The State of the Art in Parallel Languages

Calvin Lin
Department of Computer Science
The University of Texas at Austin

January 12, 2011

Outline

1. State of the art
- Pthreads ⬅
- OpenMP
- MPI

2. Moving forward
- High level languages
- PGAS languages
- Global view abstractions

3. Modern languages
- Chapel
- (X10)

2

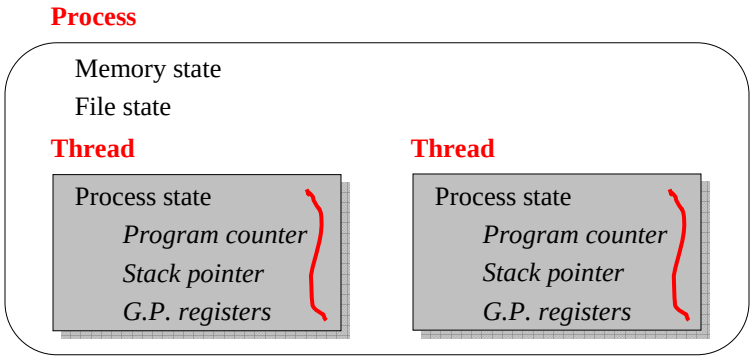
Pthreads Introduction

Consider a simple situation

- Simple hardware
- A trivial problem

3

Multithreading

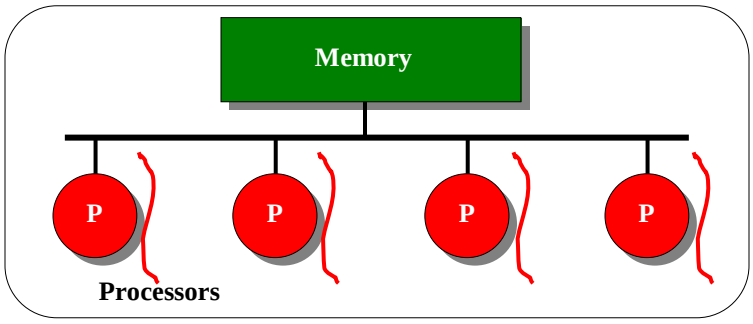


Threads

- Each thread has its own process state, but threads share memory and file state

4

Symmetric Multiprocessors



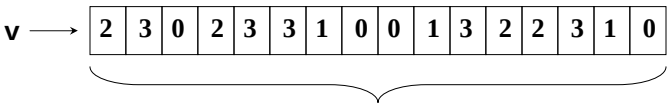
Symmetric Multiprocessor

- Processors share physical memory, and all physical memory is equidistant from all processors
- Multiple threads can execute in parallel on multiple processors, and threads can communicate through shared memory

5

Simple Example

Count the number of 3's in an array.



n=16 count=5

Serial code:

```
int *v;
int n;
int count;
```

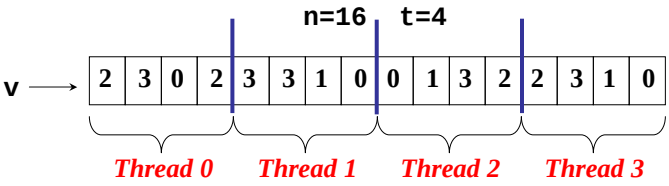
```
void count3s () {
  count = 0;
  for (int i=0; i<n; i++)
      if (v[i]==3)
          count++;
}
```

6

Simple Example: Parallelization

Each thread is responsible for counting the 3's in some portion of the array.

With `n` elements and `t` threads, each thread is responsible for `n/t` array elements.



7

Simple Example

```
int t;

void count3s() {
  count = 0;

  /* Create t threads */
  for (i=0; i<t; i++)
       /*
        * Each thread calls count3s_thread with
        * parameter i
        */
       thread_create(count3s_thread, i);

  /* Wait for threads to terminate */
  for (i=0; i<t; i++)
        thread_join();
}
```

Simple Example (cont)

```
void count3s_thread(int id) {

  /* Determine portion of array to work on */
   int n_per_thread = n/t;
   int start = id * n_per_thread;

  /* Count the 3's in my portion of array */
   for (i=start; i<start+n_per_thread; i++)
       if (v[i]==3)
           count++;
}
```

Are there any problems with this code?

