Effective (Parallel) Programming for the Masses

Optimizing High-level Languages

Arun Chauhan
School of Informatics and Computing
Indiana University, Bloomington

Purdue University
October 28, 2011

Computing as a Fundamental Science

"Computing is as fundamental as the physical, life, and social sciences."

Peter J. Denning and Paul S. Rosenbloom
Communications of the ACM, Sep 2009

"What our community should really aim for is the development of a curriculum that turns our subject into the fourth R—as in 'rogramming—of our education systems.

…

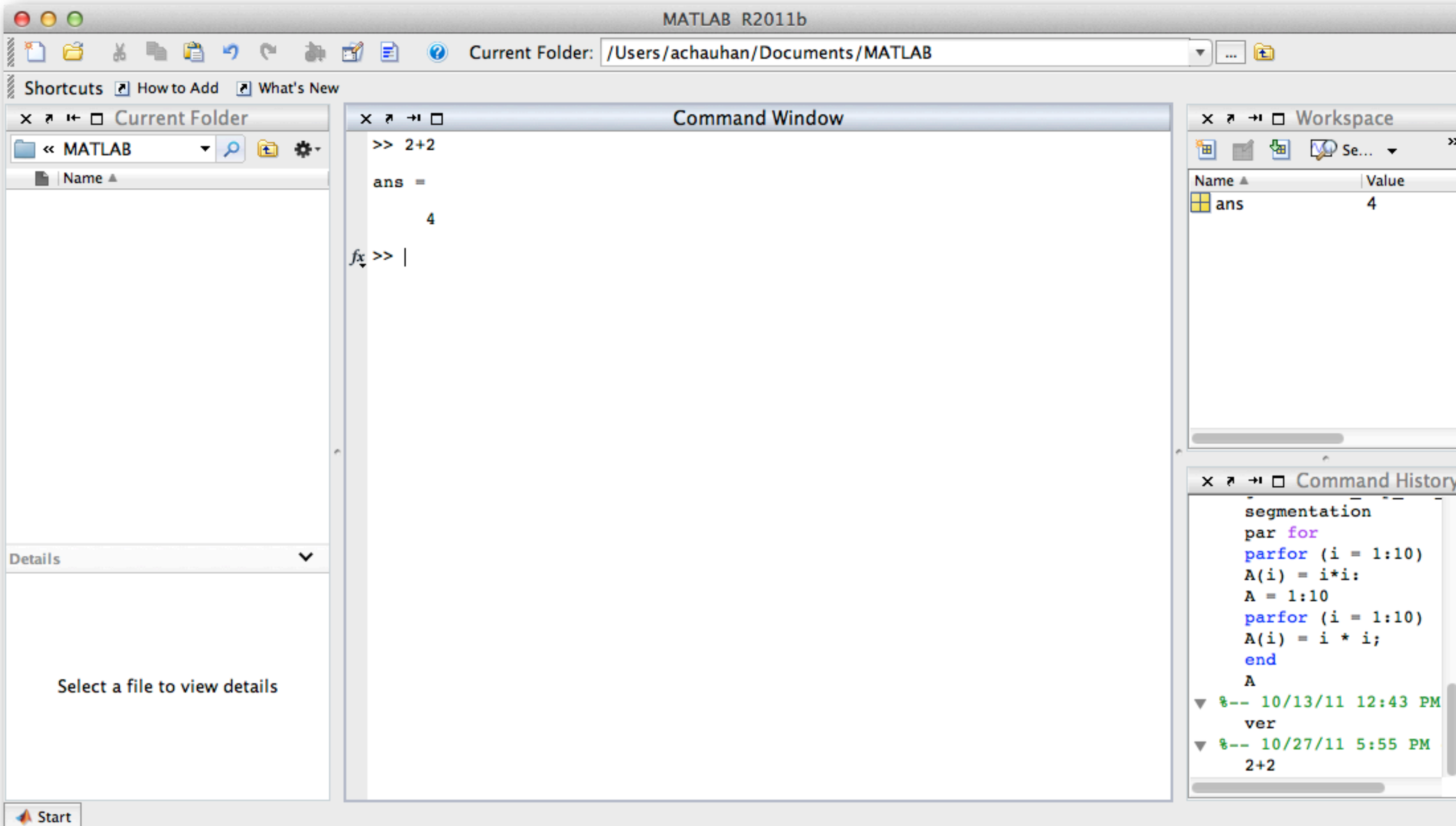
A form of mathematics can be used as a full- fledged programming language, just like Turing Machines."

Matthias Felleisen and Shriram Krishnamurthy
Communications of the ACM, Jul 2009

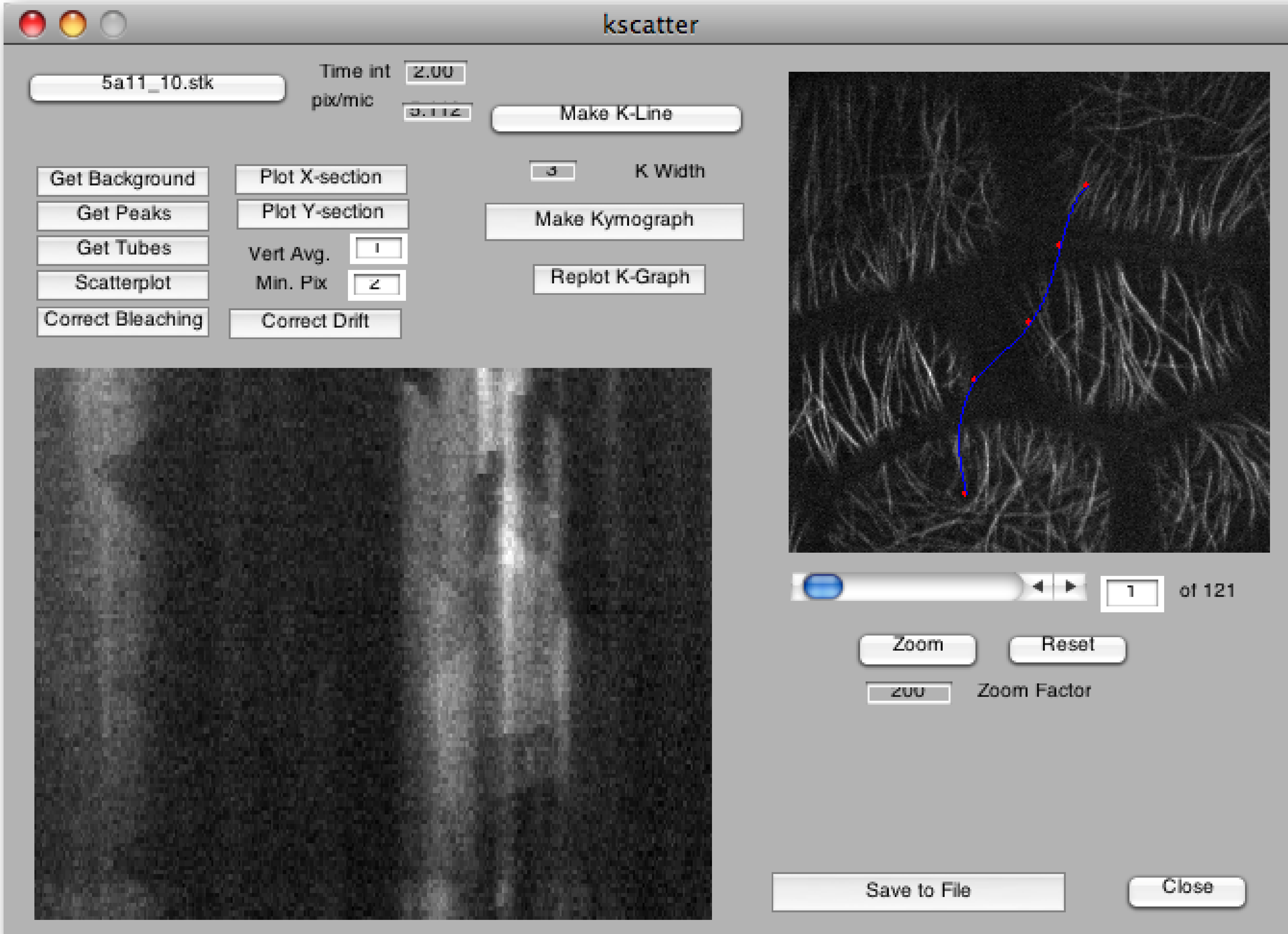
Computing is Inexpensive

> "I would rather spend 10 minutes coding and letting the program run overnight, than spend weeks writing and debugging to be able to run the program in 10 minutes."
>
> *A DSP researcher in EE*

MATLAB


MATLAB R2011b
Current Folder: /Users/achauhan/Documents/MATLAB
Shortcuts
How to Add
What's New
Current Folder
MATLAB
Name
Details
Select a file to view details
Command Window
>> 2+2
ans =
4
>>
Workspace
Name
Value
ans
4
Command History
segmentation
par for
parfor (i = 1:10)
A(i) = i*i:
A = 1:10
parfor (i = 1:10)
A(i) = i * i;
end
A
%-- 10/13/11 12:43 PM
ver
%-- 10/27/11 5:55 PM
2+2
Start

MATLAB: Ease of Use



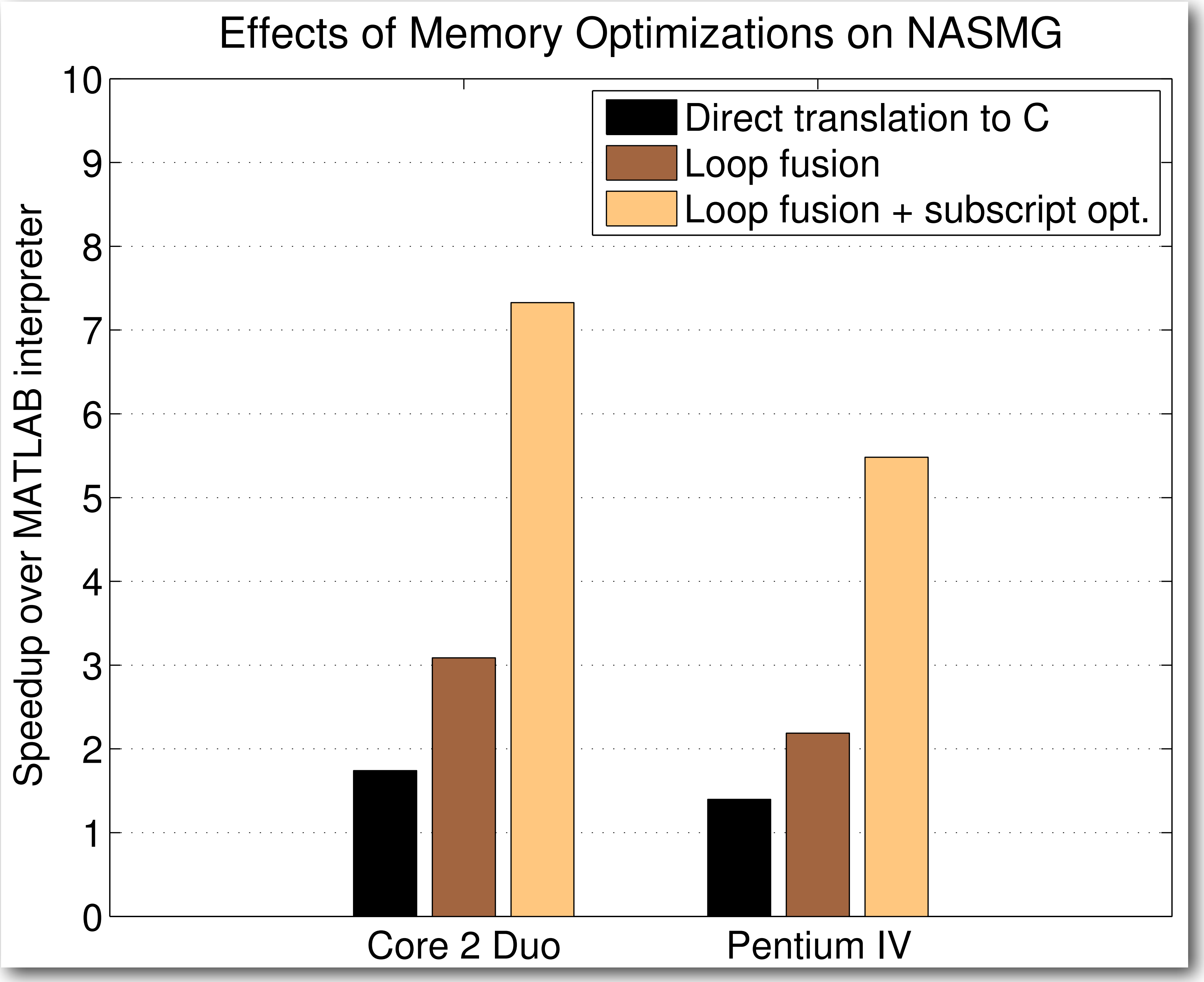
MATLAB in a Nutshell

- C-like syntax
  - x = 1+1;
  - y = 2*x + 100;
- Array operations
  - C = A*B;
  - C = A .* B;
- IF, FOR, WHILE, SWITCH statements

