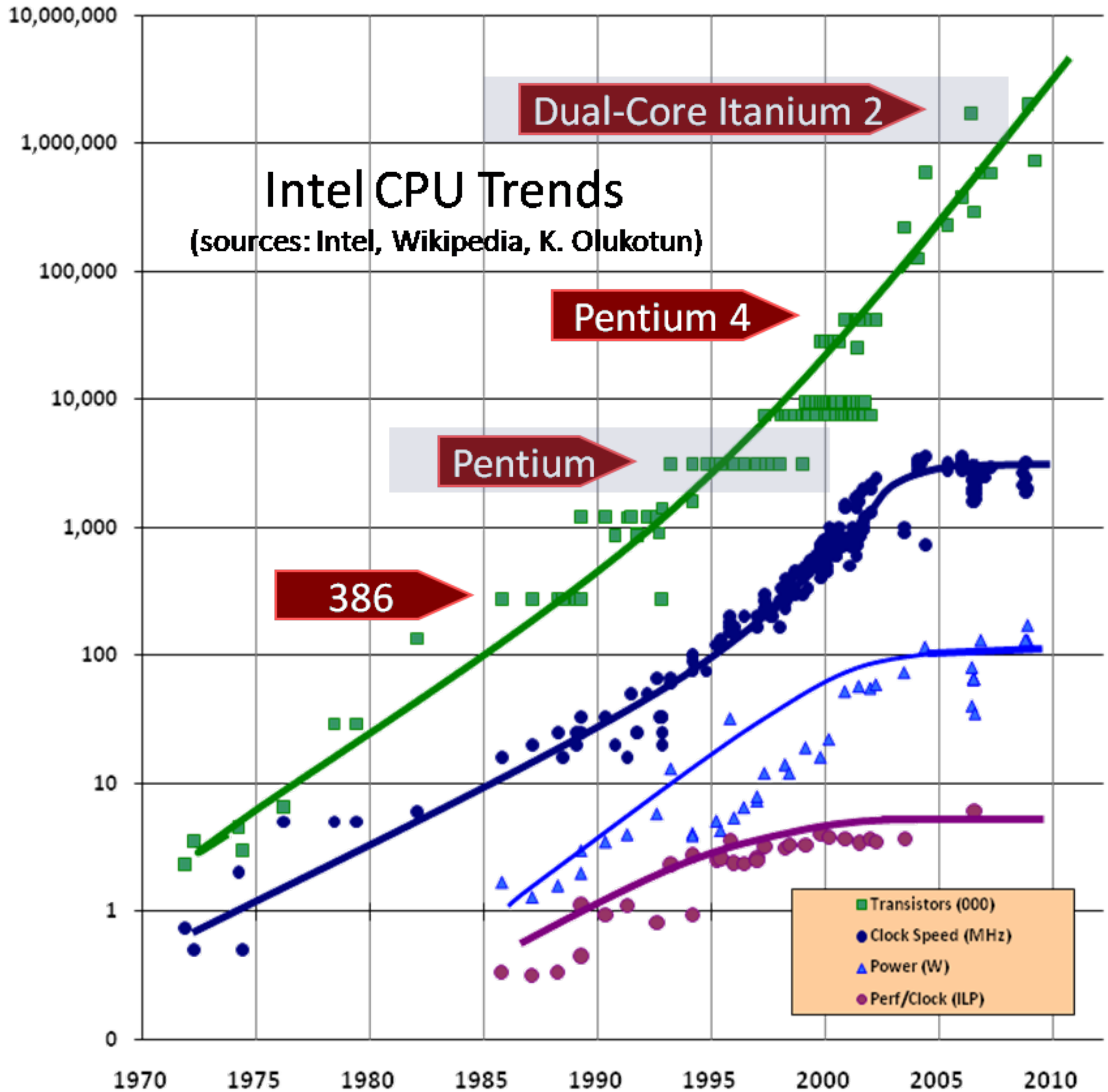
Performance to Productivity

Arun Chauhan

School of Informatics and Computing
Indiana University, Bloomington, USA

Auburn University
October 3, 2011

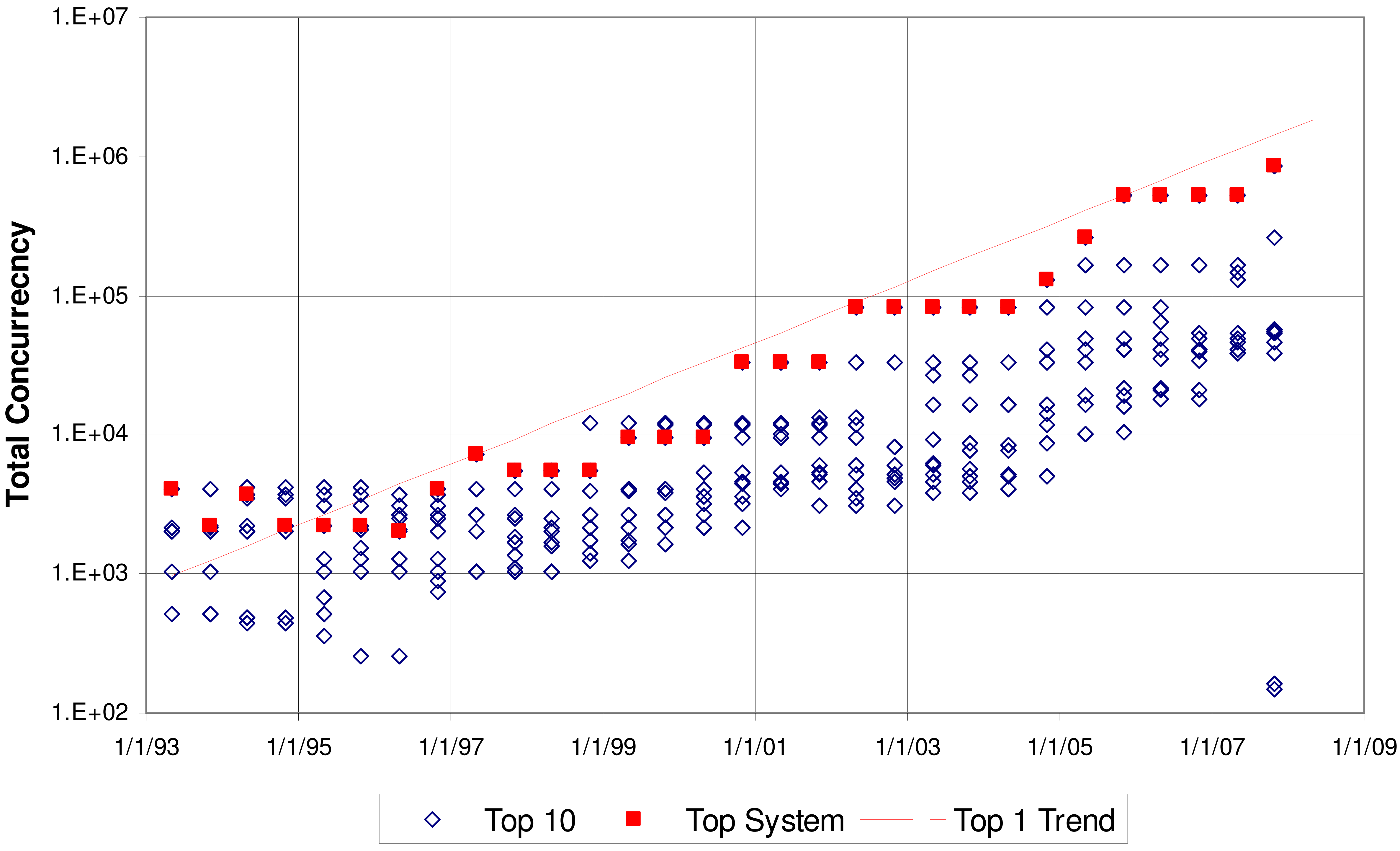
The Free Lunch is Over



Herb Sutter, *The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software, Dr. Dobb's Journal, 30(3), March 2005*

Exa-scale Challenge

Trends in Concurrency



***Peter Kogge** et al. Exascale Computing Study, Technology Challenges in Achieving Exascale Systems, 2008.*

Long History of Parallelism

- Vector processors
- Symmetric multi-processors (SMPs)
- Nodes over inter-connection networks
- Instruction-level parallelism
- Multi-cores
- GPUs
- ...

Déjà vu all over again?

Déjà vu all over again?

"... today's processors ... are nearing an impasse as technologies approach the speed of light.."

David Mitchell, The Transputer: The Time Is Now (1989)

Déjà vu all over again?

"... today's processors ... are nearing an impasse as technologies approach the speed of light.."

David Mitchell, The Transputer: The Time Is Now (1989)

"We are dedicating all of our future product development to multicore designs. ... This is a sea change in computing"

Paul Otellini, President, Intel (2004)

Déjà vu all over again?

"... today's processors ... are nearing an impasse as technologies approach the speed of light.."

David Mitchell, The Transputer: The Time Is Now (1989)

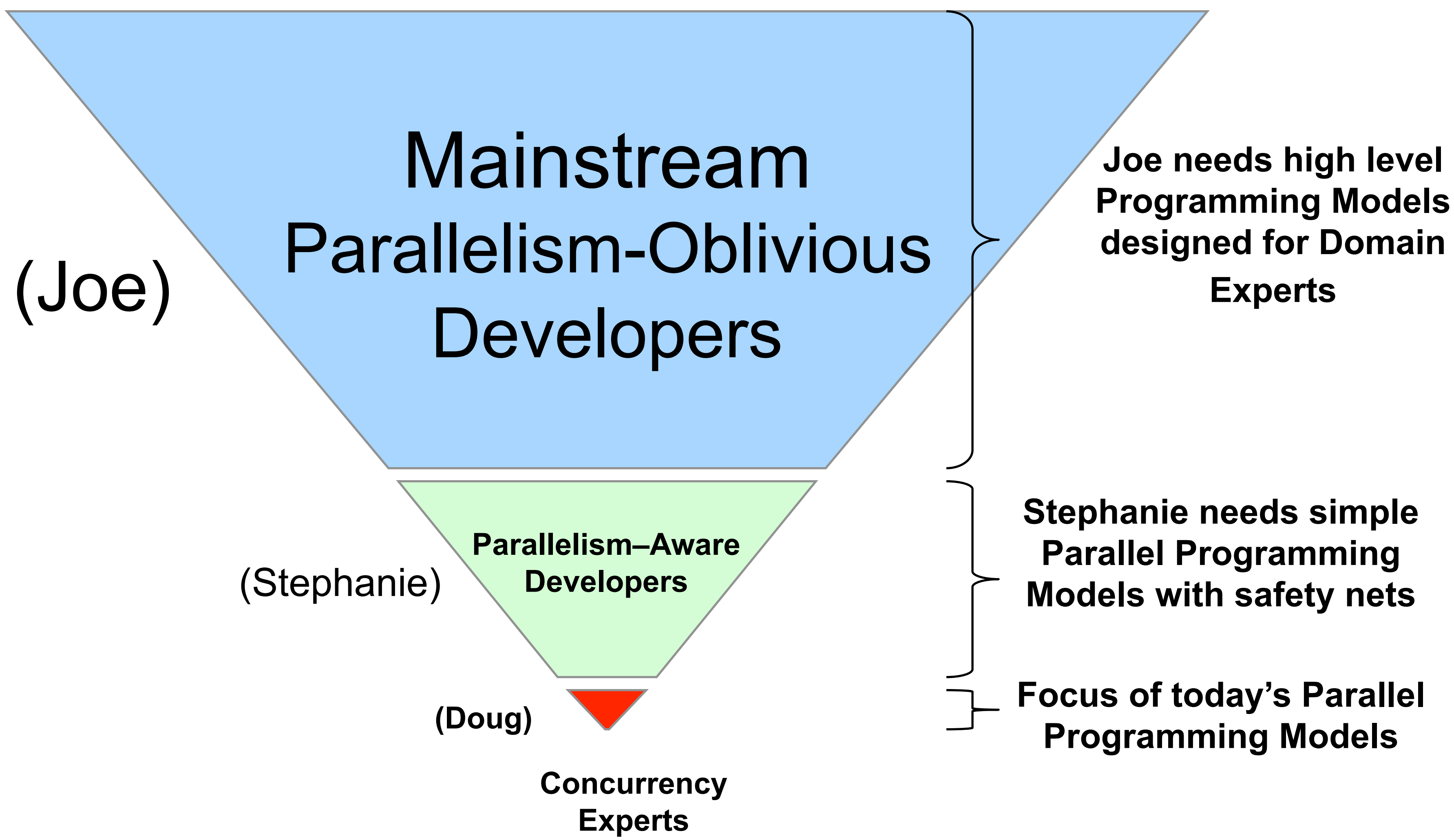
"We are dedicating all of our future product development to multicore designs. ... This is a sea change in computing"