This code will not work because of a data race at the increment of **`count`**

9

Data Races

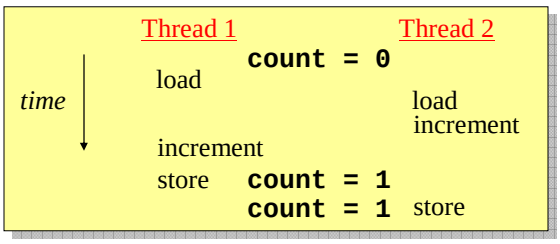
Definition

A **data race** occurs when two or more threads can modify the same memory location at the same time

Example

The statement **`count++`** is actually translated into 3 instructions:

1. Load **`count`** in register
2. Increment register contents
3. Store register in **`count`**



10

Mutual Exclusion

Solution

- To prevent the data race, we must ensure that at all times at most one thread is executing the **`count++`** statement
- We can guarantee mutual exclusion by using a data object called a **mutex** **(**also called a **lock)**

11

Mutexes

Usage

- A **mutex** is a data object with
  - 2 states: **`locked`** and **`unlocked`**
  - 2 methods: **`lock`** and **`unlock`**
- Critical code is then bracketed by calls to **`lock`** and **`unlock`** a mutex

```
mutex m;

mutex_lock(m);
                          } critical
                            section
mutex_unlock(m);
```

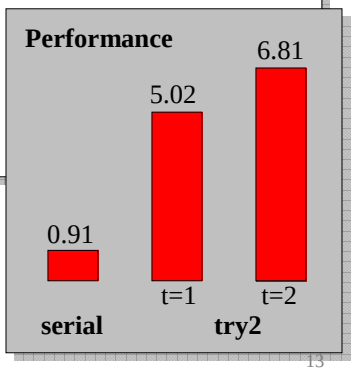
12

Simple Example—Second Try

```
mutex m;

void count3s_thread(int id) {
  /* Count 3's in portion of array */
   for (i=start; i<start+n_per_thread; i++)
       if (v[i]==3) {
           mutex_lock(m);
           count++;
           mutex_unlock(m);
       }
}
```

Does this code work correctly? Yes.

Execution time in seconds
on SPARCstation 20
n=8 million, count=2 million



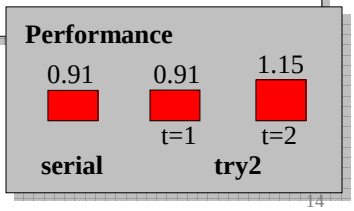
Simple Example—Third Try

Each thread counts in a private counter, then combines at the end

```
int private_count[16];

void count3s_thread(int id) {
   for (i=start; i<start+n_per_thread; i++)
       if (v[i]==3)
           private_count[id]++;

   mutex_lock(m);
   count += private_count[id];
   mutex_unlock(m);
}
```

What's going on? *We have false sharing*

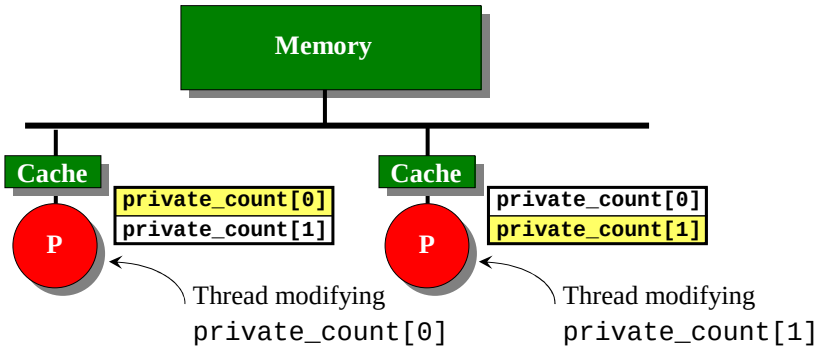
Execution time in seconds
on SPARCstation 20
n=8 million, count=2 million