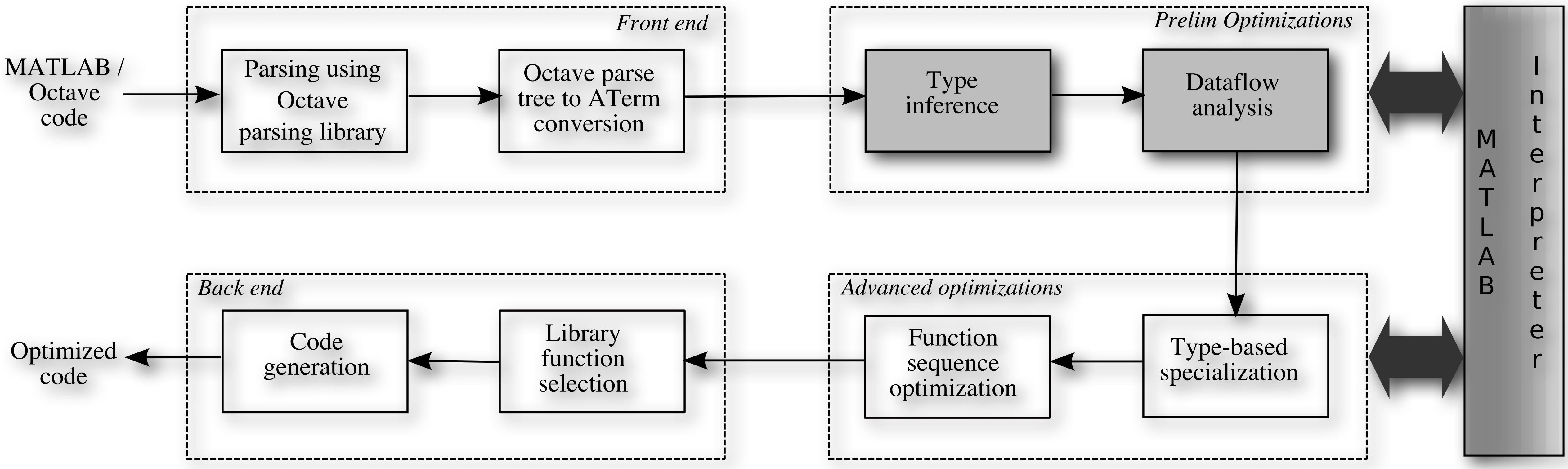
Motivation: NASMG

```
m = f(1).*(n(c,c,c))
  + f(2).*(n(c,c,u)+n(c,c,d)
          +n(c,u,c)+n(c,d,c)
          +n(u,c,c)+n(d,c,c))
  + f(3).*(n(c,u,u)+n(c,u,d)
          +n(c,d,u)+n(c,d,d)
          +n(u,c,u)+n(u,c,d)
          +n(d,c,u)+n(d,c,d)
          +n(u,u,c)+n(u,d,c)
          +n(d,u,c)+n(d,d,c))
  + f(4).*(n(u,u,u)+n(u,u,d)
          +n(u,d,u)+n(u,d,d)
          +n(d,u,u)+n(d,u,d)
          +n(d,d,u)+n(d,d,d));
```

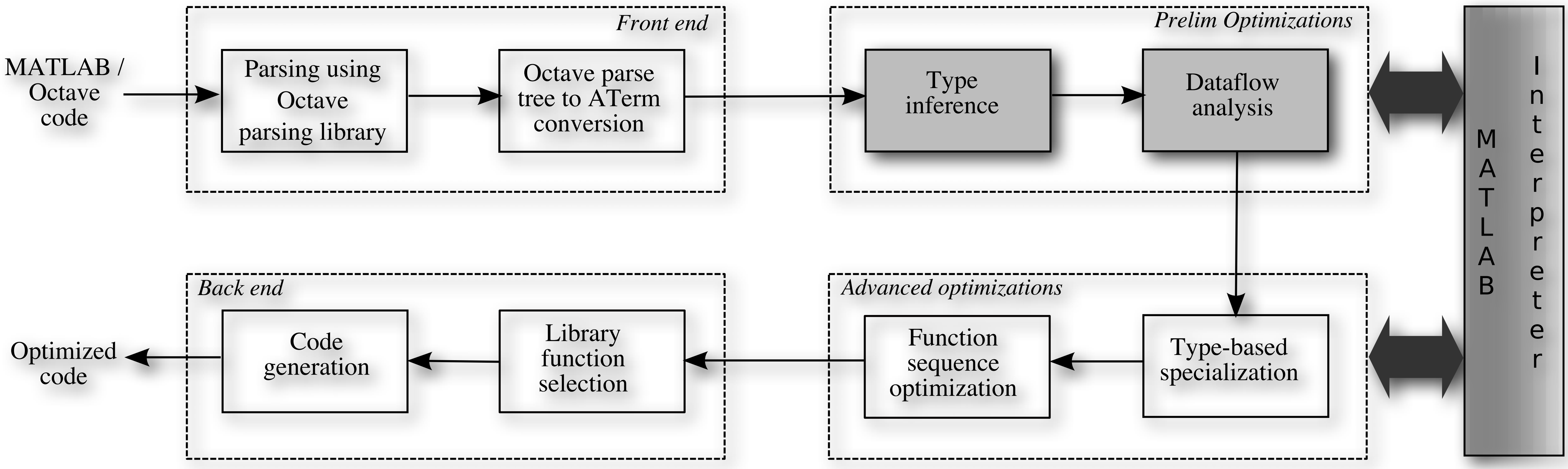
Optimization Potential



MATLAB / Octave Compiler



MATLAB / Octave Compiler



Infrastructure written in Ruby
Uses our own embedded DSL called RubyWrite

Basic Compilation Issues

- Dynamic types
  - infer types to enable translation to lower-level (statically typed) language
- Dynamic dispatch
  - specialize for static dispatch
  - use types info. to specialize based on input types
- High-level operators
  - intelligently map to underlying libraries

Chun-Yu Shei, Arun Chauhan, and Sidney Shaw. *Compile-time Disambiguation of MATLAB Types through Concrete Interpretation with Automatic Run-time Fallback. In Proceedings of the 2009 International Conference on High Performance Computing (HiPC), 2009.*

Basic Compilation Issues

- Dynamic types
  - infer types to enable translation to lower-level (statically typed) language
- Dynamic dispatch
  - specialize for static dispatch
  - use types info. to specialize based on input types
- High-level operators
  - intelligently map to underlying libraries

Chun-Yu Shei, Arun Chauhan, and Sidney Shaw. *Compile-time Disambiguation of MATLAB Types through Concrete Interpretation with Automatic Run-time Fallback. In Proceedings of the 2009 International Conference on High Performance Computing (HiPC), 2009.*

MATLAB Type Inference: Past Efforts

- As a data flow problem
  - abstract interpretation to propagate types
  - can be combined with constant propagation
  - not easy to handle complex library functions
- As a set-theoretic problem
  - need external symbolic analysis tool (e.g., Mathematica)
- As constraint equations over sets
  - could be too loosely constrained

Leverage MATAB Interpreter (1)

```
x = 10;
y = 20;
z = x + y;
```

Leverage MATAB Interpreter (1)

```
x = 10;

y = 20;

z = x + y;
```

Leverage MATAB Interpreter (1)

```
BT_x = 'i';
x = 10;


y = 20;


z = x + y;
```

Leverage MATAB Interpreter (1)

```
BT_x = 'i';
x = 10;
BT_y = 'i';
y = 20;

z = x + y;
```

Leverage MATAB Interpreter (1)

```
BT_x = 'i';
x = 10;
BT_y = 'i';
y = 20;
BT_z = BXF_sum(BT_x,BT_y);
z = x + y;
```

Leverage MATAB Interpreter (2)

```
x = 10.5;

y = [1, 2; 3, 4];

y = x*y + a;
```

Leverage MATAB Interpreter (2)

```
x = 10.5;
y = [1, 2; 3, 4];
t = x*y;
y = t + a;
```

Leverage MATAB Interpreter (2)

```
x$1 = 10.5;
y$1 = [1, 2; 3, 4];
t$1 = x$1*y$1;
y$2 = t$1 + a$1;
```

Leverage MATAB Interpreter (2)

```
x$1 = 10.5;



y$1 = [1, 2; 3, 4];

t$1 = x$1*y$1;

y$2 = t$1 + a$1;
```

Leverage MATAB Interpreter (2)

```
BT_x$1 = 'd';
x$1 = 10.5;



y$1 = [1, 2; 3, 4];

t$1 = x$1*y$1;

y$2 = t$1 + a$1;
```

Leverage MATAB Interpreter (2)

```
BT_x$1 = ‘d’;
x$1 = 10.5;
BT_y$1 = BXF_vertcat( …
           BXF_horzcat(‘i’,’i’),…
           BXF_horzcat(‘i’,’i’)…
         );
y$1 = [1, 2; 3, 4];

t$1 = x$1*y$1;

y$2 = t$1 + a$1;
```

Leverage MATAB Interpreter (2)

```
BT_x$1 = ‘d’;
x$1 = 10.5;
BT_y$1 = BXF_vertcat( …
            BXF_horzcat(‘i’,’i’),…
            BXF_horzcat(‘i’,’i’)…
           );
y$1 = [1, 2; 3, 4];
BT_t$1 = BXF_product(BT_x$1,BT_y$1);
t$1 = x$1*y$1;

y$2 = t$1 + a$1;
```

Leverage MATAB Interpreter (2)

```
BT_x$1 = ‘d’;
x$1 = 10.5;
BT_y$1 = BXF_vertcat( …
            BXF_horzcat(‘i’,’i’),…
            BXF_horzcat(‘i’,’i’)…
           );
y$1 = [1, 2; 3, 4];
BT_t$1 = BXF_product(BT_x$1,BT_y$1);
t$1 = x$1*y$1;
BT_y$2 = BXF_sum(BT_t$1,BT_a$1);
y$2 = t$1 + a$1;
```

Leverage MATAB Interpreter (2)

```
x$1 = 10.5;
y$1 = [1, 2; 3, 4];
t$1 = x$1*y$1;
BT_y$2 = BXF_sum(BT_t$1,BT_a$1);
y$2 = t$1 + a$1;
```

Leverage MATAB Interpreter (3)

```
if x < 0
  y = 1.5;
else
  y = 2;
end
```

Leverage MATAB Interpreter (3)

```
if x$1 < 0
  y$1 = 1.5;
else
  y$2 = 2;
end
y$3 = φ(y$1,y$2)
```