Paul Otellini, President, Intel (2004)

Difference is all microprocessor companies have switched to multiprocessors (AMD, Intel, IBM)
⇒ Procrastination penalized: 2X sequential perf. / 5 yrs

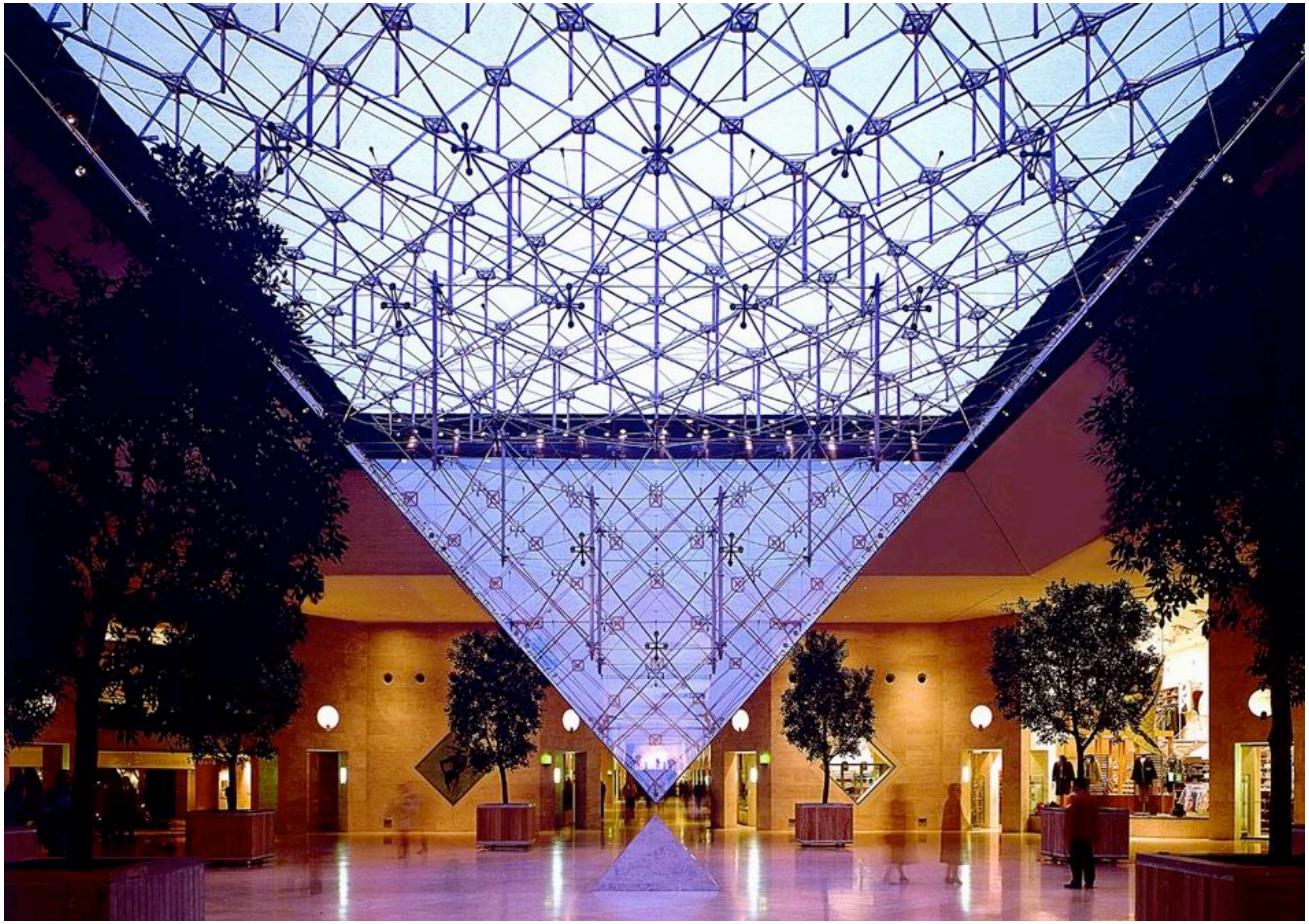
⇒ Biggest programming challenge: 1 to 2 CPUs