False Sharing

Cache consistency

- On our hardware, caches are kept consistent
- False sharing occurs when 2 or more threads modify different data on the same cache line



The effort expended to maintain consistency can hurt performance

15

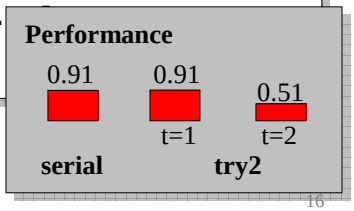
Simple Example—Success at Last

Allocate padding between private counters

```
struct padded_int {
  int value;
  char padding[32];
} private_count[16];

void count3s_thread(int id) {
   for (i=start; i<start+n_per_thread; i++)
       if (v[i]==3)
           private_count[id].value++;
   mutex_lock(m);
   count += private_count[id].
   mutex_unlock(m);
}
```

Notice how convoluted our data structure has become

Execution time in seconds on SPARCstation 20 n=8 million, count=2 million



16

Lessons

Programming with Pthreads is difficult

- The parallel code is considerably more complicated than its sequential counterpart
- Getting things right can be tricky
- Getting good performance can be trickier
- Getting good performance can require knowledge of low-level details

17

But Wait, There's More!

More complexity

- Threads typically need to synchronize, which can be tricky

18

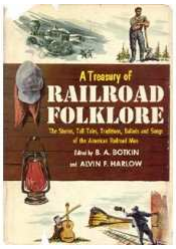
Synchronization Error

Railway Safety

"When two trains approach each other at a crossing, both shall come to a full stop and neither shall start up again until the other has gone."

– Statute passed by the Kansas state legislature

[*A Treasury of Railroad Folklore*, Botkin and Harlow, eds, Bonanza Books, p. 381]



Synchronization Using Condition Variables

Condition variables

– Allow a thread to wait until some event has occurred

Example

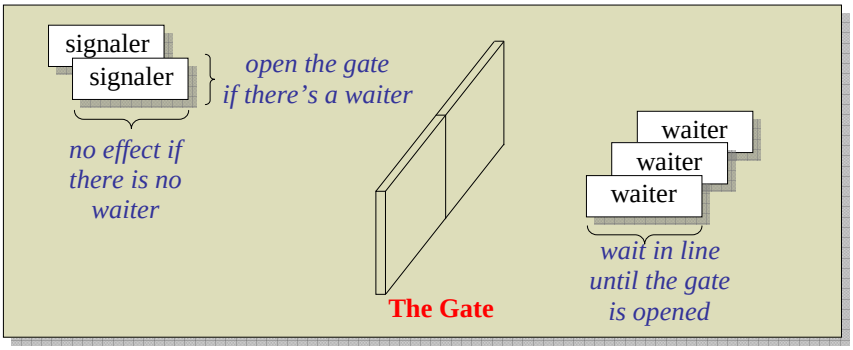
– Suppose a thread wants to eat an apple on a table

– If there is no apple on the table, the thread can wait until an apple is placed on the table

20

Semantics of Condition Variables

Conceptually, the condition variable is a gatekeeper



21

Protecting Condition Variables

Does the following code work?

```
if (apples==0) {                        // Wait for more apples
    pthread_cond_wait (&cond, &lock);
    // Wakeup and eat an apple
```

Error: Some other thread may awaken first and eat the last apple

Hoare vs. Hansen Semantics

Correct usage:

```
pthread_mutex_                                  pples with lock
while (apples==0) {
     The lock is relinquished while the thread waits
     pthread_cond_wait (&cond, &lock);
    // The lock is re-acquired here
    // Loop to see if there are any apples to eat
}
pthread_mutex_unlock (&lock);
```

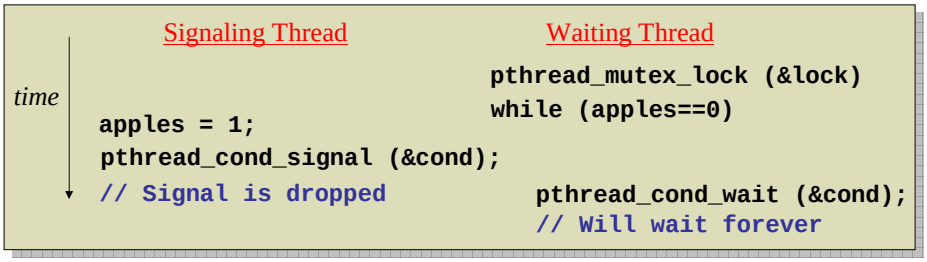
22

Protecting Condition Variables (cont)