Leverage MATAB Interpreter (3)

```
if x$1 < 0

  y$1 = 1.5;
else

  y$2 = 2;
end

y$3 = φ(y$1,y$2)
```

Leverage MATAB Interpreter (3)

```
if x$1 < 0
  BT_y$1 = ‘d’;
  y$1 = 1.5;
else
  BT_y$2 = ‘i’;
  y$2 = 2;
end
BT_y$2 = BTMAX(BT_y$1,BT_y$2);
y$3 = φ(y$1,y$2)
```

Leverage MATAB Interpreter (3)

```
if x$1 < 0
  BT_y$1 = 'd';
  y$1 = 1.5;
else
  BT_y$2 = 'i';
  y$2 = 2;
end
BT_y$2 = BTMAX(BT_y$1,BT_y$2);
y$3 = φ(y$1,y$2)
```

Inference Steps

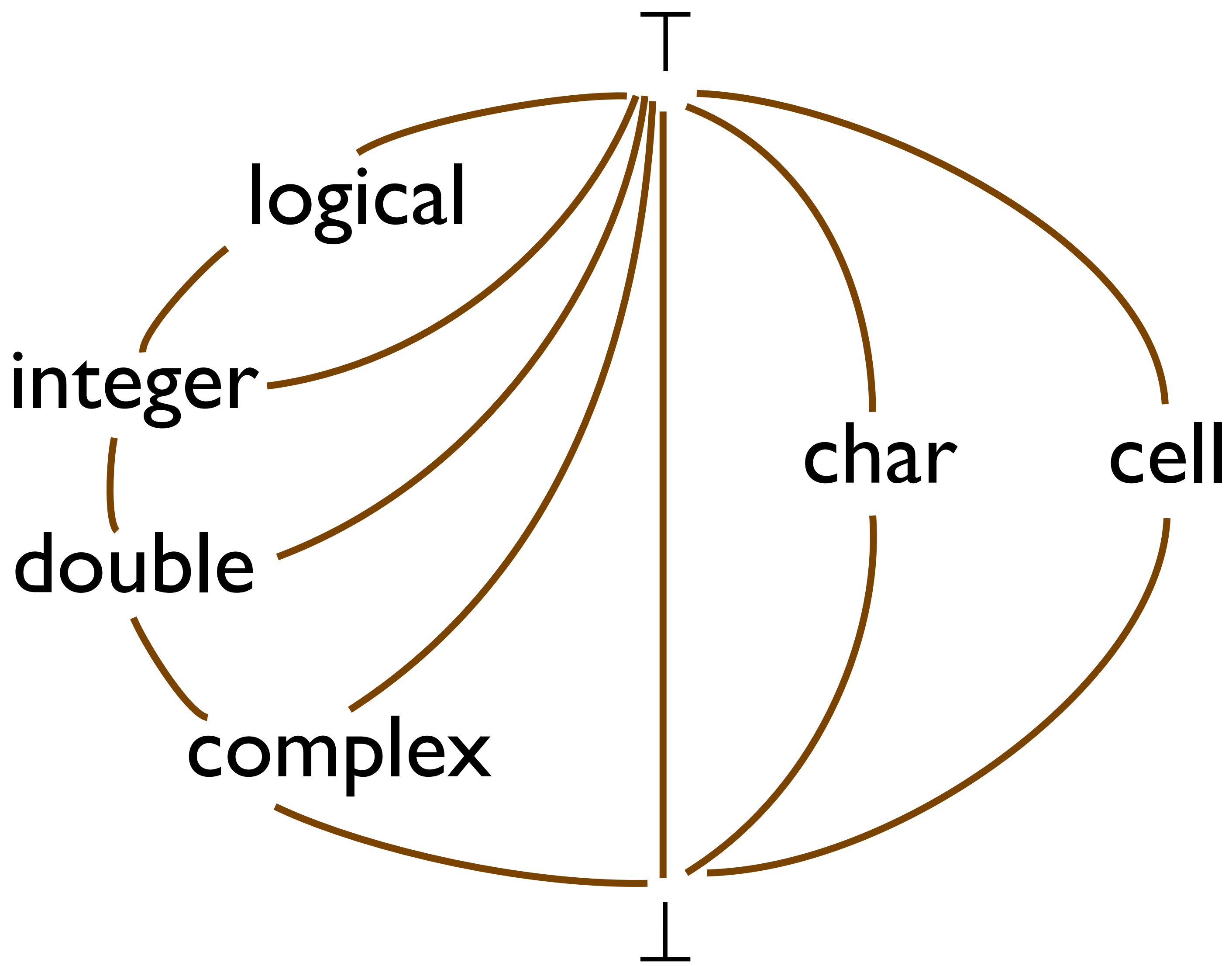
- For each statement of the form $\rho = f(\alpha)$, insert a statement $\rho_T = f_{BXF}(\alpha_T)$
- Perform concrete partial evaluation
- Perform dead-code elimination
  - leaves those type computations that are used for run time optimization

Inference Steps

- For each statement of the form $\rho = f(\alpha)$, insert a statement $\rho_T = f_{BXF}(\alpha_T)$
- Perform concrete partial evaluation
- Perform dead-code elimination
  - leaves those type computations that are used for run time optimization

Need to do a bit more for loops (details in the paper)

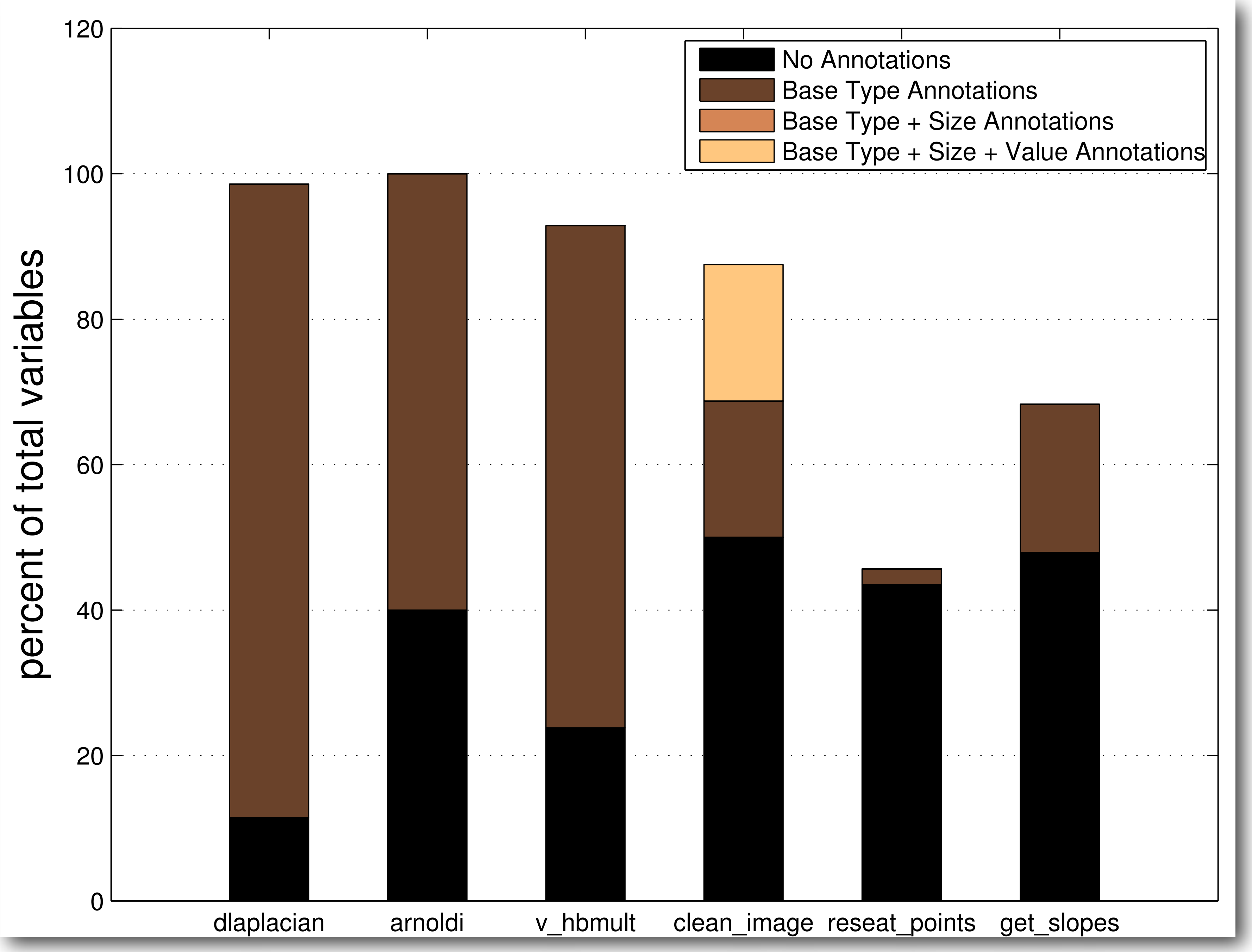
Base Type Lattice



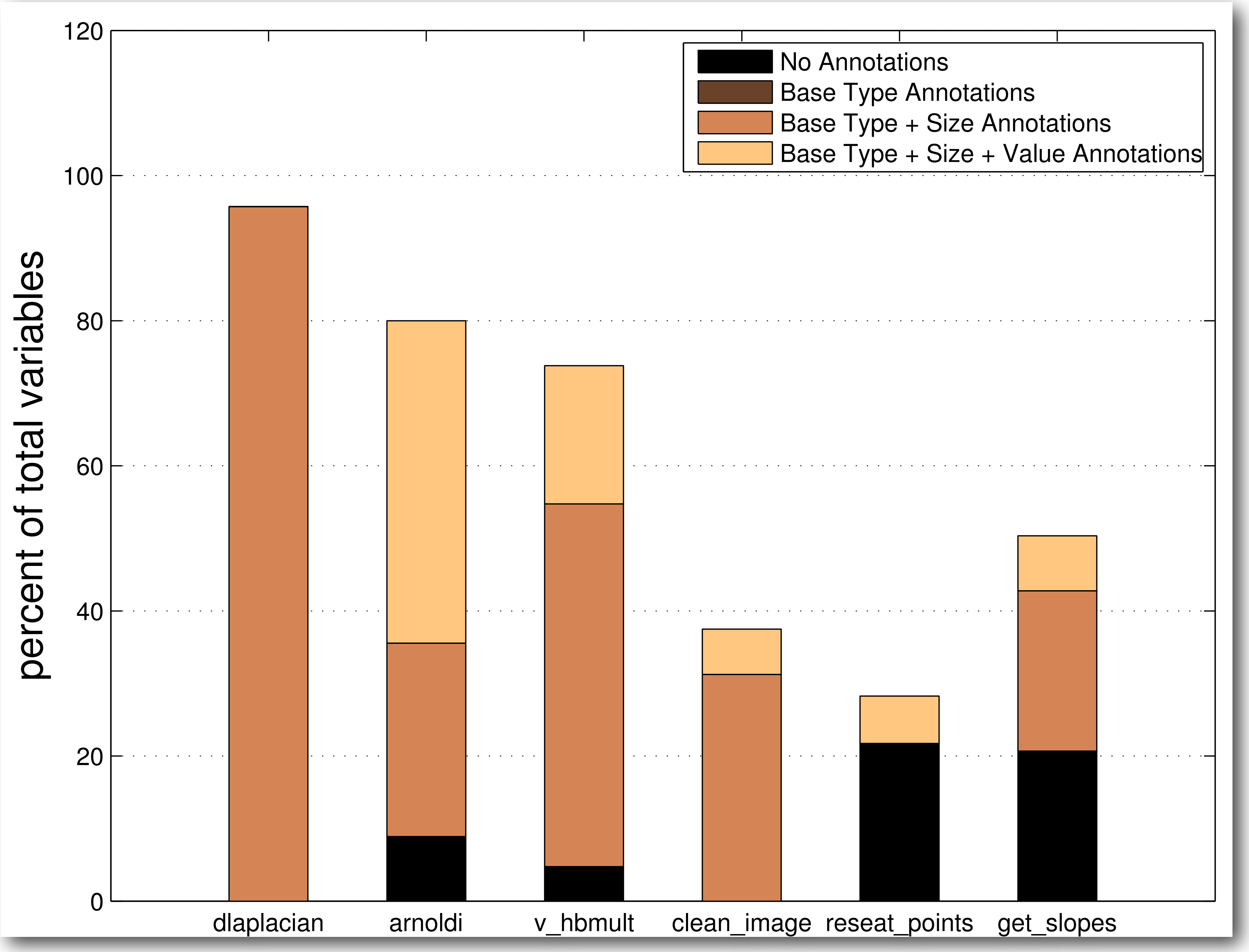
Other Type Inference Issues

- Struct types
  - each field can be considered a separate variable
- Procedures with side-effects
  - output types cannot be computed if that involves executing a slice of the original procedure with side-effects
- Recursive procedures
  - can be handled with a fixed-point evaluation