Parallelism



Courtesy: Vivek Sarkar, Rice University

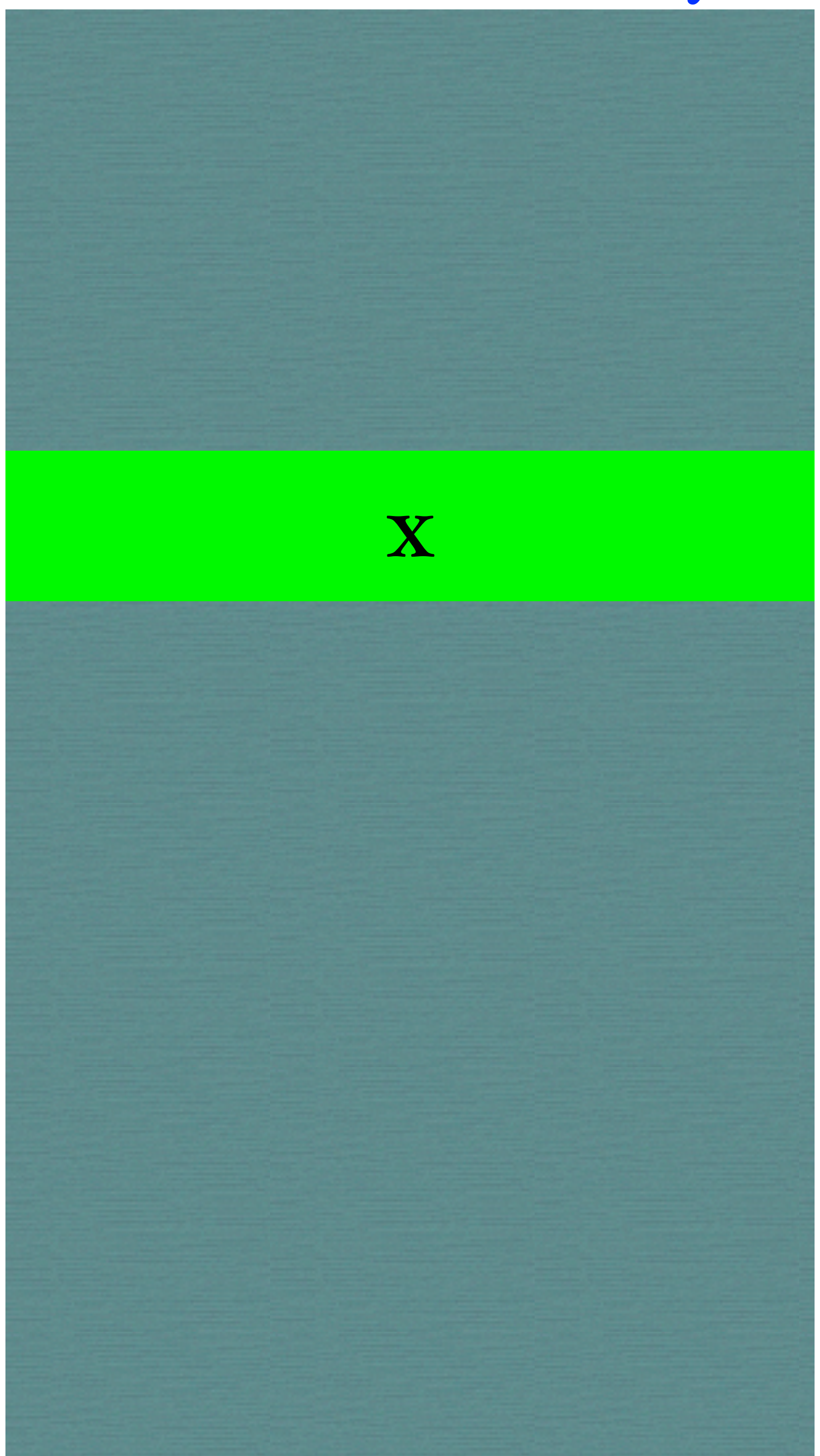
Parallelism



Parallelism

Paralleling Programming on a Slide

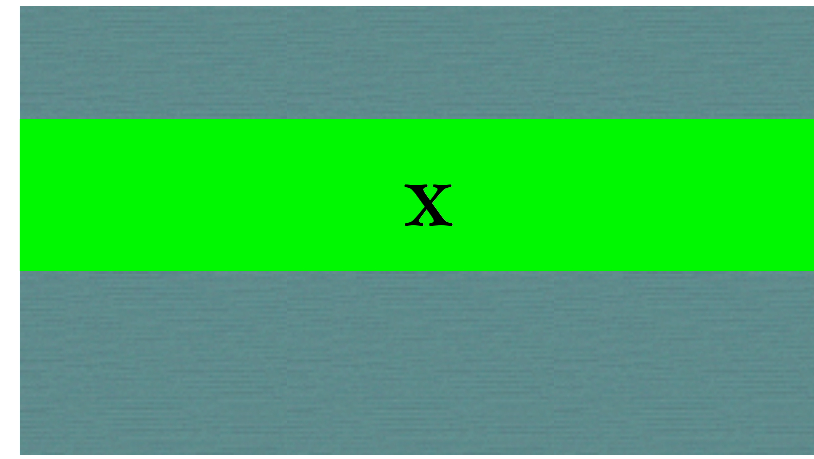
Shared Memory

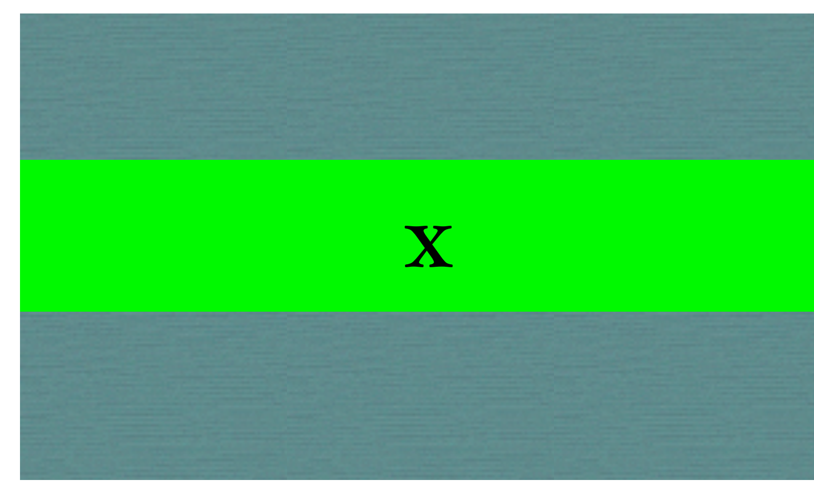



```
x = 10;
...
y = x + 2;
```

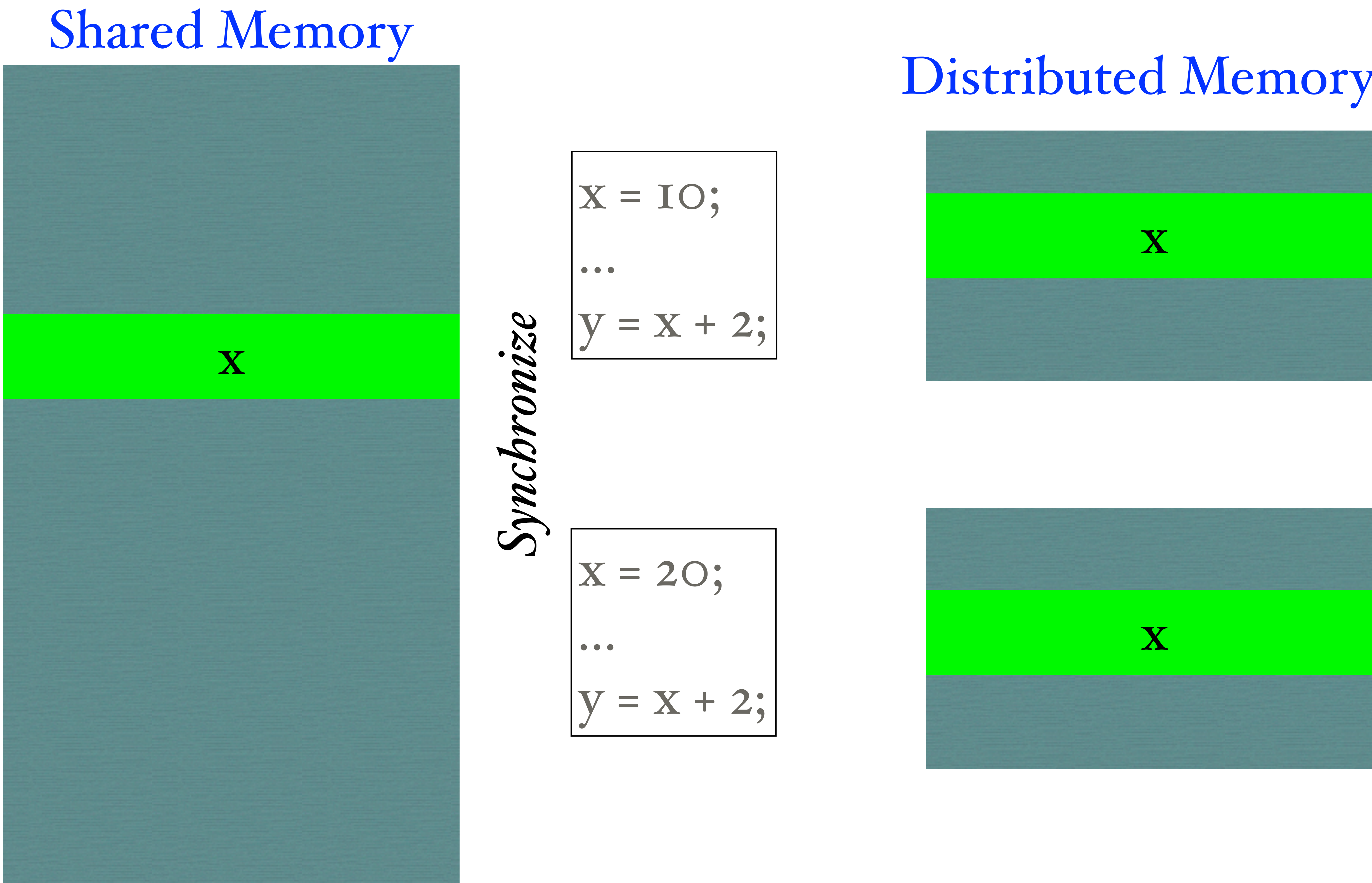
```
x = 20;
...
y = x + 2;
```

Distributed Memory

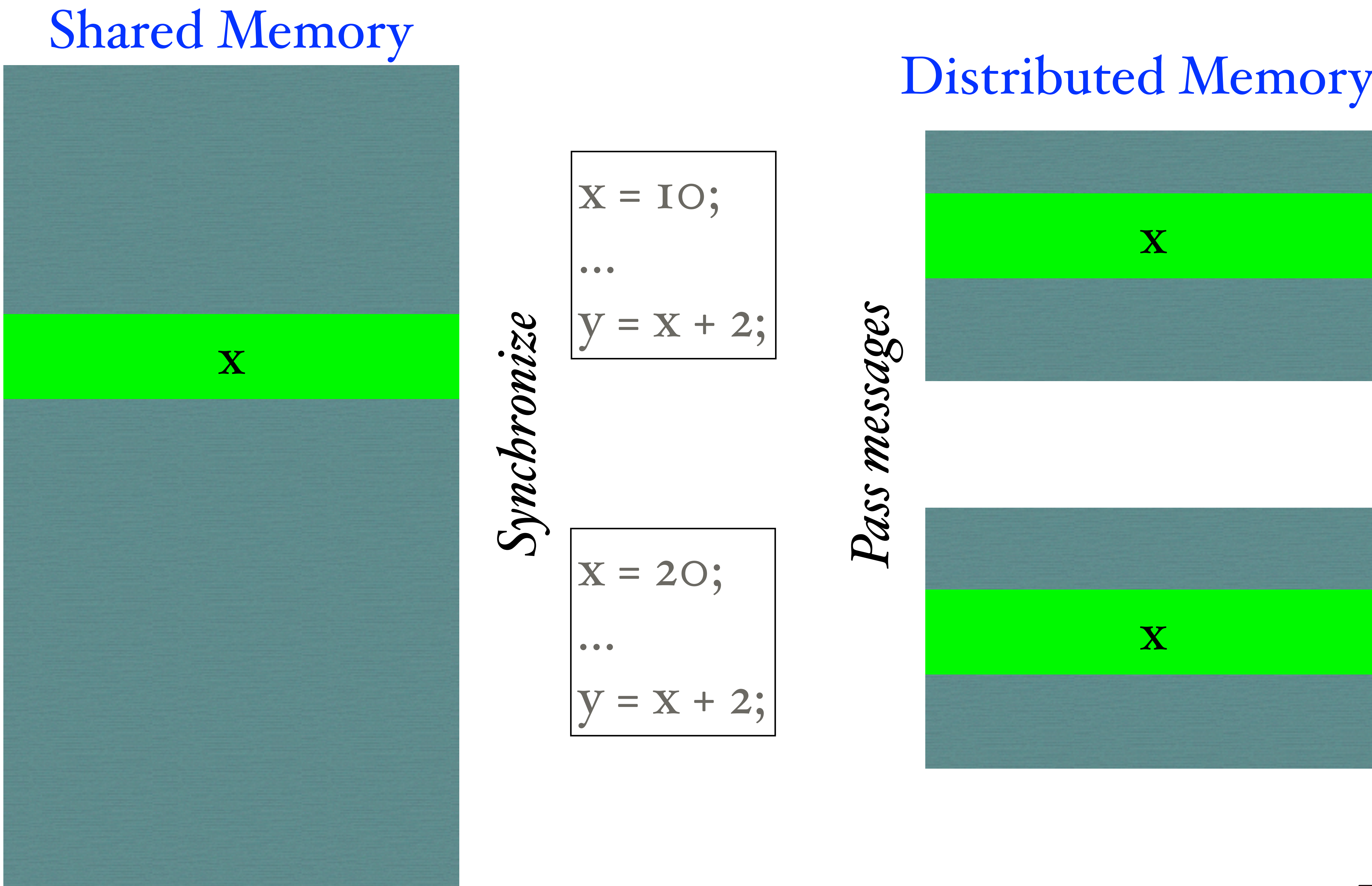




Paralleling Programming on a Slide



Paralleling Programming on a Slide



Thinking of Stephanie programmers

High Performance Fortran

```
PROGRAM SUM
    REAL A(10000)
    READ (9) A
    SUM = 0.0
    DO I = 1, 10000
        SUM = SUM + A(I)
    ENDDO
    PRINT SUM
END
```

High Performance Fortran

```
PROGRAM SUM
    REAL A(10000)
    READ (9) A
    SUM = 0.0
    DO I = 1, 10000
        SUM = SUM + A(I)
    ENDDO
    PRINT SUM
END
```

```
PROGRAM PARALLEL_SUM
    REAL A(100), BUFF(100)
    IF (PID == 0) THEN
        DO IP = 0, 99
            READ (9) BUFF(1:100)
            IF (IP == 0) A(1:100) = BUFF(1:100)
            ELSE SEND(IP, BUFF, 100) ! 100 words to Proc 1
        ENDDO
    ELSE
        RECV(0, A, 100) ! 100 words from proc 0 into A
    ENDIF
    SUM = 0.0
    DO I = 1, 100
        SUM = SUM + A(I)
    ENDDO
    IF (PID == 0) SEND(1, SUM, 1)
    IF (PID > 0)
        RECV(PID-1, T, 1)
        SUM = SUM + T
        IF (PID < 99) SEND(PID+1, SUM, 1)
        ELSE SEND(0, SUM, 1)
    ENDIF
    IF (PID == 0) THEN; RECV(99, SUM, 1); PRINT SUM; ENDIF
END
```

High Performance Fortran

```
PROGRAM SUM
    REAL A(10000)
    READ (9) A
    SUM = 0.0
    DO I = 1, 10000
        SUM = SUM + A(I)
    ENDDO
    PRINT SUM
END
```

```
PROGRAM HPF_SUM
    REAL A(10000)
    !HPF$ DISTRIBUTE A(BLOCK)
    READ (9) A
    SUM = 0.0
    DO I = 1, 10000
        SUM = SUM + A(I)
    ENDDO
    PRINT SUM
END
```

HPF: Victim of its own Success?