The correct code for the waiting thread:

```
pthread_mutex_lock (&lock);       // Protects apples
while (apples==0) {
    pthread_cond_wait (&cond, &lock);
}
pthread_mutex_unlock (&lock);     // Now safe to eat an apple
```

Should the signaling thread use the lock, too? Yes.



23

Protecting Condition Variables (cont)

Case 1:

Signaling Thread

```
pthread_mutex_lock(&lock);
apples = 1;
pthread_cond_signal(&cond);
pthread_mutex_unlock(&lock);
```

Waiting Thread

```
pthread_mutex_lock(&lock);
while (apples==1)
    // Apples now available
pthread_mutex_unlock(&lock);
```

Case 2:

Signaling Thread

```
pthread_mutex_lock(&lock);
apples = 1;
pthread_cond_signal(&cond);
pthread_mutex_unlock(&lock);
```

Waiting Thread

```
pthread_mutex_lock(&lock)
while (apples==0)
   // No apples available
   pthread_cond_wait (&cond);
   // Lock is released here

while (apples==0)
   // Apples now available
   pthread_mutex_unlock(&lock);
```

24

Exercise: Where Is It Safe to Signal?

Code 1:

```
pthread_mutex_lock(&lock);
apples = 1;
pthread_cond_signal(&cond);
pthread_mutex_unlock(&lock);
```

Code 2:

```
pthread_mutex_lock(&lock);
pthread_cond_signal(&cond);
apples = 1;
pthread_mutex_unlock(&lock);
```

Code 3:

```
pthread_mutex_lock(&lock);
apples = 1;
pthread_mutex_unlock(&lock);
pthread_cond_signal(&cond);
```

25

Where Is It Safe to Signal?

All of the three code fragments are correct.

The condition variable must be signaled either (1) while the lock is held or (2) after the final result of the logical condition has been changed.

26

All Three Codes are Correct

Code 1:

```
pthread_mutex_lock(&lock);
apples = 1;
pthread_cond_signal(&cond);
pthread_mutex_unlock(&lock);
```

Code 2:

```
pthread_mutex_lock(&lock);
pthread_cond_signal(&cond);
apples = 1;
pthread_mutex_unlock(&lock);
```

Which code will perform the best?

Code 3:

```
pthread_mutex_lock(&lock);
apples = 1;
pthread_mutex_unlock(&lock);
pthread_cond_signal(&cond);
```

27

Exercise: Which Will Perform Best?

Code 3 will perform best.

- Gives the awakening threads the best chance of not blocking on the lock
- Has the shortest critical section

Code 3:

```
pthread_mutex_lock(&lock);
apples = 1;
pthread_mutex_unlock(&lock);
pthread_cond_signal(&cond);
```

28

Lessons

Added complexity

- Synchronization with condition variables can be tricky
- It's often hard to reason about race conditions

29

What About Optimizations?

Consider barrier synchronization

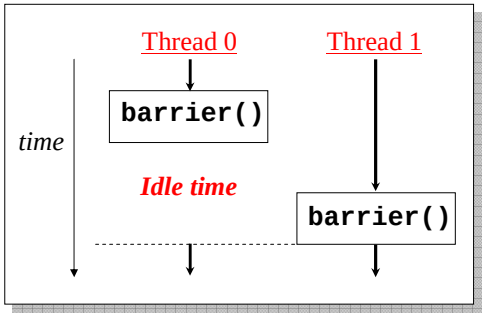
- All threads agree to wait at a certain point in the code
- e.g. "Let's all meet at Largo da Oliveira**,** and then we'll choose a restaurant"