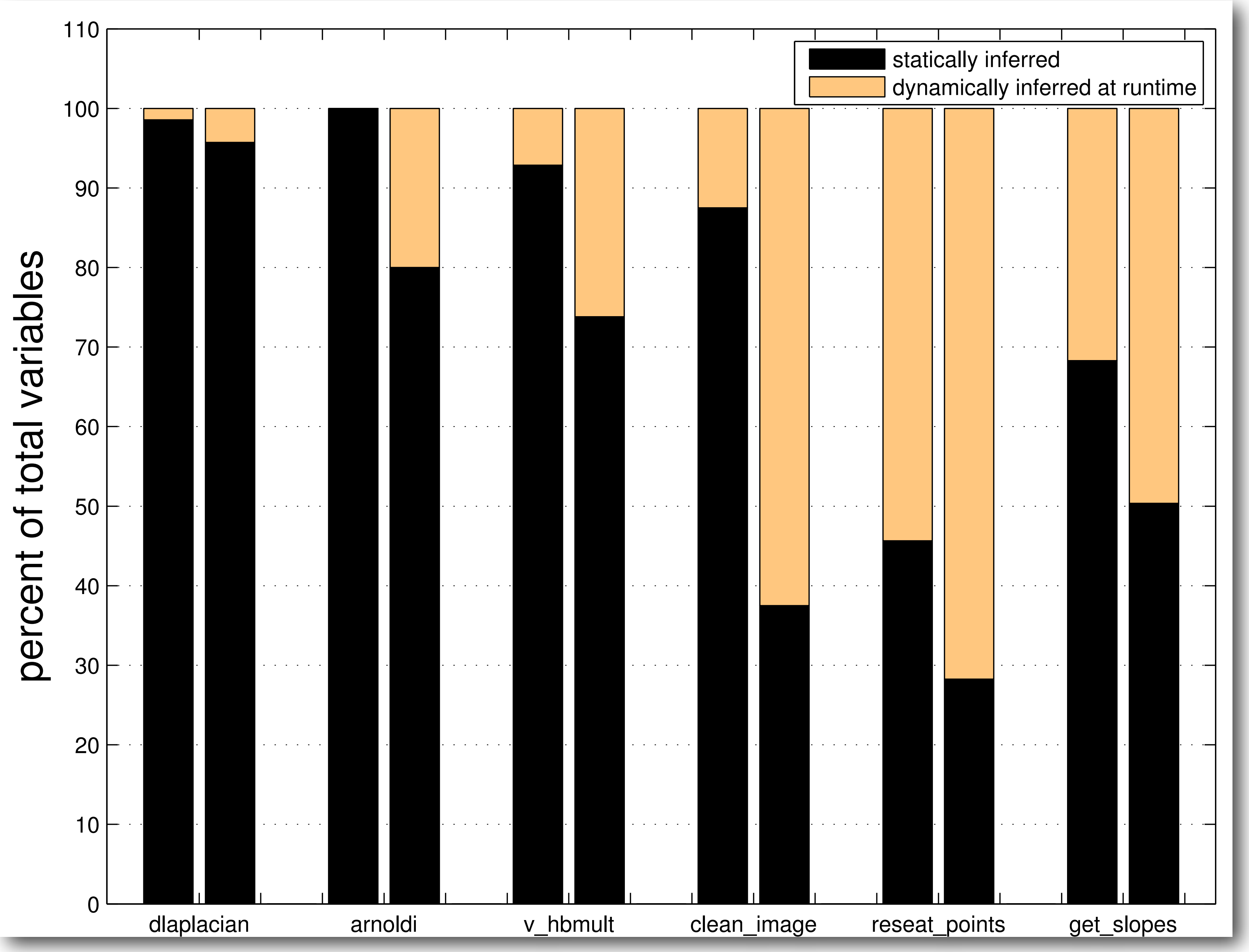
Evaluation: Precision (Base)



Evaluation: Precision (Size)



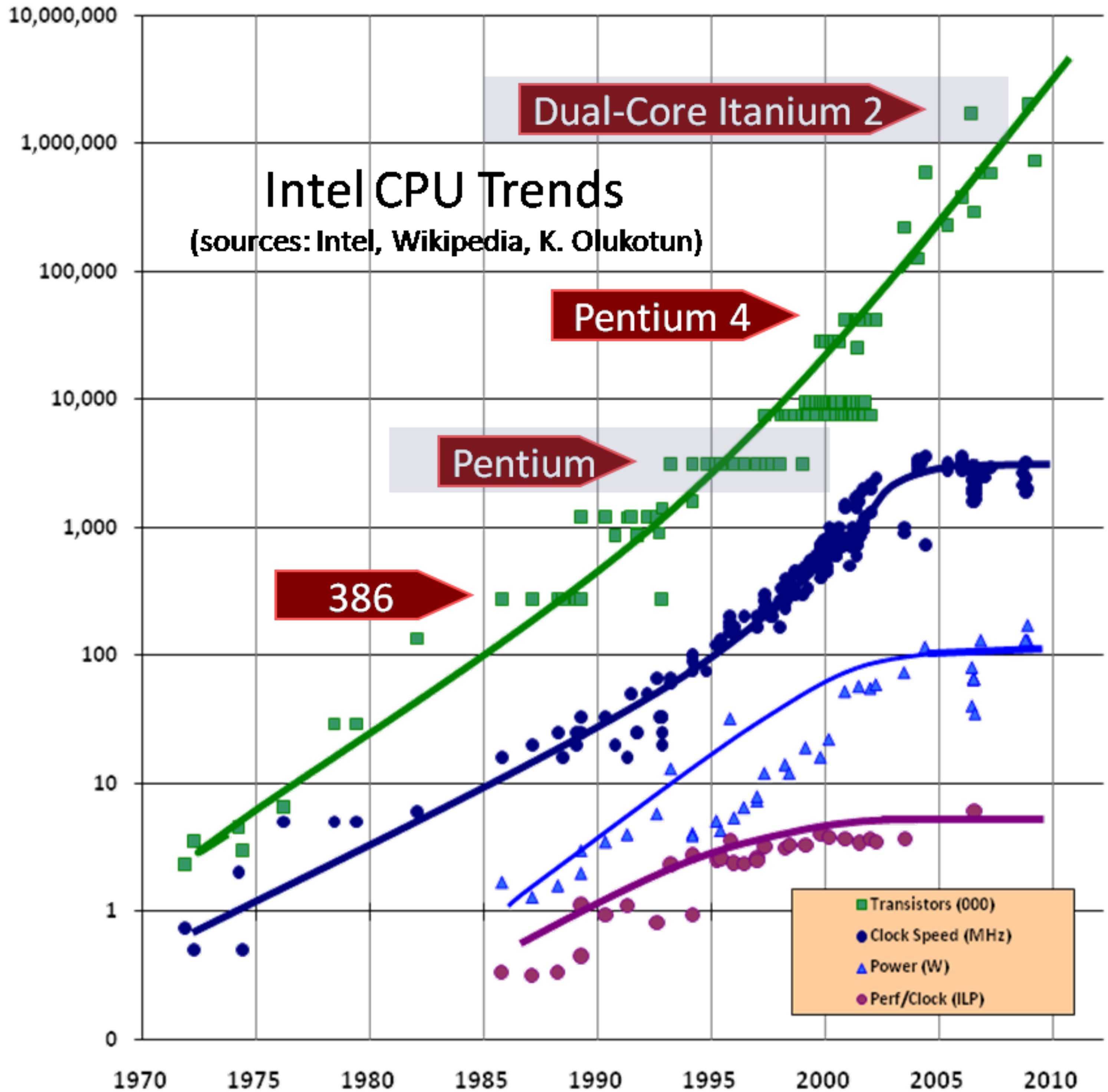
Evaluation: Static vs Dynamic



Observations

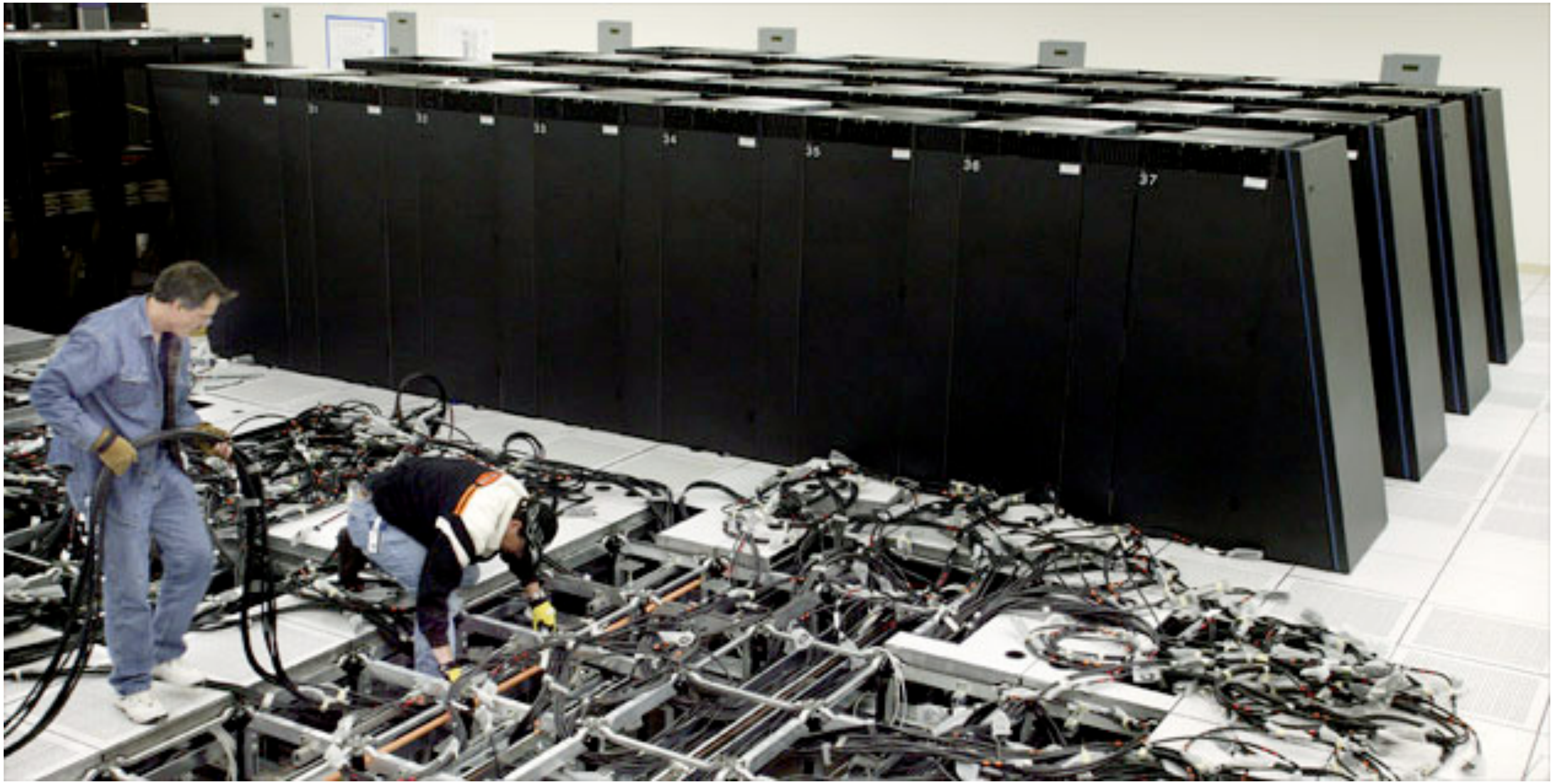
- Advantages of concrete interpretation
  - maintains semantic fidelity for languages defined by their interpreters
  - protects against language changes
  - avoids duplication of effort
- Solving other problems
  - can be seen as an alternative to traditional data flow analysis, for certain problems