- No prior compiler technology to learn from
- Limited number of data distribution primitives
  - not user expandable
- Paucity of good HPF libraries
- Lack of performance-tuning tools
- Lack of patience of user community!

Ken Kennedy, Charles Koelbel, and Hans Zima. *The Rise and Fall of High Performance Fortran: An Historical Object Lesson. In Proceedings of the third ACM SIGPLAN Conference on History of Programming Languages, pages 7-1–7-22, 2007.*

HPF: Victim of its own Success?

- No prior compiler technology to learn from
- Limited number of data distribution primitives
  - not user expandable
- Paucity of good HPF libraries
- Lack of performance-tuning tools
- Lack of patience of user community!

Does not motivate users to think in parallel

Ken Kennedy, Charles Koelbel, and Hans Zima. *The Rise and Fall of High Performance Fortran: An Historical Object Lesson. In Proceedings of the third ACM SIGPLAN Conference on History of Programming Languages, pages 7-1–7-22, 2007.*

Design Principles

- Users must think in parallel (creativity)
  - but not be encumbered with optimizations that can be automated, or proving synchronization correctness
- Compiler focuses on what it can do (mechanics)
  - not creative tasks, such as determining data distributions, or creating new parallel algorithms
- Incremental deployment
  - not a new programming language
  - more of a *coordination language* (DSL)
- Formal semantics
  - provable correctness

Overview of Our Solution

- Declarative approach to parallel programming
  - focus on *what*, not *how*
  - partitioned address space
- Code generation
  - data movement
  - GPU kernel splitting
- Compiler optimizations
  - data locality
  - GPU memory hierarchy (including registers)

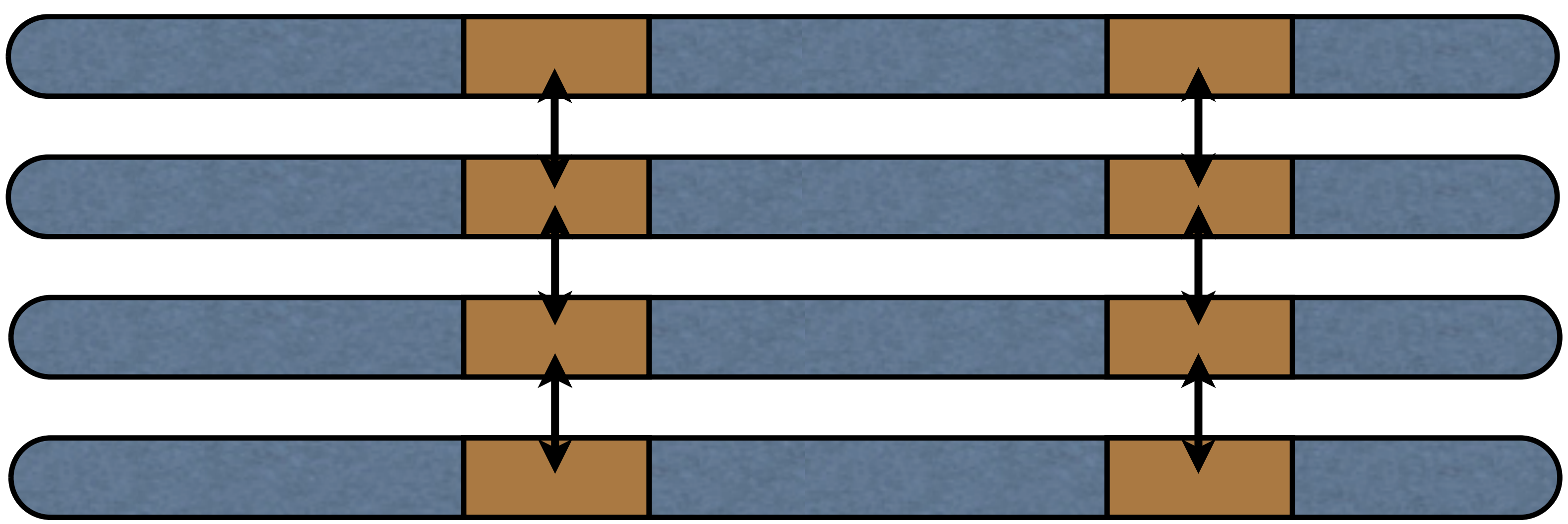
Torsten Hoefler, Jeremiah Willcock, Arun Chauhan, and Andrew Lumsdaine. *The Case for Collective Pattern Specification. In Proceedings of the First Workshop on Advances in Message Passing (AMP), 2010. Held in conjunction with the ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI).*

Declarative Approach

- Originally motivated by Block-synchronous Parallel (BSP) programs, especially for collective communication
    - alternate between computation and communication
    - communication optimization breaks the structure

Declarative Approach

- Originally motivated by Block-synchronous Parallel (BSP) programs, especially for collective communication
  - alternate between computation and communication
  - communication optimization breaks the structure



Declarative Approach

- Originally motivated by Block-synchronous Parallel (BSP) programs, especially for collective communication
  - alternate between computation and communication
  - communication optimization breaks the structure

Declarative Approach

- Originally motivated by Block-synchronous Parallel (BSP) programs, especially for collective communication
    - alternate between computation and communication
    - communication optimization breaks the structure

- Extend to non BSP-style applications

Kanor for Clusters

$$\textbf{@communicate}\ \{\ b@recv\_rank <<= a@send\_rank\ \}$$

Eric Holk, William E. Byrd, Jeremiah Willcock, Torsten Hoefler, Arun Chauhan, and Andrew Lumsdaine. *Kanor: A Declarative Language for Explicit Communication. In Proceedings of the Thirteenth International Symposium on the Practical Aspects of Declarative Languages (PADL), 2011. Held in conjunction with the ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL).*

Kanor for Clusters

$$\textbf{@communicate}\ \{\ b@recv\_rank <<= a@send\_rank\ \}$$

$$e_0@e_1 << op << e_2@e_3\ \texttt{where}\ e_4$$

Eric Holk, William E. Byrd, Jeremiah Willcock, Torsten Hoefler, Arun Chauhan, and Andrew Lumsdaine. *Kanor: A Declarative Language for Explicit Communication. In Proceedings of the Thirteenth International Symposium on the Practical Aspects of Declarative Languages (PADL), 2011. Held in conjunction with the ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL).*

Kanor for Clusters

$$\textbf{@communicate} \; \{ \; b@recv\_rank <<= a@send\_rank \; \}$$

$$e_0@e_1 << op << e_2@e_3 \; \texttt{where} \; e_4$$

$$e_0@e_1 <<= e_2@e_3 \; \texttt{where} \; e_4$$

Eric Holk, William E. Byrd, Jeremiah Willcock, Torsten Hoefler, Arun Chauhan, and Andrew Lumsdaine. *Kanor: A Declarative Language for Explicit Communication. In Proceedings of the Thirteenth International Symposium on the Practical Aspects of Declarative Languages (PADL), 2011. Held in conjunction with the ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL).*

Kanor for Clusters

$$\textbf{@communicate}\ \{\ b@recv\_rank <<= a@send\_rank\ \}$$

$$e_0@e_1 << op << e_2@e_3\ \texttt{where}\ e_4$$

$$e_0@e_1 <<= e_2@e_3\ \texttt{where}\ e_4$$

$$\underbrace{\texttt{A[j]}}_{\text{storage location}}\ @\ \underbrace{\texttt{i}}_{\text{receiver rank}}\ \underbrace{<<=}_{\text{reduction operator}}\ \underbrace{\texttt{B[i]}}_{\text{data}}\ @\ \underbrace{\texttt{j}}_{\text{sender rank}}\ \texttt{where}\ \underbrace{\texttt{i in world}}_{\text{generator}},\ \underbrace{\texttt{j in \{0...i\}}}_{\text{generator}},\ \underbrace{\texttt{i \% 2 == 0}}_{\text{filter}}$$

Eric Holk, William E. Byrd, Jeremiah Willcock, Torsten Hoefler, Arun Chauhan, and Andrew Lumsdaine. *Kanor: A Declarative Language for Explicit Communication. In Proceedings of the Thirteenth International Symposium on the Practical Aspects of Declarative Languages (PADL), 2011. Held in conjunction with the ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL).*

Kanor for Clusters

$$\textbf{@communicate}\ \{\ b@recv\_rank <<= a@send\_rank\ \}$$

$$e_0@e_1 << op << e_2@e_3\ \texttt{where}\ e_4$$

$$e_0@e_1 <<= e_2@e_3\ \texttt{where}\ e_4$$

$$\underbrace{\texttt{A[j]}}_{\substack{\textit{storage}\\ \textit{location}}} @ \underbrace{\texttt{i}}_{\substack{\textit{receiver}\\ \textit{rank}}}\ \underbrace{\texttt{<<=}}_{\substack{\textit{reduction}\\ \textit{operator}}}\ \underbrace{\texttt{B[i]}}_{\textit{data}} @ \underbrace{\texttt{j}}_{\substack{\textit{sender}\\ \textit{rank}}}\ \texttt{where}\ \underbrace{\texttt{i in world}}_{\textit{generator}},\ \underbrace{\texttt{j in}\ \{\texttt{0}\ldots\texttt{i}\}}_{\textit{generator}},\ \underbrace{\texttt{i}\ \%\ \texttt{2 == 0}}_{\textit{filter}}$$

$$\downarrow$$

Source-level compiler (using ROSE)

$$\downarrow$$

standard C++ code

Eric Holk, William E. Byrd, Jeremiah Willcock, Torsten Hoefler, Arun Chauhan, and Andrew Lumsdaine. *Kanor: A Declarative Language for Explicit Communication. In Proceedings of the Thirteenth International Symposium on the Practical Aspects of Declarative Languages (PADL), 2011. Held in conjunction with the ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL).*

Distributed Memory Targets

- Generate MPI
- Recognize collectives that map to MPI collectives
- Optimize communication
  - computation-communication overlap
  - communication coalescing

Software Pipelining

Sweep3D

```
for (int i = 0; i < OCTANTS; i++) {
  for (int j = 0; j < ANGLES; j++) {
    // loop though the diagonals, N is the number of processors
    for (int diag = 0; diag < 2 * N + 1; diag++) {
      if ((myid.x + myid.y) == diag) { compute(); } /* wave front */
      @communicate {temp_s@(x, y+1) <<= A[lastrow]@(x, y)
                    where x, y in {0...N-1} and x + y = diag;}
      @communicate {temp_e@(x + 1, y) <<= A[][lastcol]@(x, y)
                    where x, y in {0...N-1} and x + y = diag;}
}}}
```

Software Pipelining

Sweep3D

```
for (int i = 0; i < OCTANTS; i++) {
  for (int j = 0; j < ANGLES; j++) {
    // loop though the diagonals, N is the number of processors
    for (int diag = 0; diag < 2 * N + 1; diag++) {
      if ((myid.x + myid.y) == diag) { compute(); } /* wave front */
      @communicate {temp_s@(x, y+1) <<= A[lastrow]@(x, y)
                    where x, y in {0...N-1} and x + y = diag;}
      @communicate {temp_e@(x + 1, y) <<= A[][lastcol]@(x, y)
                    where x, y in {0...N-1} and x + y = diag;}
}}}
```