30

The Problem

Synchronization leads to idle time

- Threads arrive at different times
- All except the last one has to wait

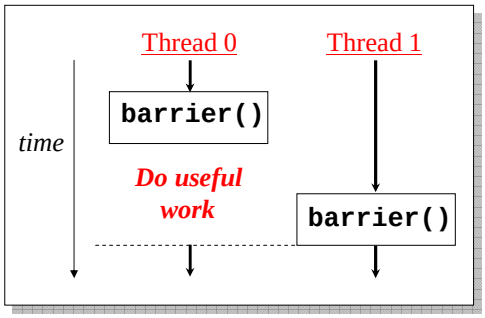


31

Hiding the Cost of Synchronization

Optimization

- Do useful work while waiting
- **Overlap synchronization latency with computation**



- **How do we accomplish this?**

32

Split-Phase Operations

Basic idea

- Separate **initiation** from **completion**
- No thread **completes** until all threads have **initiated**

Example: Split-phase barrier

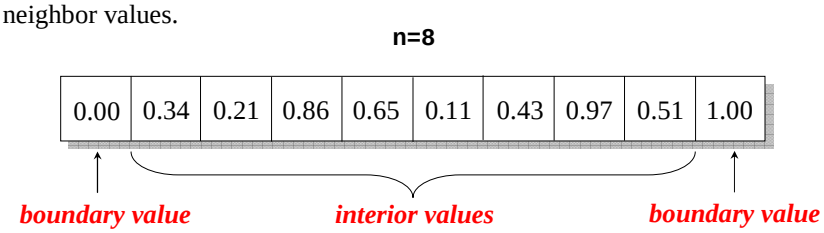
- Initiation: `barrier.arrive()`
- Completion: `barrier.wait()`

```
/* Initiate synchronization */
  barrier.arrive();

/* Do useful work */

/* Complete synchronization */
  barrier.wait();
```

33

Example: Relaxation

We start with an array of **n+2** values: **n** interior values and 2 boundary values. At each iteration, we replace each interior value with the average of its 2 neighbor values.



Relaxation on 2D and 3D arrays is used in practice to solve systems of differential equations such as the Navier-Stokes equations for fluid flow

34

Parallel Relaxation

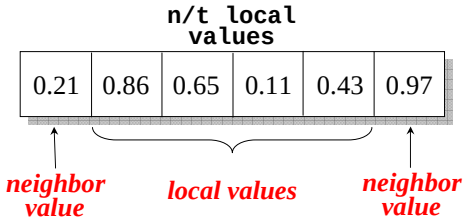
```
double *val, *new;          // Values array
int n;                      // Number of interior values
int t;                      // Number of threads
int iterations;             // Iterations to perform

thread_main(int self) {
  int n_per_thread = n / t;
  int start = self * n_per_thread;

  // For each iteration
  for (int i=0; i<iterations; i++) {
      // Update values
      for (int j=start; j<start+n_per_thread; j++)
         new[j] = (val[j-1] + val[j+1]) / 2.0;
      swap(new, val);
      barrier();            // Synchronize
  }
}
```

Can we use split-phase barriers to reduce synchronization costs?

Parallel Relaxation—A Closer Look

Consider an individual thread's work

- Each thread is responsible for updating `n/t` values at each iteration
- Each thread accesses **2** additional neighbor values:



  - The neighbor values are modified by the left and right neighbors, so the access and update of these neighbor values must be synchronized
  - The local values are only accessed by this thread, so they require no synchronization

Relaxation with Split-Phase Barriers

```
thread_main(int self) {
  ...
  // For each iteration
  for (int i=0; i<iterations; i++) {
      // Update neighbor values
      int j = start;
      val[j] = (val[j-1] + val[j+1]) / 2.0;
      j = start + n_per_thread –1;
      val[j] = (val[j-1] + val[j+1]) / 2.0;

      barrier.arrive();          // Initiate barrier

      // Update local values
      for (j=start+1; j<start+n_per_thread_-1; j++)
          new[j] = (val[j-1] + val[j+1]) / 2.0;
      swap(new, val);
      barrier.wait();            // Complete barrier
  }
}
```