The Free Lunch is Over

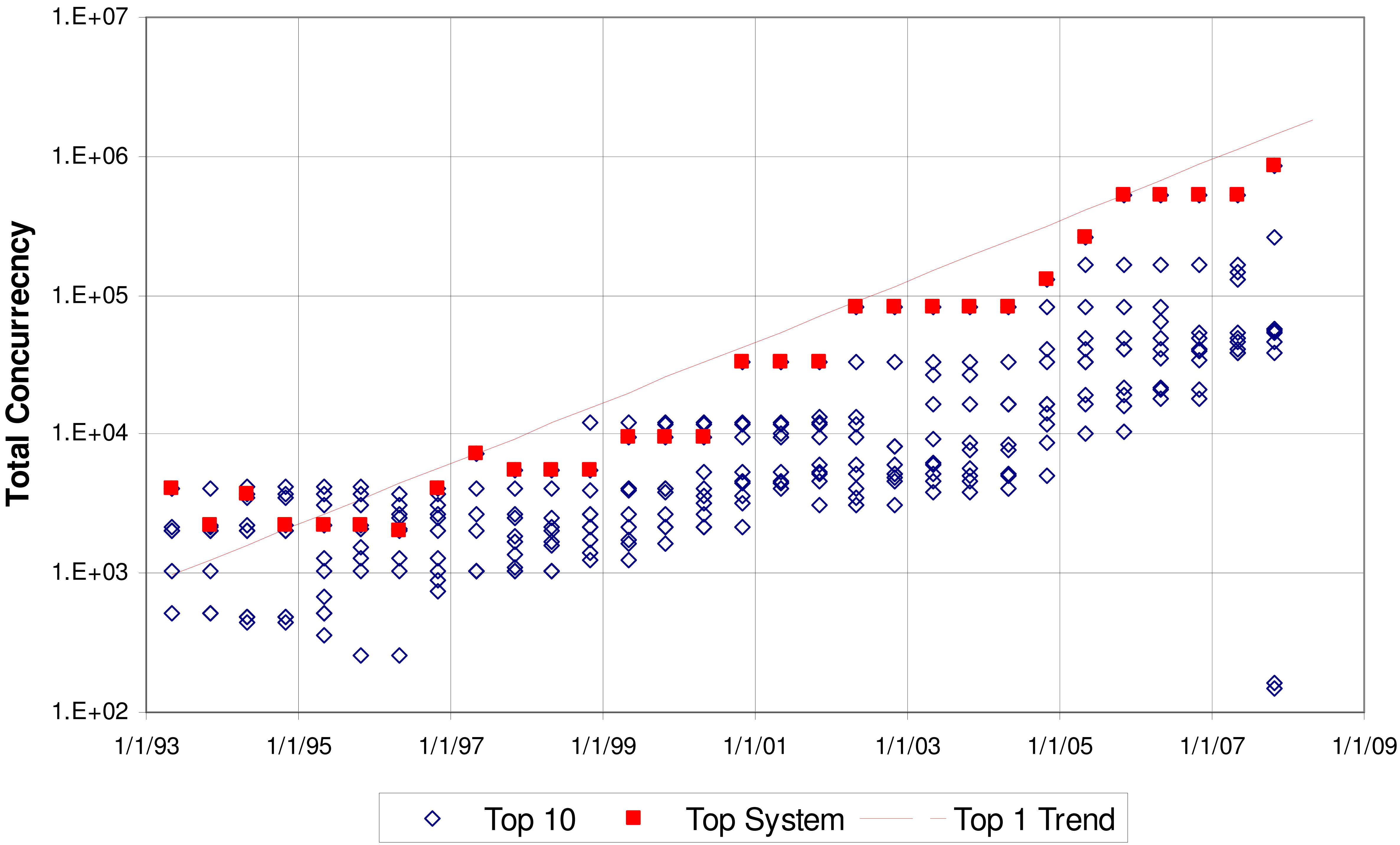


Herb Sutter, *The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software, Dr. Dobb's Journal, 30(3), March 2005*

Exa-scale Challenge



Trends in Concurrency

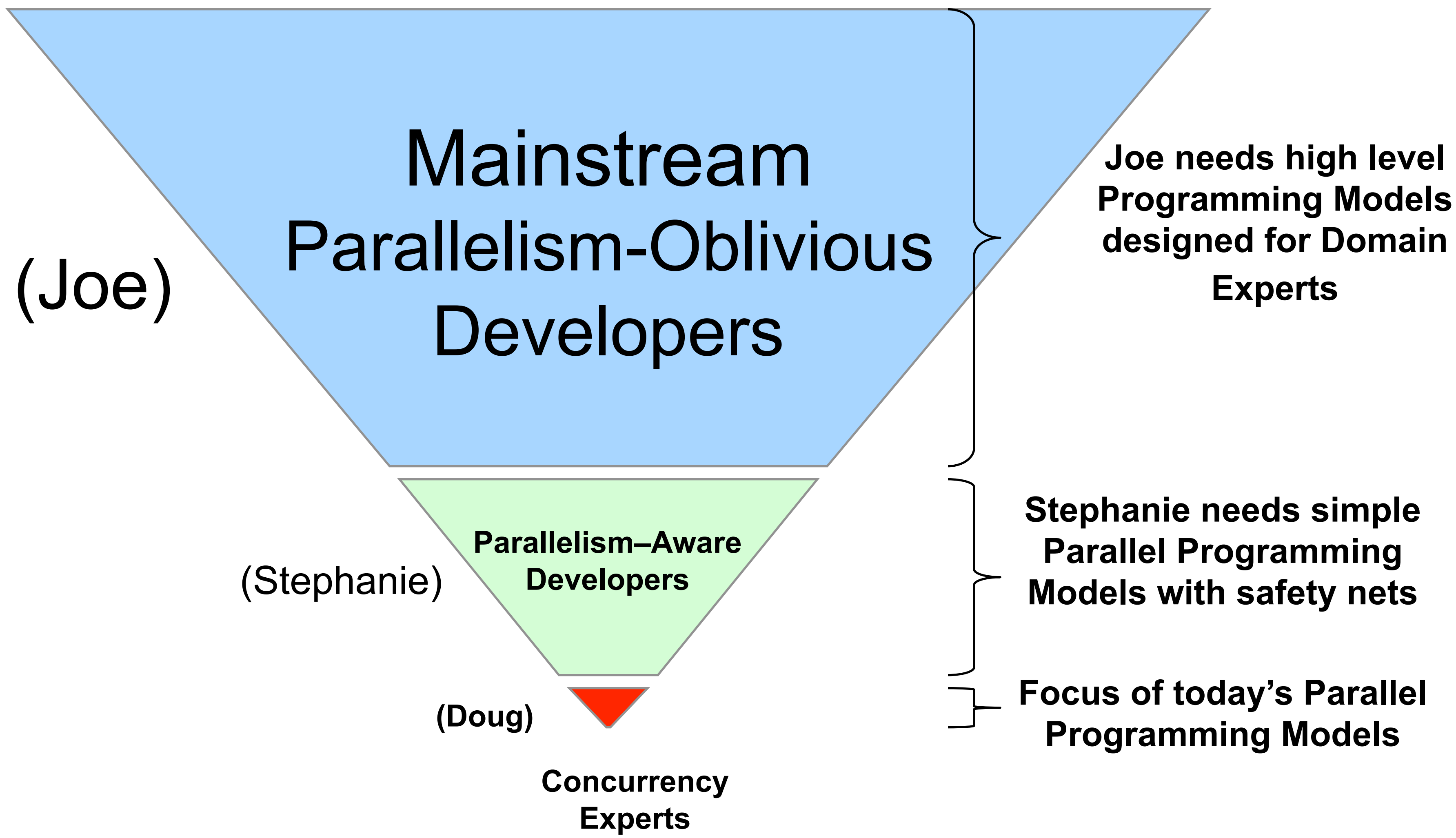


Peter Kogge *et al. Exascale Computing Study, Technology Challenges in Achieving Exascale Systems, 2008.*

Long History of Parallelism

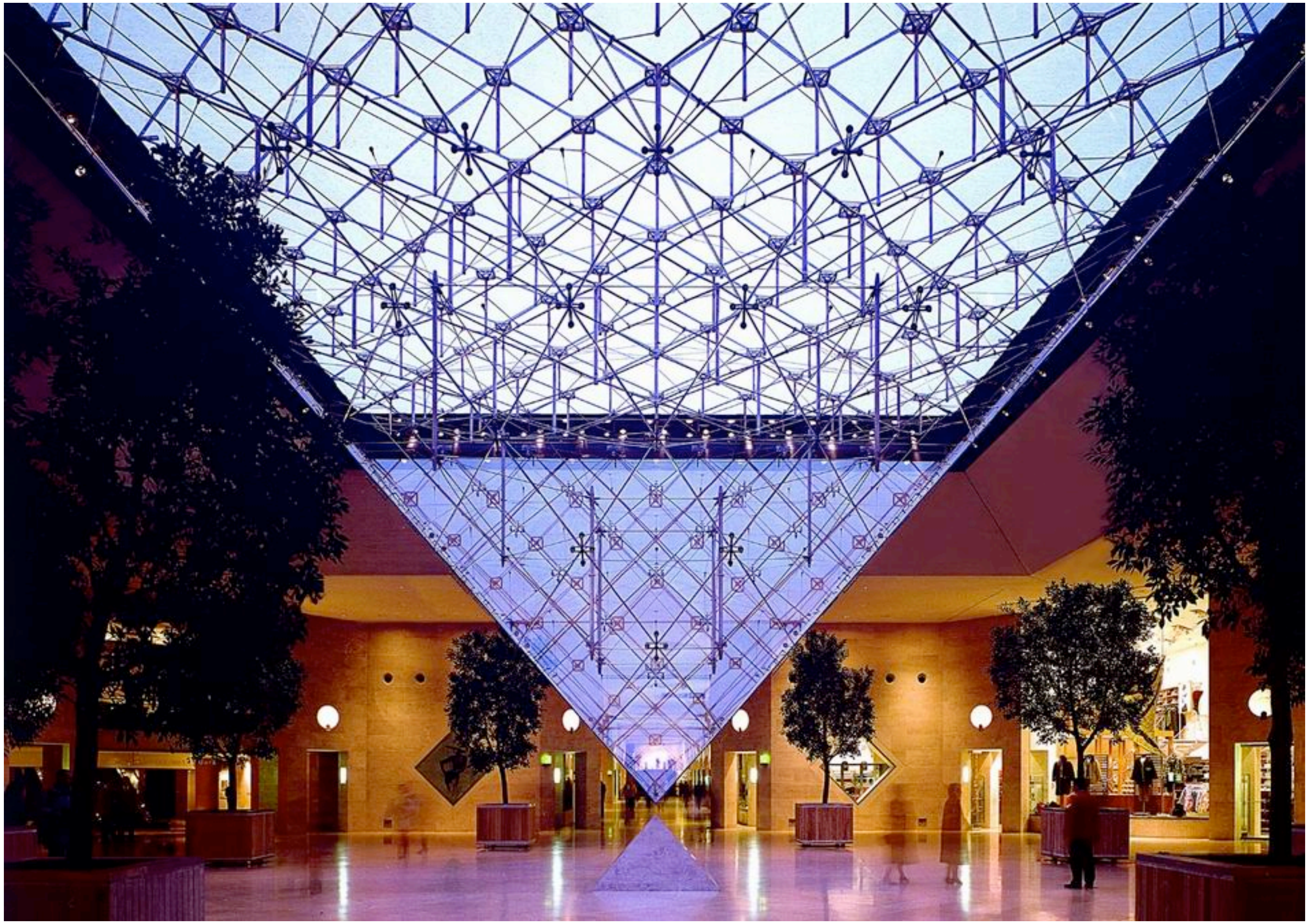
- Vector processors
- Symmetric multi-processors (SMPs)
- Nodes over inter-connection networks
- Instruction-level parallelism
- Multi-cores
- GPUs
- ...