Sweep3D pipelined

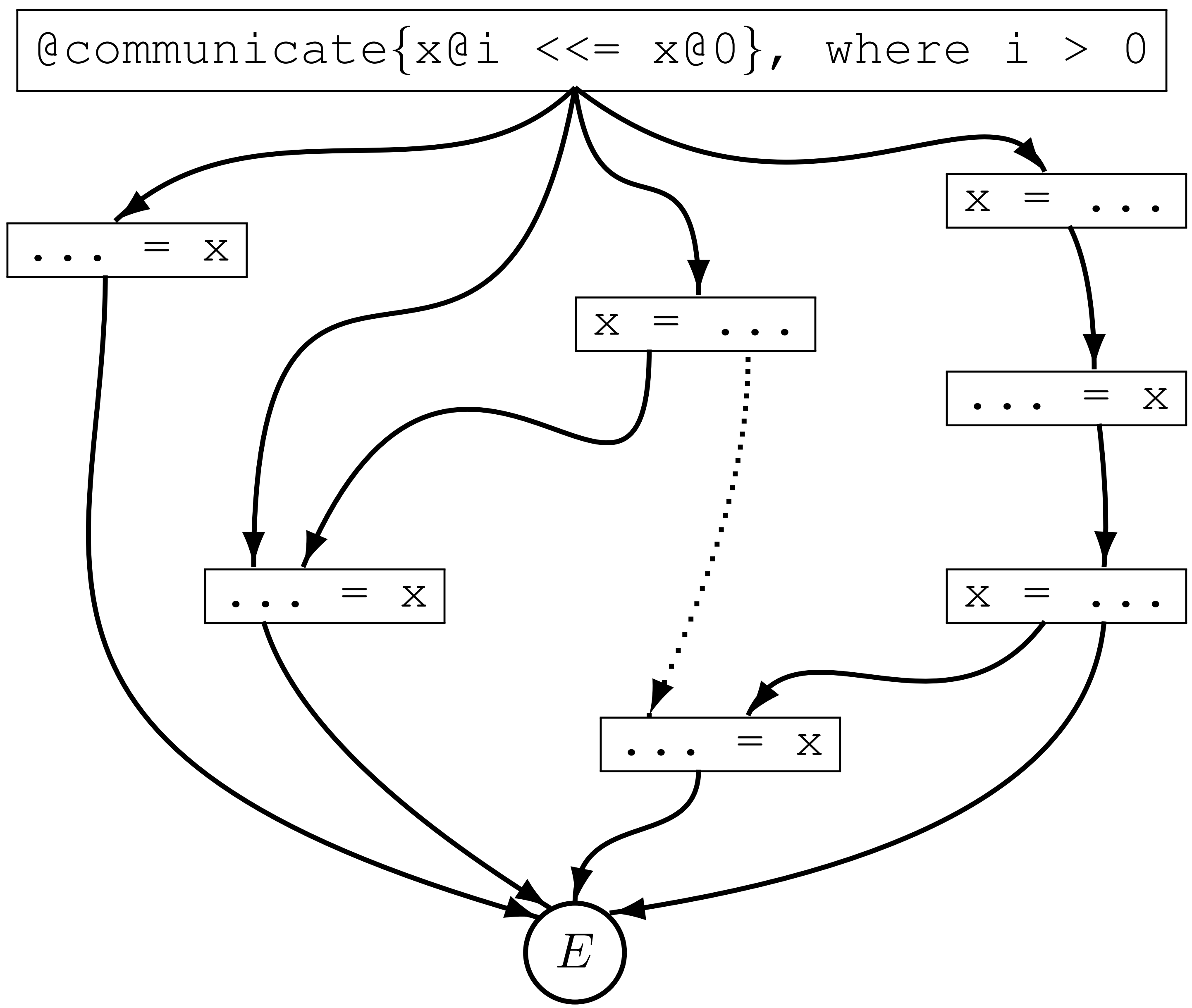
```
for (int i = 0; i < OCTANTS; i++) {
  for (int j = 0; j < ANGLES; j++) {
    for (int s = 0; s < min(SIZE, s + BLOCK_SIZE); s+=BLOCK_SIZE) {
      // loop though the diagonals, N is the number of processors
      for (int diag = 0; diag < 2 * N + 1; diag++) {
        if ((myid.x + myid.y) == diag) { strip_mined_compute(); }
        @communicate {temp_s@(x, y+1) <<= A[lastrow]@(x, y)
                      where x, y in {0...N-1} and x + y = diag;}
        @communicate {temp_e@(x + 1, y) <<= A[][lastcol]@(x, y)
                      where x, y in {0...N-1} and x + y = diag;}
}}}}
```

Shared Memory Targets

- Use partitioned address space
- Leverage shared memory for communication
- Eliminate buffer copying
  - identify opportunities for aliasing
  - insert synchronization for correctness
  - optimize at run time to eliminate synchronization overheads

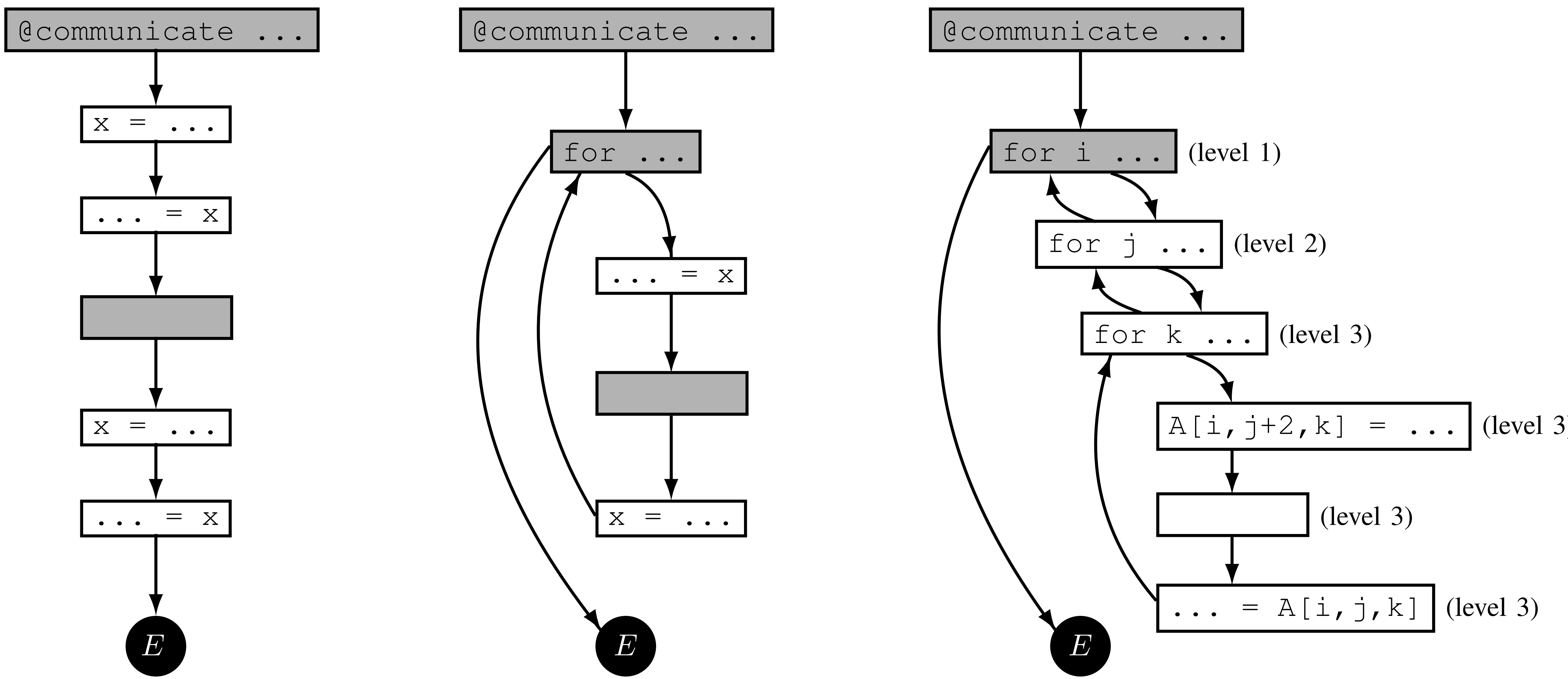
Fangzhou Jiao, Nilesh Mahajan, Jeremiah Willcock, Arun Chauhan, and Andrew Lumsdaine. **Partial Globalization of Partitioned Address Space for Zero-copy Communication with Shared Memory**. In *Proceedings of the 18th International Conference on High Performance Computing (HiPC)*, 2011. *To appear*.

Optimizing for Shared Memory



Fangzhou Jiao, Nilesh Mahajan, Jeremiah Willcock,* Arun Chauhan*, and Andrew Lumsdaine. *Partial Globalization of Partitioned Address Space for Zero-copy Communication with Shared Memory. In Proceedings of the 18th International Conference on High Performance Computing (HiPC), 2011. To appear.*

Subtleties


@communicate ...
x = ...
... = x
x = ...
... = x
E
@communicate ...
for ...
... = x
x = ...
E
@communicate ...
for i ...
(level 1)
for j ...
(level 2)
for k ...
(level 3)
A[i,j+2,k] = ...
(level 3)
(level 3)
... = A[i,j,k]
(level 3)
E

Harlan for GPUs

```
__global__ void add_kernel(int size, float *X, float *Y, float *Z)
{
  int i = threadIdx.x;
  if(i < size) { Z[i] = X[i] + Y[i]; }
}

void vector_add(int size, float *X, float *Y, float *Z)
{
  float *dX, *dY, *dZ;
  cudaMalloc(&dX, size * sizeof(float));
  cudaMalloc(&dY, size * sizeof(float));
  cudaMalloc(&dZ, size * sizeof(float));

  cudaMemcpy(dX, X, size * sizeof(float), cudaMemcpyHostToDevice);
  cudaMemcpy(dY, Y, size * sizeof(float), cudaMemcpyHostToDevice);

  add_kernel<<<1, size>>>(size, dX, dY, dZ);

  cudaMemcpy(Z, dZ, size * sizeof(float), cudaMemcpyDeviceToHost);

  cudaFree(dX);
  cudaFree(dY);
  cudaFree(dZ);
}
```

Harlan for GPUs

```
__global__ void add_kernel(int size, float *X, float *Y, float *Z)
{
  int i = threadIdx.x;
  if(i < size) { Z[i] = X[i] + Y[i]; }
}

void vector_add(int size, float *X, float *Y, float *Z)
{
  float *dX, *dY, *dZ;
  cudaMalloc(&dX, size * sizeof(float));
  cudaMalloc(&dY, size * sizeof(float));
  cudaMalloc(&dZ, size * sizeof(float));

  cudaMemcpy(dX, X, size * sizeof(float), cudaMemcpyHostToDevice);
  cudaMemcpy(dY, Y, size * sizeof(float), cudaMemcpyHostToDevice);

  add_kernel<<<1, size>>>(size, dX, dY, dZ);

  cudaMemcpy(Z, dZ, size * sizeof(float), cudaMemcpyDeviceToHost);

  cudaFree(dX);
  cudaFree(dY);
  cudaFree(dZ);
}
```

```
void vector_add (vector<float> X, vector <float> Y, vector<float> Z)
{
  kernel (x : X, y : Y, z : Z) { z = x + y; };
}
```