37

Split-Phase Barrier

```
class Barrier {
  int nThreads;
  int count = 0;
  pthread_mutex_t lock;
  pthread_cond_t all_here;

public:
  Barrier (int t);
  ~Barrier (void);

  // Initiate barrier
  void arrive (void);

  // Check if done
  int done (void);

  // Block until done
  void wait (void);
};
```

```
int Barrier::done (void) {
  int rval;
  pthread_mutex_lock (&lock);

  // Done if the count is zero
  rval = !count;

  pthread_mutex_unlock (&lock);
  return rval;
};
```

38

Split-Phase Barrier

```
void Barrier::arrive (void) {
  pthread_mutex_lock (&lock);

  count++;              // Another thread has arrived
  // If last thread arrives, then wake up any waiters
  if (count==nThreads) {
     pthread_cond_broadcast (&all_here);
     count = 0;
  }
  pthread_mutex_unlock (&lock);
};
```

```
void Barrier::wait (void) {
  pthread_mutex_lock (&lock);

  // If not done, then wait
  if (count!=0)
     pthread_cond_wait (&all_here, &lock);

  pthread_mutex_unlock (&lock);
};
```

This code will not work!

What's wrong?

39

Deadlock

	Thread 0	Thread 1	count
			0
	`barrier.arrive()`		1
		`barrier.arrive()`	0
time		`barrier.wait()`	
		`barrier.arrive()`	1
	`barrier.wait()`	`barrier.wait()`	

Both threads are now in **`pthread_cond_wait()`**, so we have deadlock

What caused the problem?
Two different instances of the barrier share the same value of **`count`**

40

Split-Phase Barrier—Revisited

Keep track of the current **phase**. The **`arrive()`** method returns the current phase, which is then passed into the **`done()`** and **`wait()`** methods

```
class Barrier {
  int nThreads;
  int count;
  int phase;     // current phase
  pthread_mutex_t lock;
  pthread_cond_t all_here;

public:
  Barrier (int t);
  ~Barrier (void);
  // Initiate barrier and return phase
  int arrive (void);
  // Check if phase p is done
  int done (int phase);
  // Block until phase p is done
  void wait (int phase);
};
```

41

Split-Phase Barrier—Revisited (cont)

```
int Barrier::arrive (void) {
  int p;
  pthread_mutex_lock (&lock);

  p = phase;       // Get phase
  count++;         // Another thread has arrived

  // If last thread to arrive,
  // then wake up any waiters and go to next phase
  if (count==nThreads) {
     pthread_cond_broadcast (&all_here);
     count = 0;
     phase = 1 - phase;
  }

  pthread_mutex_unlock (&lock);
  return p;
};
```

42

Split-Phase Barrier—Revisited (cont)

```
int Barrier::done (int p) {
  int rval;
  pthread_mutex_lock (&lock);

  // Done if phase has changed
  rval = (phase != p);

  pthread_mutex_unlock (&lock);
  return rval;
};
```

```
void Barrier::wait (int p) {
  pthread_mutex_lock (&lock);

  // If not done, then wait
  while (p==phase)
     pthread_cond_wait (&all_here, &lock);

  pthread_mutex_unlock (&lock);
};
```

43

Deadlock Problem Resolved

	Thread 0	Thread 1	count	phase
			0	0
time ↓	barrier.arrive()		1	
		barrier.arrive()	0	1
		barrier.wait(0)		
		barrier.arrive()	1	
	barrier.wait(0)	barrier.wait(1)		

Deadlock does not occur: Thread 0's call to **`barrier.wait(0)`** returns without waiting.