Parallelism



Courtesy: Vivek Sarkar, Rice University

Parallelism



Parallelism

Thinking of Joe programmers

Automatic parallelization

"The reports of my death are highly exaggerated"

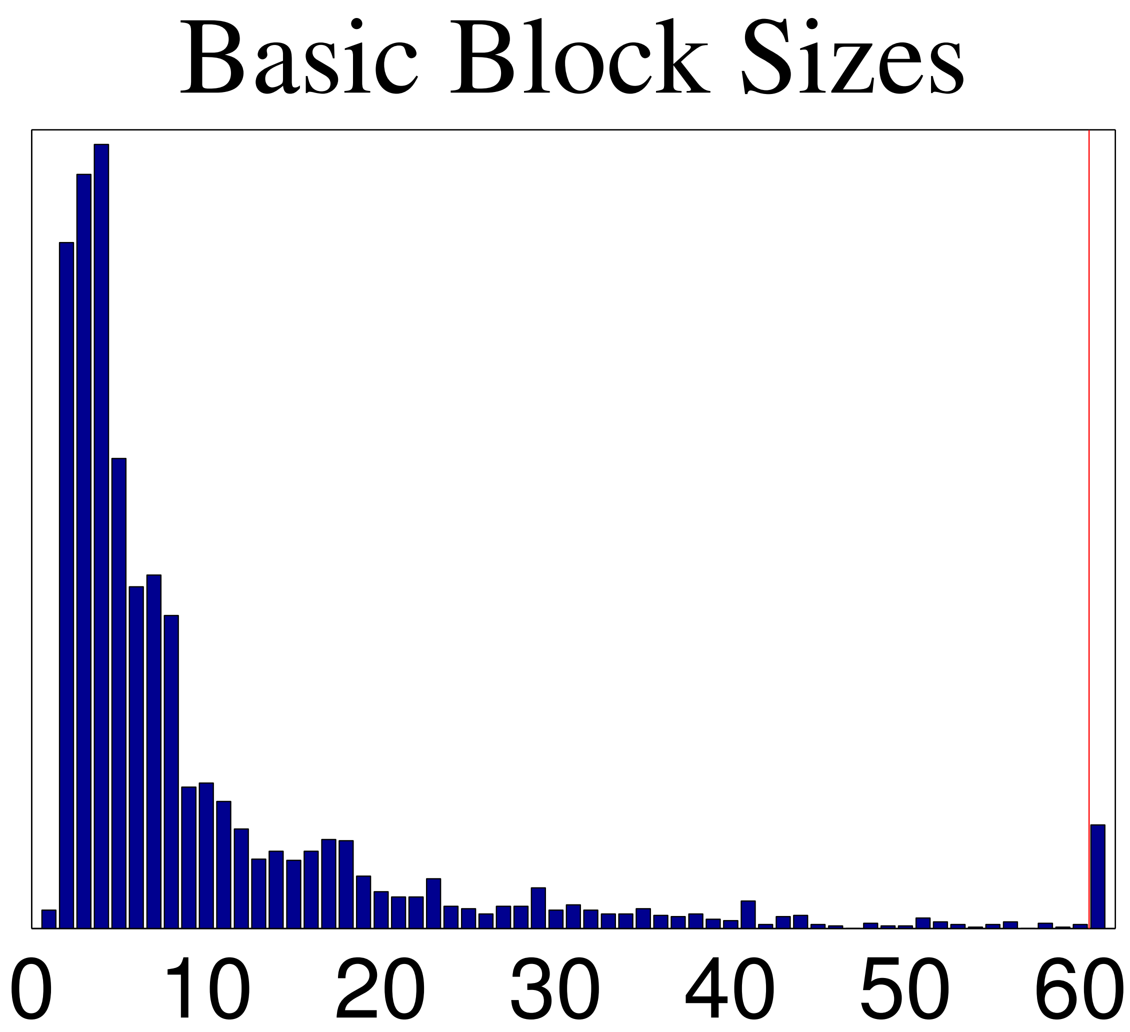
- MATLAB is the *lingua franca* of scientists and engineers
- Joe programmers would rather write in 10 minutes and let the program run for 24 hours, than vice versa
- They would still like their programs to run in 10 minutes!
- **We can leverage inferred types for automatic parallelization**