Harlan Features

Reductions

```
z = +/kernel (x : X, y : Y) { x * y };
```

Harlan Features

Reductions

```
z = +/kernel (x : X, y : Y) { x * y };
```

Asynchronous kernels

```
handle = async kernel (x : X, y : Y) { x * y };
// other concurrent kernels of program code here
z = +/wait(handle);
```

Harlan Features

Reductions

```
z = +/kernel (x : X, y : Y) { x * y };
```

Asynchronous kernels

```
handle = async kernel (x : X, y : Y) { x * y };
// other concurrent kernels of program code here
z = +/wait(handle);
```

Nested kernels

```
total = +/kernel (row : Rows) { +/kernel (x : row); };
```

Example 1: Dot Product

```
// dot product of two vectors
double dotproduct(Vector X, Vector Y) {
    double dot = +/kernel(x : X, y : Y) { x * y };
    return dot;
}
```

Example 2: Dense Matrix Multiply

```
// dense matrix-matrix multiply
Matrix matmul (Matrix A, Matrix B) {
    // this block does a transpose; it could go in a library
    Bt = kernel(j : [0 .. length(B[0])]) {
        kernel(i : [0 .. length(B)]) {
            B[j][i];
        }
    };
    C = kernel(row : A) {
        kernel(col : Bt) {
            +/kernel(a : row, b : col) {
                a * b;
            }
        }
    }
    return C;
}
```

Example 3: Sparse Mat-Vec Product

```
// sparse matrix-vector product (CSR)
Vector spmv(CSR_i Ai, CSR_v Av, Vector X) {
    Vector Y = kernel(is : Ai, vs : Av) {
        +/kernel(i : is, v : vs) { v * X[i]; }
    };
    return Y;
}
```

Combining Kanor and Harlan

```
kernel (x : X, y : Y, z : Z) { z = x * y; }
@communicate {
  Y[i]@r <<= Z[i]@((r+1) & NUM_NODES)
  where r in world,
        i in 0...length(Y)
}
kernel (x : X, y : Y, z : Z) { z = x * y; }
```

What about Joe programmers?

Automatic parallelization

"The reports of my death are highly exaggerated"

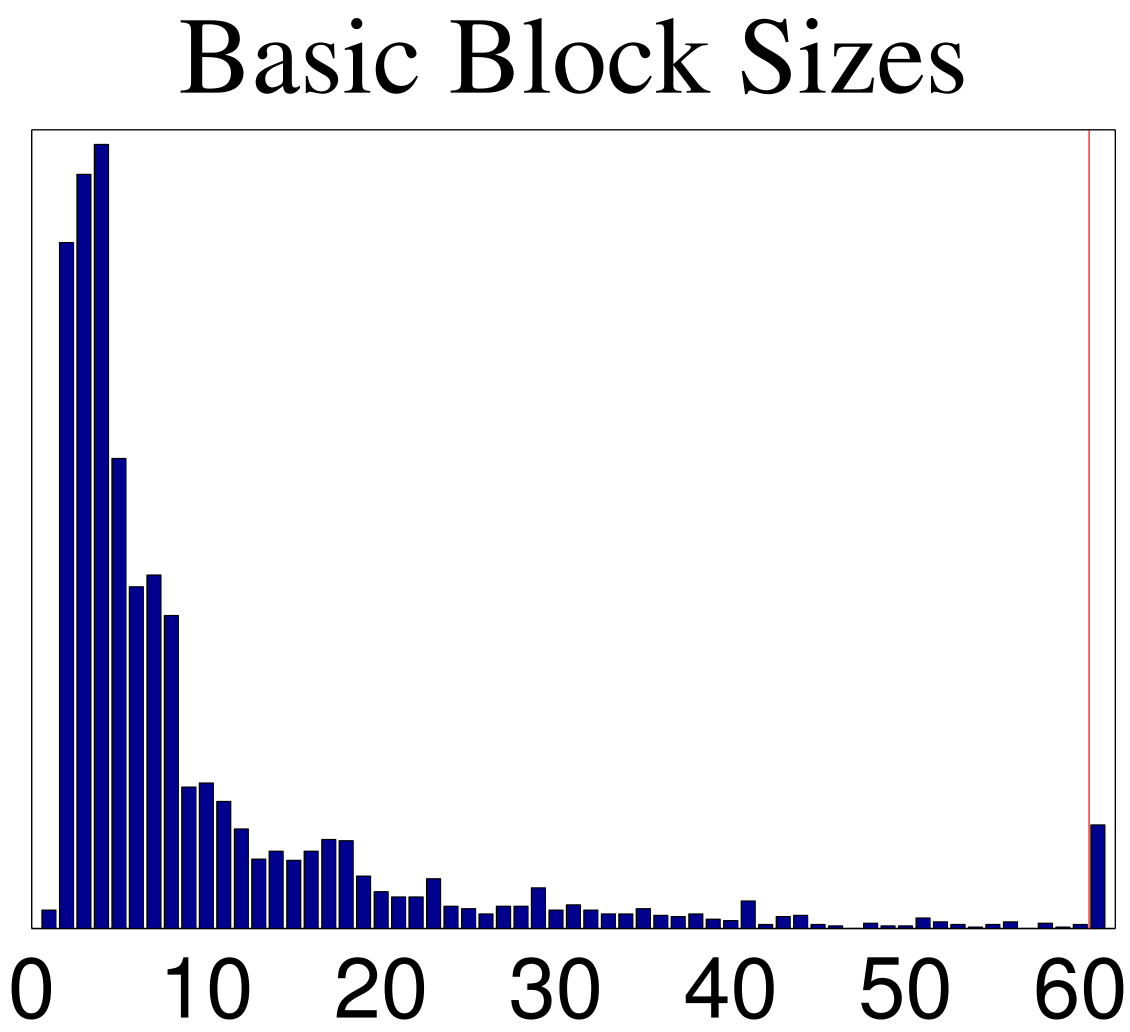
- MATLAB is the *lingua franca* of scientists and engineers
- Joe programmers would rather write in 10 minutes and let the program run for 24 hours, than vice versa
- They would still like their programs to run in 10 minutes!

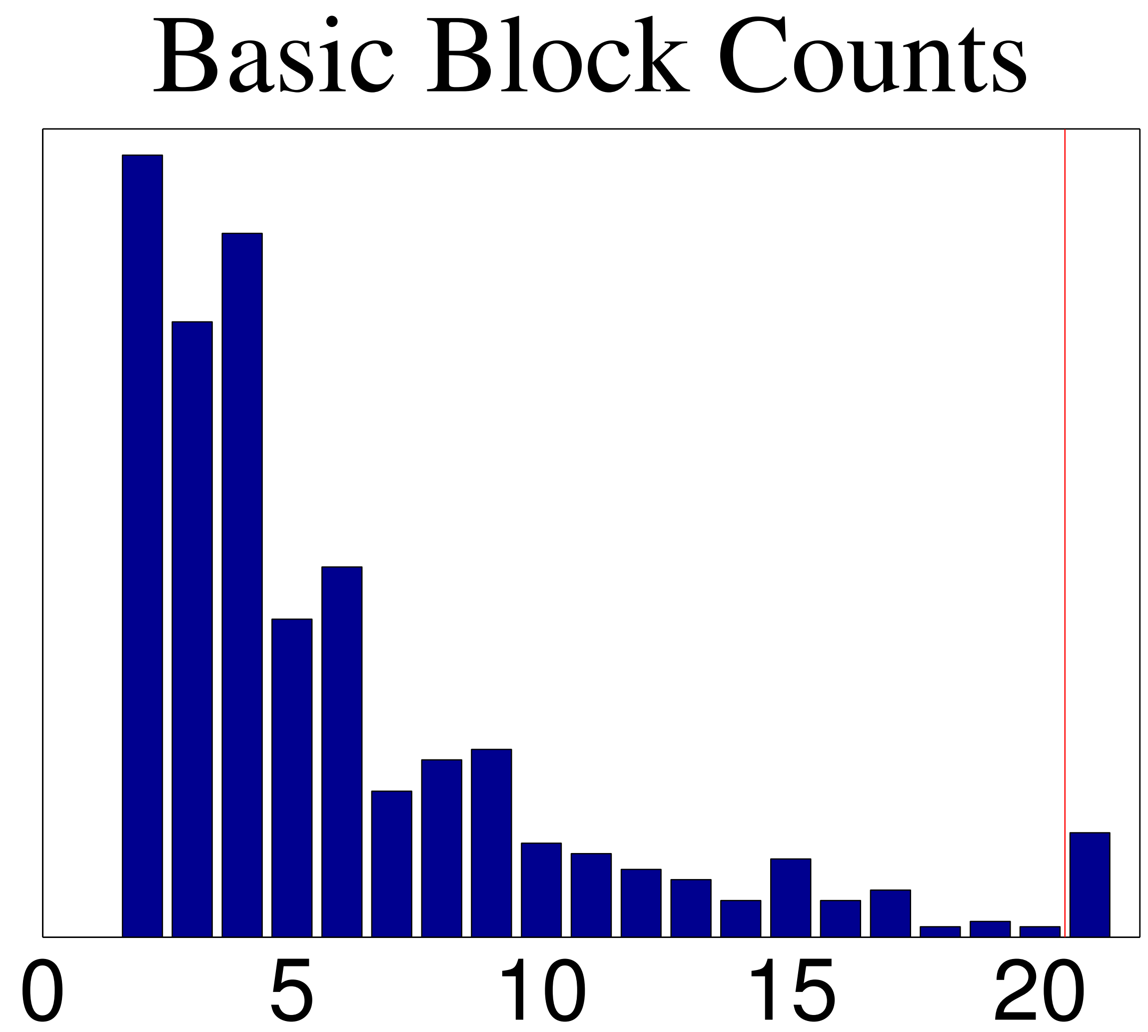
Parallelism in MATLAB

- Built-in `parallel-for` (with the parallel computing toolbox)
- Third party libraries to offload computations on clusters
- Third party and MathWorks libraries to offload computation on GPUs
  - "declare" variables to be of GPU type

```
A = GPUdouble(a);
B = GPUdouble(b);
C = A*B;
```