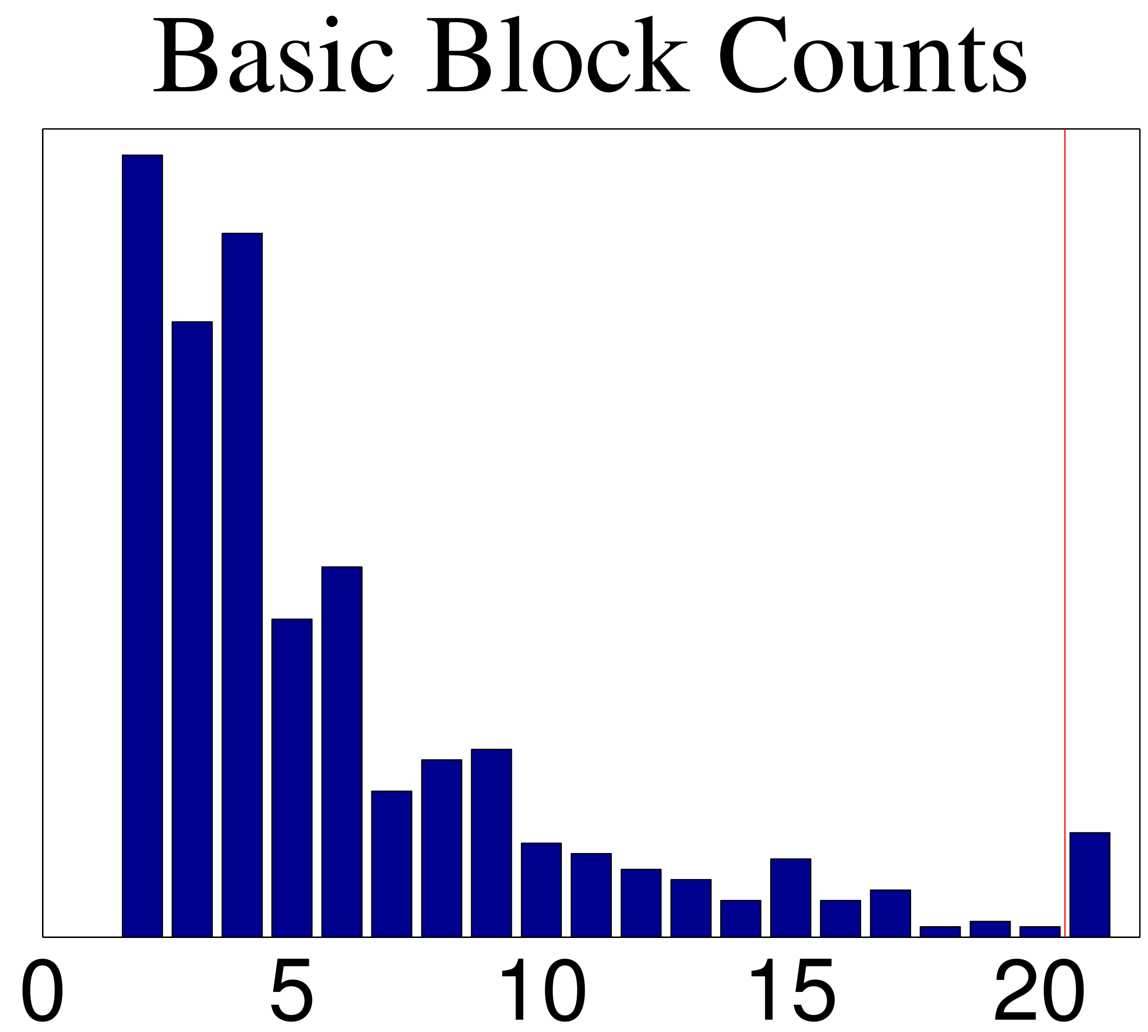
Parallelism in MATLAB

- 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





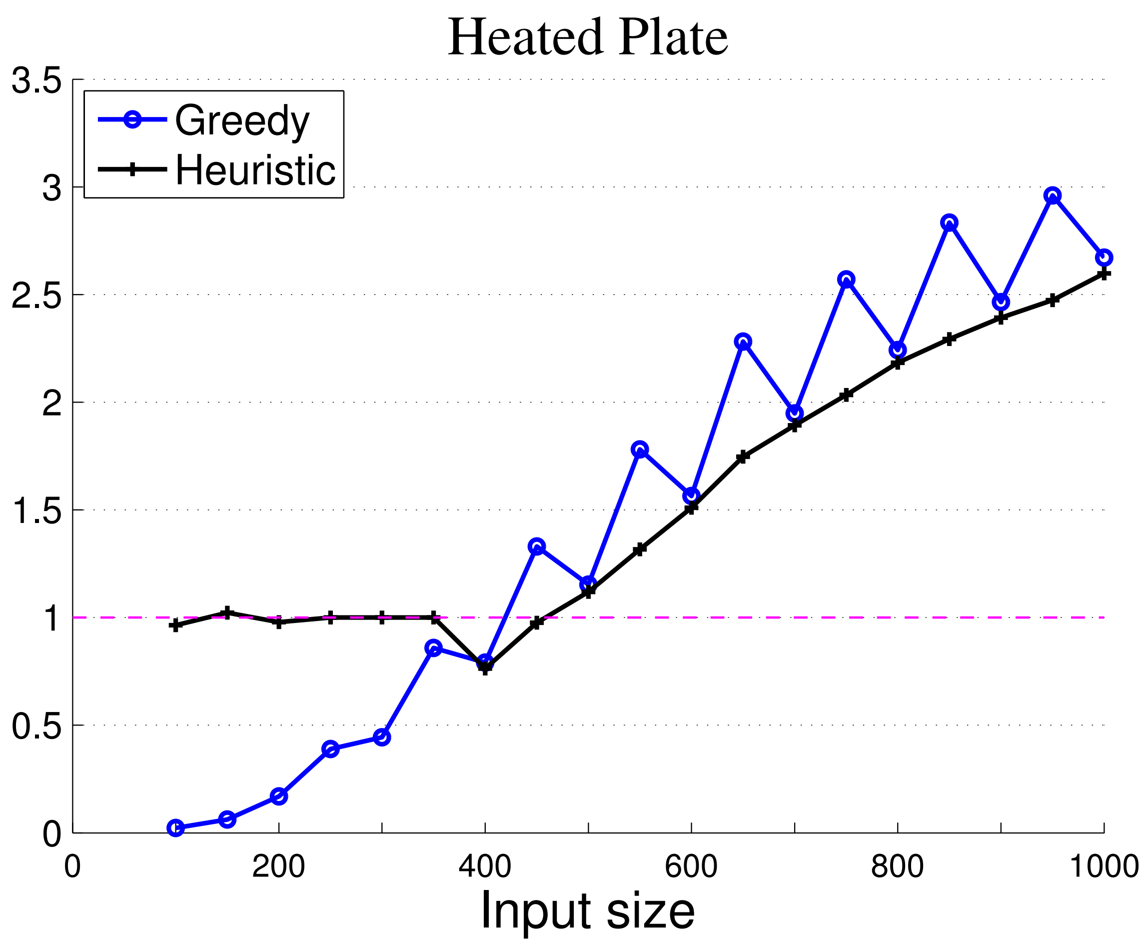
Automatic GPU Computation

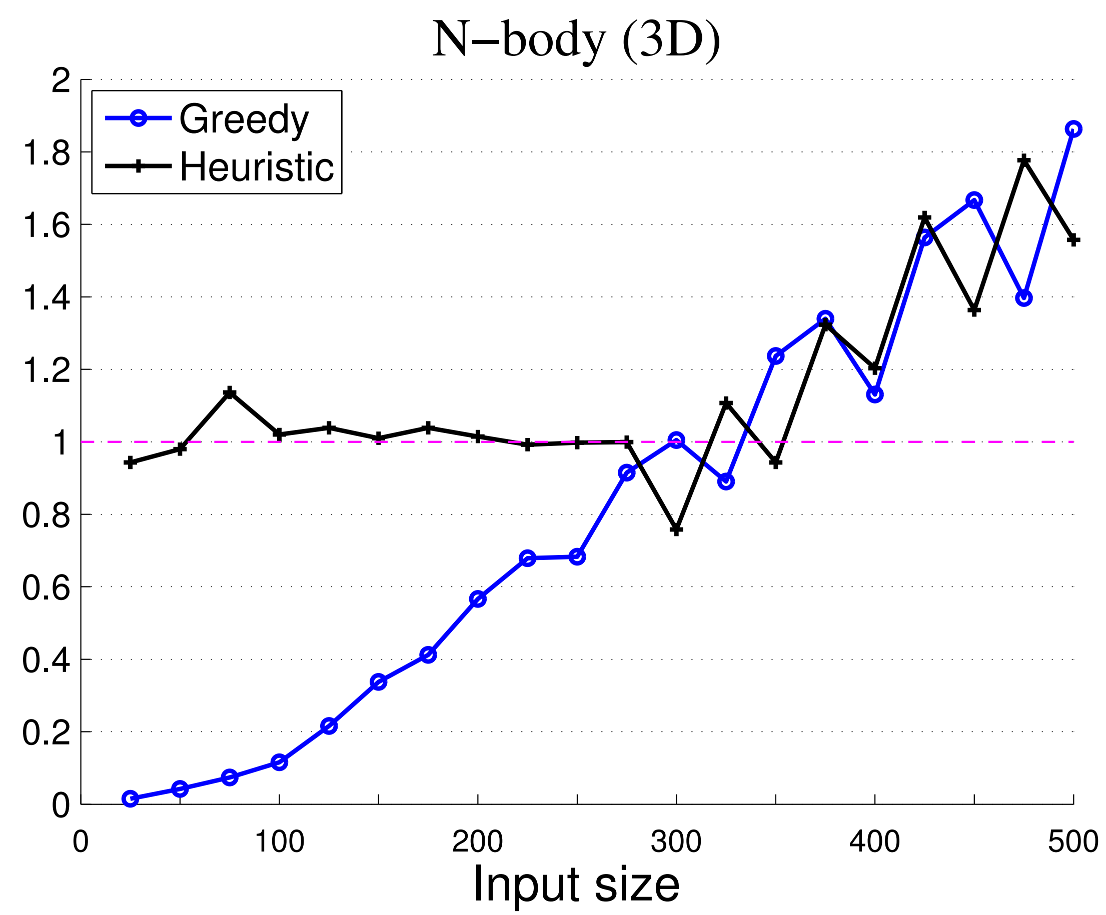
- Model the computation
  - cost model for CPU times
  - cost model for GPU times
  - cost model for CPU-GPU data transfer
- Solve a binary integer linear programming problem

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

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





Extending to other Languages

- Unique characteristics of MATLAB
  - simple basic data types
  - simple control flow
  - first-order functions
  - array language directly encodes data parallelism
- Ruby
  - object-oriented, with meta-programming support
  - closures, co-routines, higher-order functions
  - open classes

Ruby: Type Complications

```
class Foo
 def my_method
   ...
 end
end
…
f = Foo.new




…
g = Foo.new
```

Ruby: Type Complications

```
class Foo
 def my_method
   ...
 end
end
…
f = Foo.new
class Foo
  ...
end
…
g = Foo.new
```

Ruby: Type Complications

```
class Foo
  def my_method
    ...
  end
end
…
f = Foo.new

bar

…
g = Foo.new
```

```
def bar
   class Foo
     ...
   end
end
```

Challenges

- *Reasonable* static type inference
- Identifying conditions under which the inference is correct
- Detecting and verifying those conditions at run-time
- Possibly *speculating* on types

What about 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

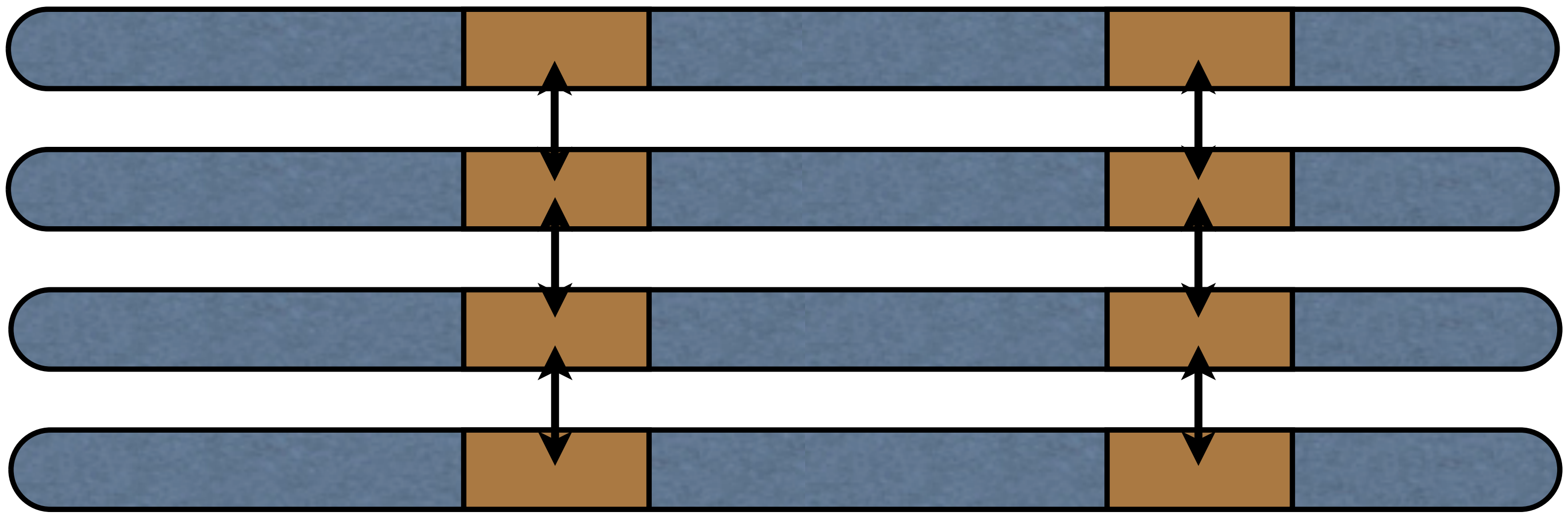
- Users must think in parallel (creativity)
  - but not be encumbered with optimizations that can be automated, or proving synchronization correctness
- Compiler focuses on what it can do (mechanics)
  - not creative tasks, such as determining data distributions, or creating new parallel algorithms
-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

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

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

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

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

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

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

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

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

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

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

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

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

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

↓

Source-level compiler (using ROSE)

↓

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;}
}}}
```

Nilesh Mahajan, Sajith Sasidharan, Arun Chauhan, and Andrew Lumsdaine. *Automatically Generating Coarse Grained Software Pipelining from Declaratively Specified Communication. In Proceedings of the 18th International Conference on High Performance Computing (HiPC), 2011. Student paper in the Student Research Symposium (SRS). To appear.*

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;}
}}}}
```

Nilesh Mahajan, Sajith Sasidharan, Arun Chauhan, and Andrew Lumsdaine. *Automatically Generating Coarse Grained Software Pipelining from Declaratively Specified Communication. In Proceedings of the 18th International Conference on High Performance Computing (HiPC), 2011. Student paper in the Student Research Symposium (SRS). To appear.*

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 };
```

Eric Holk, William Byrd, Nilesh Mahajan, Jeremiah Willcock,* Arun Chauhan*, and Andrew Lumsdaine. *Declarative Parallel Programming for GPUs. In Proceedings of the International Conference on Parallel Computing (ParCo), 2011.*

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);
```

Eric Holk, William Byrd, Nilesh Mahajan, Jeremiah Willcock,* Arun Chauhan*, and Andrew Lumsdaine. *Declarative Parallel Programming for GPUs. In Proceedings of the International Conference on Parallel Computing (ParCo), 2011.*

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); };
```

Eric Holk, William Byrd, Nilesh Mahajan, Jeremiah Willcock, Arun Chauhan, and Andrew Lumsdaine. *Declarative Parallel Programming for GPUs. In Proceedings of the International Conference on Parallel Computing (ParCo), 2011.*

Serious Joe programmer?

Scalable Speculative Parallelism on Clusters

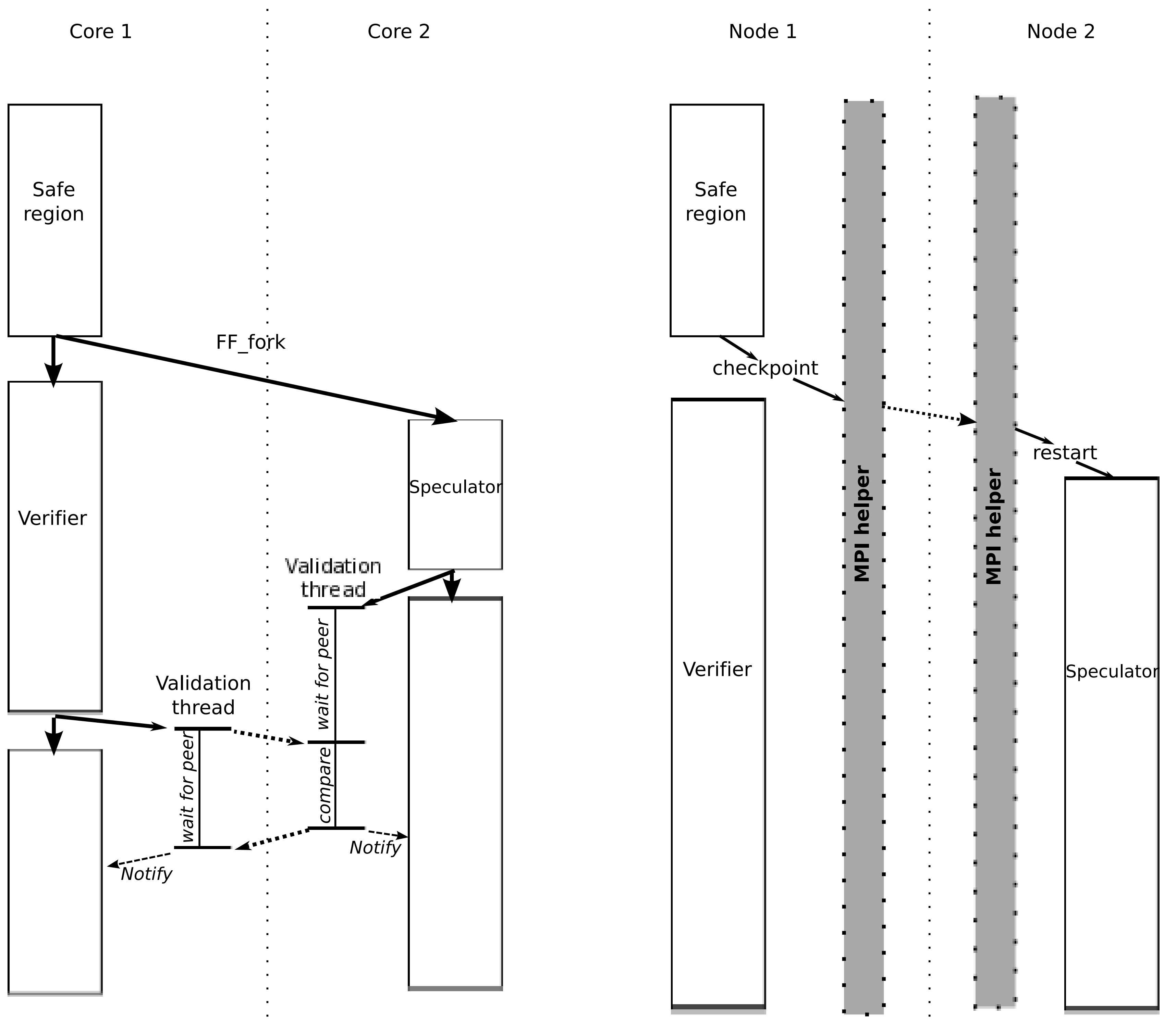
```
// safe code

// code where speculation possible (code region A)

// safe code

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

⇓

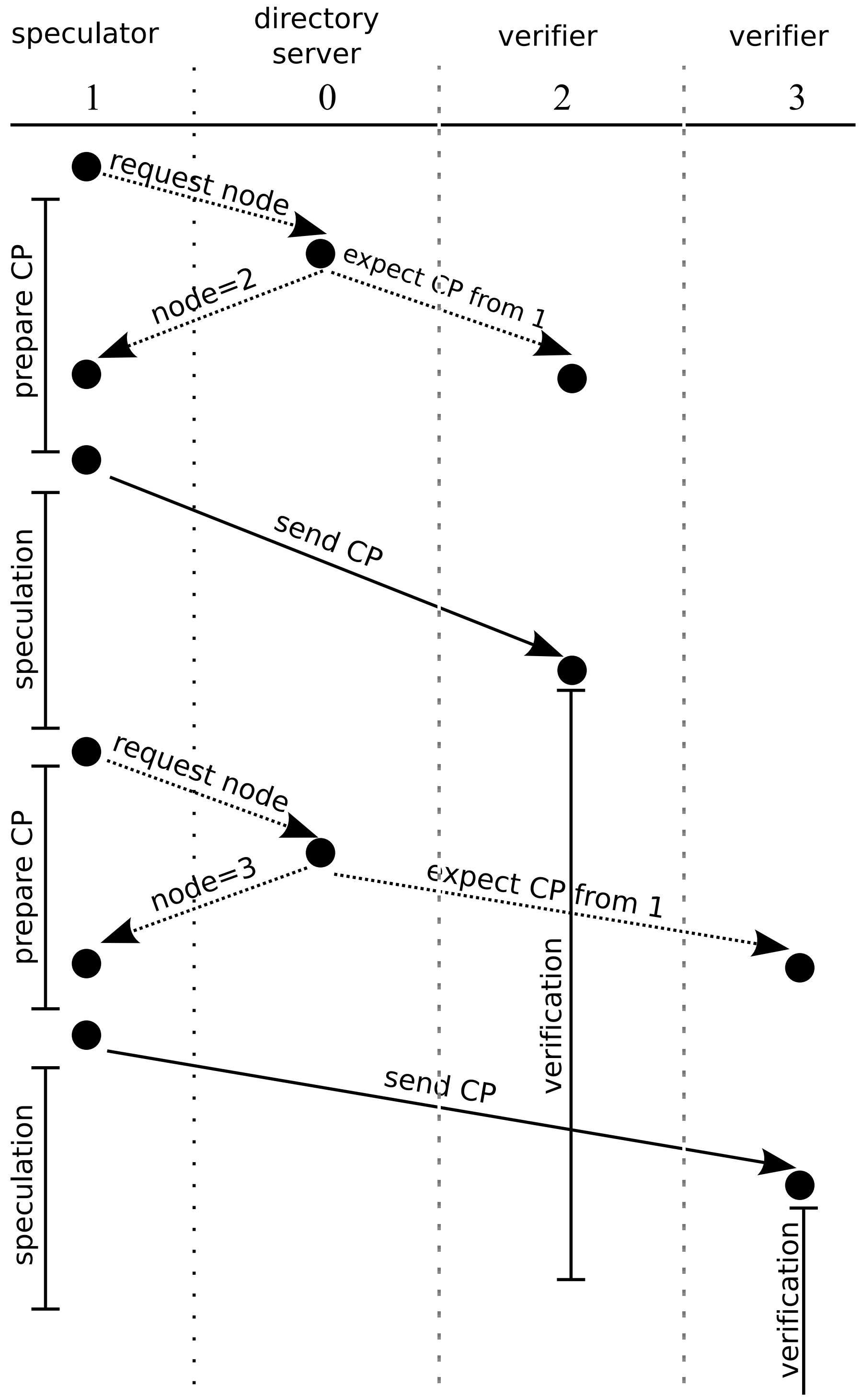
```
FF_init();

// 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();
```

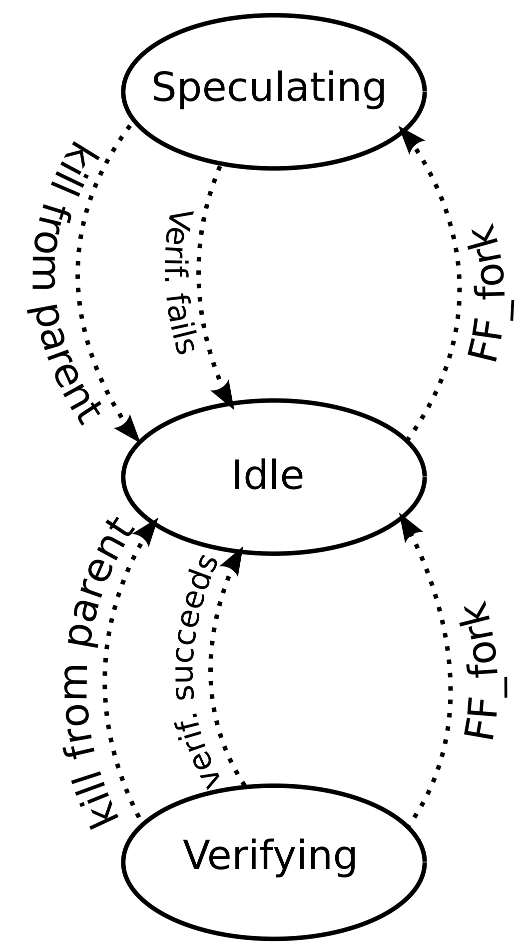
Devarshi Ghoshal, Sreesudhan R Ramkumar, and Arun Chauhan. *Distributed Speculative Parallelization using Checkpoint Restart. In Proceedings of the International Conference on Computational Science (ICCS), 2011*

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

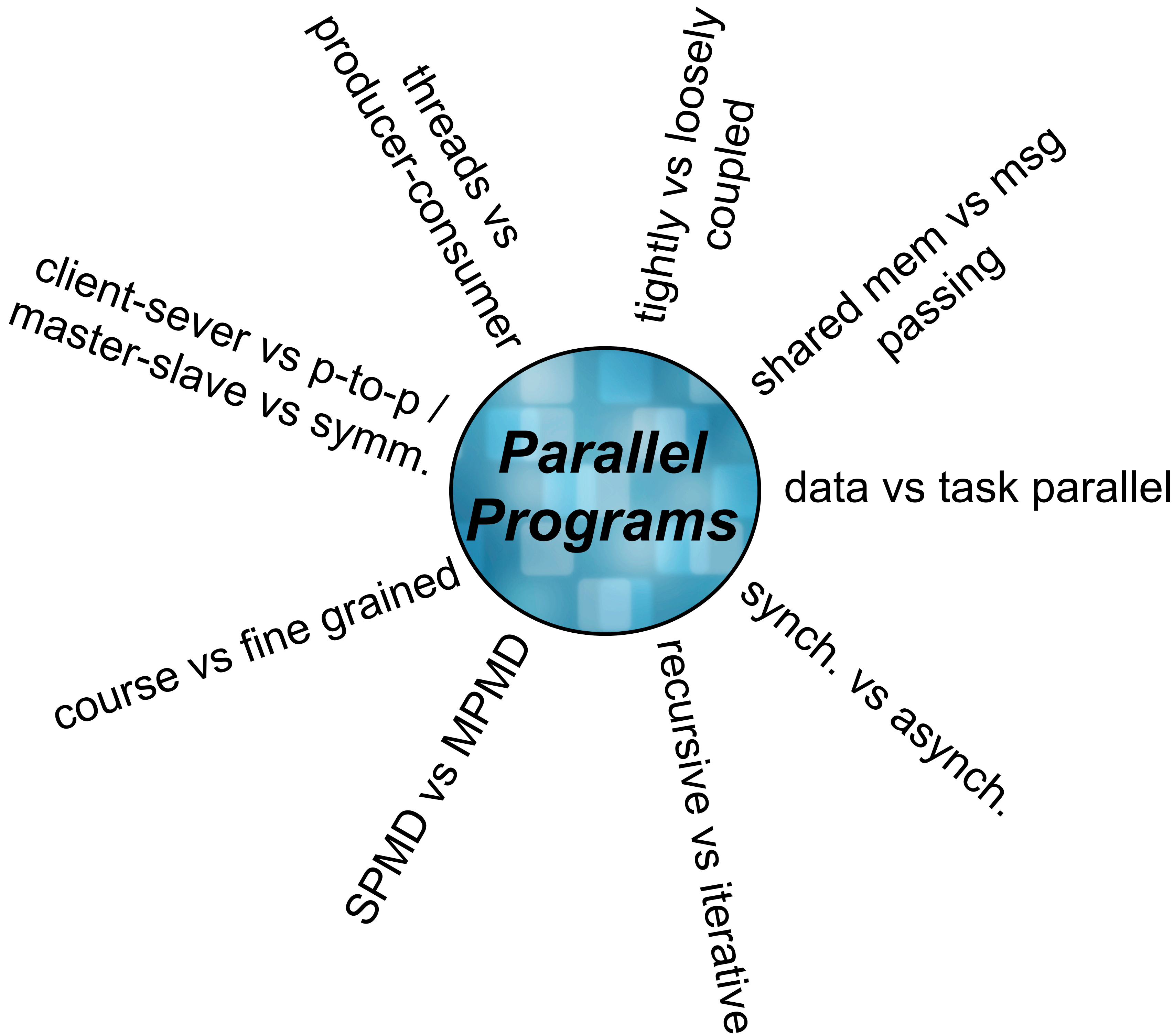
Running time of code, with speculation = $T + pk\dfrac{T}{s} + (1-p)kT$

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

- Effectively programming modern computers requires leveraging parallelism at multiple levels
- 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/

Google: arun indiana