MATLAB: Empirical Study





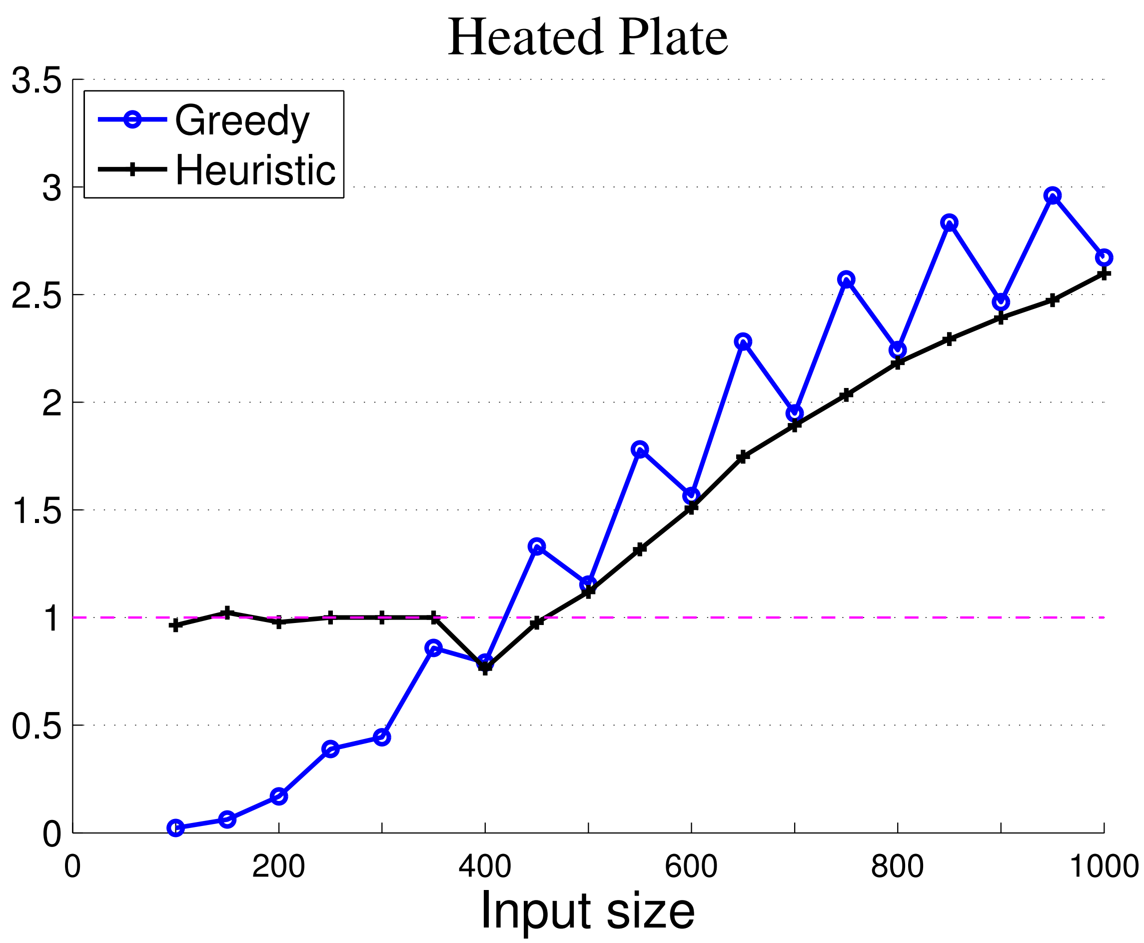
Automatic GPU Computation

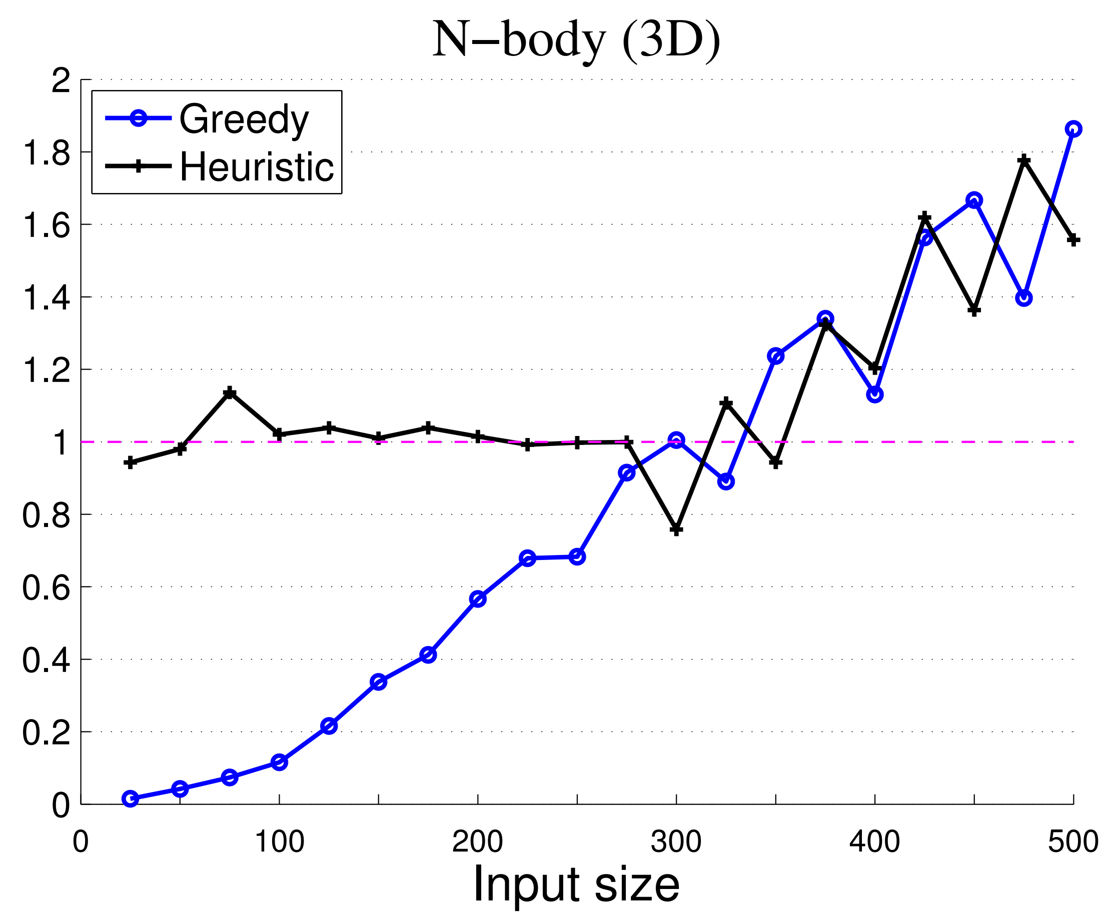
- Model the computation
  - cost model for CPU times
  - cost model for GPU times
  - cost model for CPU-GPU data transfer
- Solve a binary integer linear programming problem

$$
\begin{aligned}
\text{Minimize} \quad & \vec{f}'\vec{x} \\
\text{such that} \quad & \mathbf{A}\vec{x} \leq \vec{b} \\
\text{and} \quad & \mathbf{A}_{\text{eq}}\vec{x} = \vec{b}_{\text{eq}}
\end{aligned}
$$

Chun-Yu Shei, Pushkar Ratnalikar, and Arun Chauhan*. Automating GPU Computing in MATLAB. In Proceedings of the 2011 International Conference on Supercomputing (ICS), 2011.*

Experimental Results





Serious Joe programmer?

Speculative Parallelism

- Write mostly sequential code
- Insert code to mark "possibly parallel" regions
- Speculator + verifier
  - we support multiple concurrent verifiers to support nested speculation

Chen Ding, Xipeng Shen, Kirk Kelsey, Chris Tice, Ruke Huang, and Chengliang Zhang. *Software Behavior Oriented Parallelization. In Proceedings of the 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), pages 223–234, 2007.*

Scalable Speculative Parallelism on Clusters

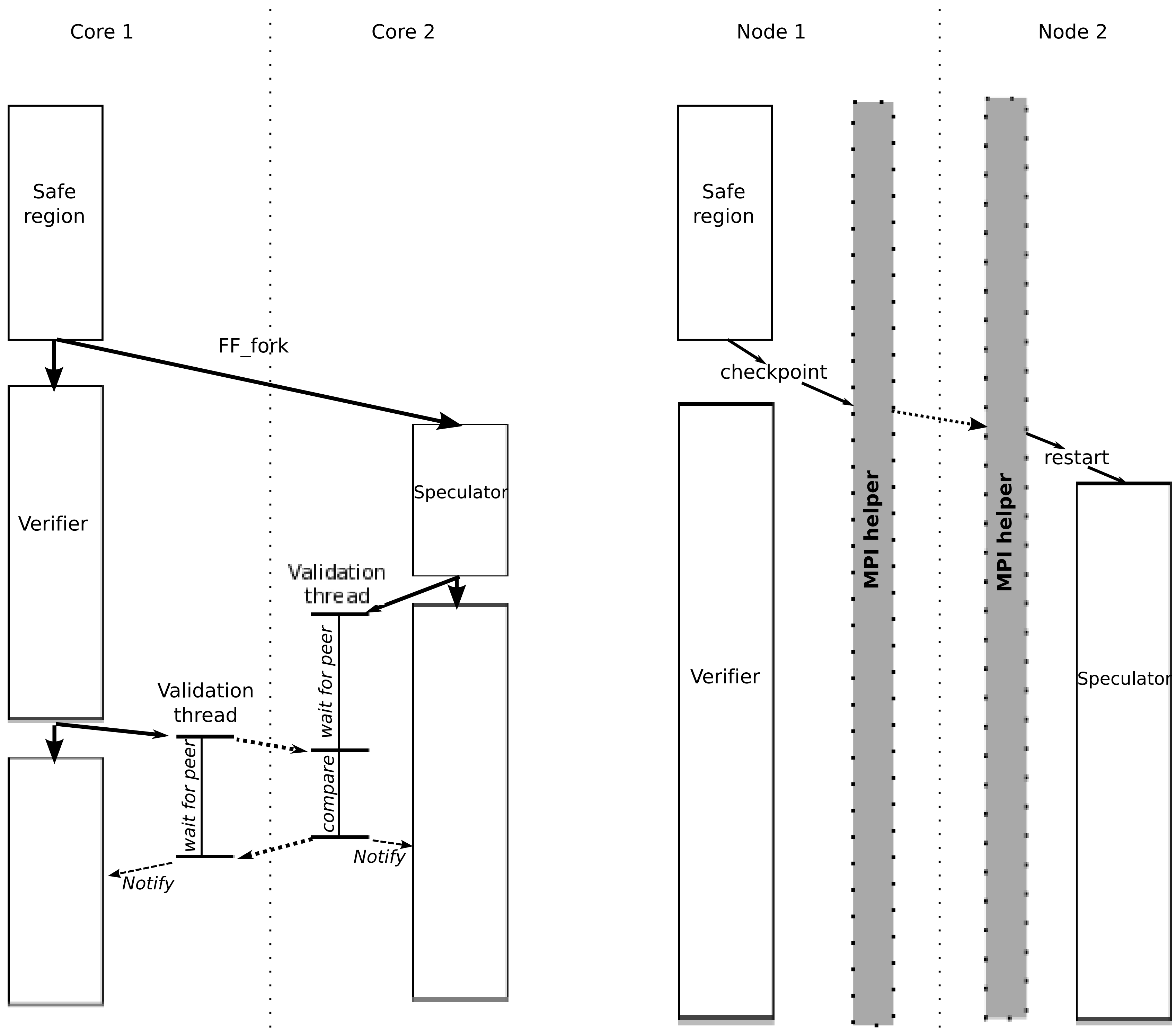
```
// safe code

// code where speculation possible (code region A)

// safe code

// code where speculation possible (code regions B)
```

$\Downarrow$

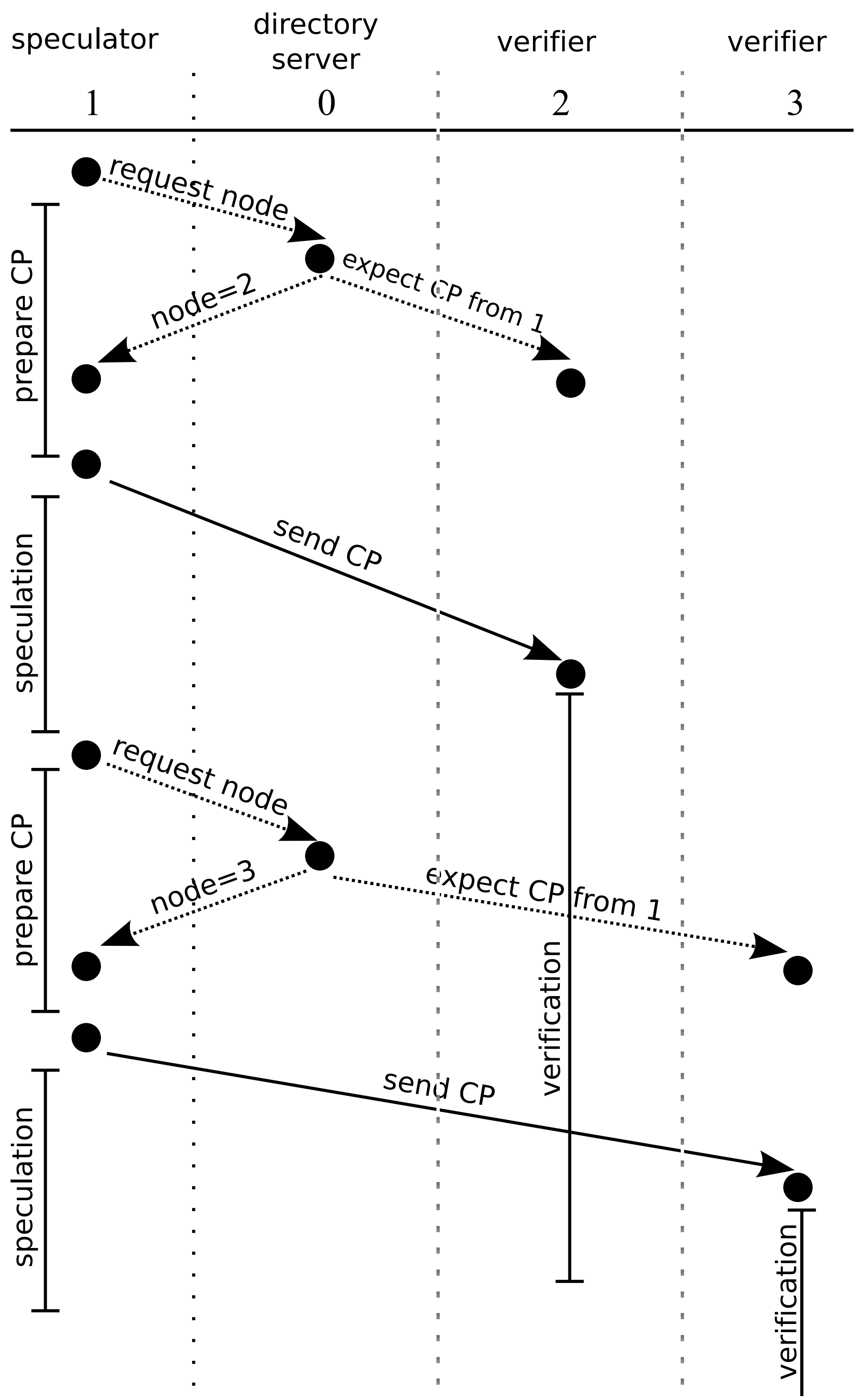
```
FF_init();

// safe code

if (FF_fork() == FF_VERIFIER) {
  // safe version of the code region A
} else { // FF_SPECULATOR
  // unsafe version of the code region A
}
FF_create_validation_thread();

// safe code

if (FF_fork() == FF_VERIFIER) {
  // safe version of the code region B
} else { // FF_SPECULATOR
  // unsafe version of the code region B
}
FF_create_validation_thread();
```

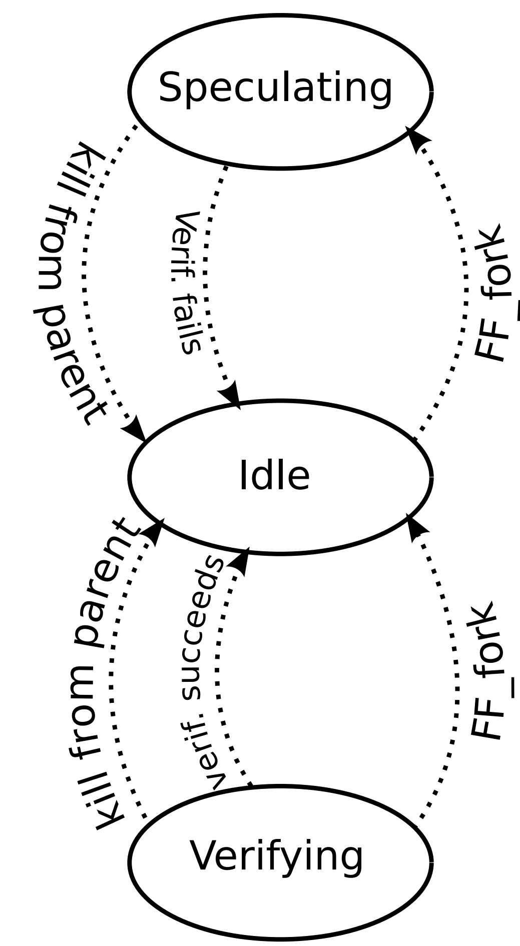
Devarshi Ghoshal, Sreesudhan R Ramkumar, and Arun Chauhan. *Distributed Speculative Parallelization using Checkpoint Restart. In Proceedings of the International Conference on Computational Science (ICCS), 2011*

Intra- and Inter-Node Speculation



Implementing Inter-Node Speculation





Analysis

T = time of execution of original program
p = probability that speculation succeeds
k = number of simultaneous speculations
s = speedup of speculatively parallelized code over the original sequential code
S = overall speedup of the program

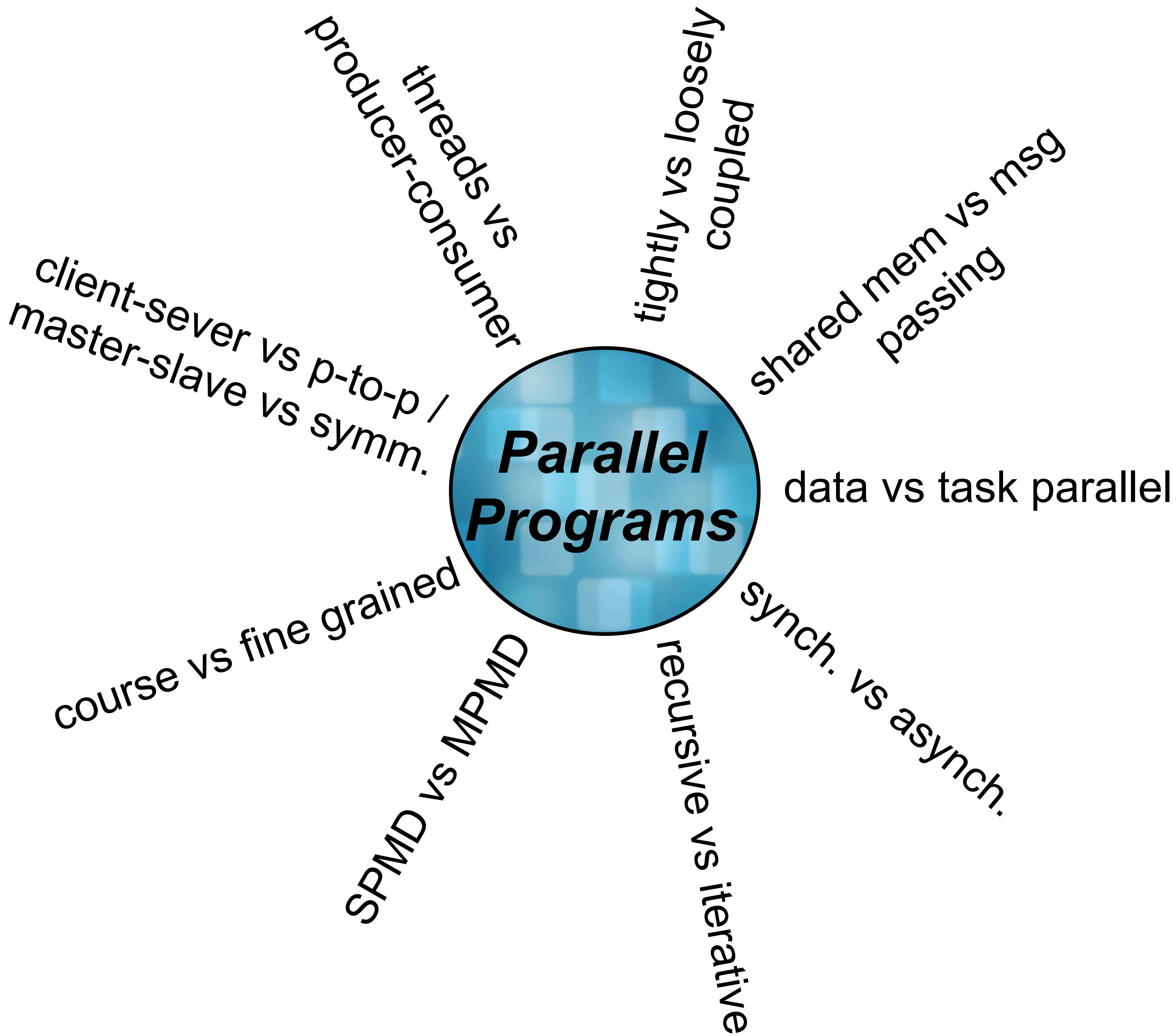
Running time of code, with speculation = $T + pk\dfrac{T}{s} + (1-p)kT$

Overall speedup, $S = \dfrac{T(k+1)}{T + pk\frac{T}{s} + (1-p)kT} = \dfrac{k+1}{k+1+pk(\frac{1}{s}-1)}$

S ≤ k+1 (strict upper bound, as s → ∞)

What next?

The Maze of Parallel Programming



Concluding Remarks

- There is no silver bullet of parallel programming (and there may never be)
- Tool (compiler developers, OS developers, architects) need to recognize the different needs of parallel programmers
- Parallel programming needs to become an integrated core of computer science education
    - every future programmer is a parallel programmer

Questions?

http://www.cs.indiana.edu/~achauhan