We can now distinguish between the first barrier and the second barrier

44

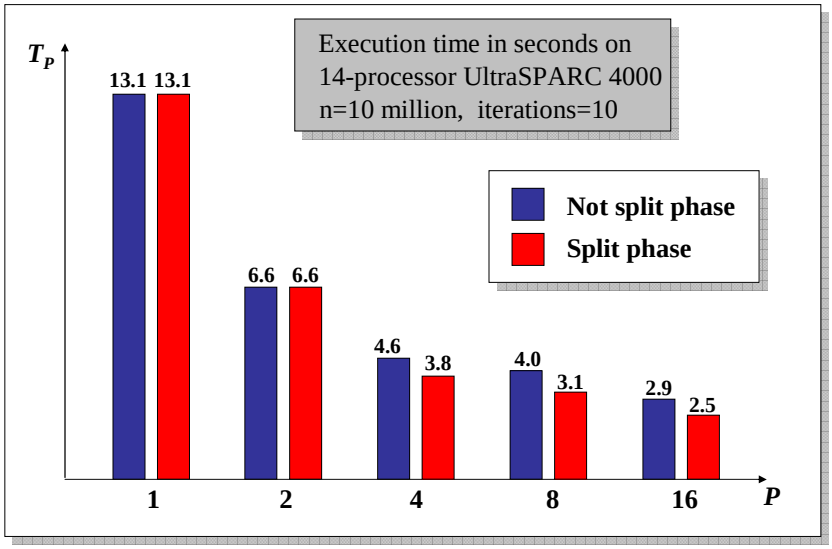
Relaxation Revisited

```
thread_main(int self) {
  int phase
  ...
  // For each iteration
  for (int i=0; i<iterations; i++) {
      // Update neighbor values
      int j = start;
      val[j] = (val[j-1] + val[j+1]) / 2.0;
      j = start + n_per_thread –1;
      val[j] = (val[j-1] + val[j+1]) / 2.0;

      phase = barrier.arrive(); // Inititate barrier

      // Update local values
      for (j=start+1; j<start+n_per_thread_-1; j++)
          new[j] = (val[j-1] + val[j+1]) / 2.0;
      swap(new, val);
      barrier.wait(phase);      // Complete barrier
  }
}
```

The logic of our code is more complex now

45

Relaxation Performance

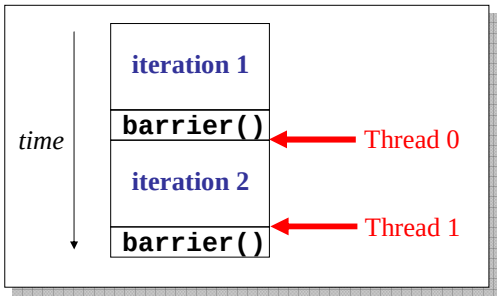


46

Split Phase Operations

Relaxed synchronization

– With standard barriers, how far out of synch can two threads become?



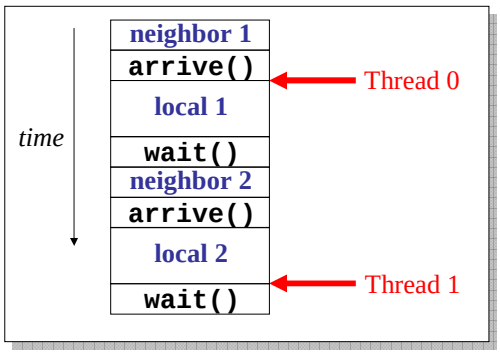
– With split-phase barriers, how far out of synch can two threads become?

47

Split Phase Operations (cont)

Relaxed synchronization

– About 1.5 iterations. Looser synchronization ⇒ improved performance



48

Split Phase Operations (cont)

A common technique

- Split-phase message receive
- Split-phase reductions
  - Reductions apply some operator, such as compute-minimum, across some set of values
- Asynchronous I/O
- Asynchronous Remote Procedure Call
- Speculative loads in the IA64
- . . .

Increase parallelism
Increased complexity

49

Pthreads Summary

Advantages

- Powerful

Disadvantages

- Low-level
- Error-prone
- Limited to shared memory machines

50

Outline

1. State of the art
   - Pthreads
   - OpenMP
   - MPI
2. **Moving forward**
   - High level languages
   - PGAS languages
   - Global view abstractions
3. **Modern languages**
   - Chapel

51

OpenMP Interface

Easy to use

- Popular and easy to use
- Supports C/C++ and Fortran
- Assumes a shared memory platform

Limited model

- FORALL loops– loop iterations can execute independently
- Reductions
- Tasks
- Atomic sections

http://www.openmp.org

52

MPI—Message Passing Interface

Goals

- Portable communication interface
- Support efficient communication across a wide variety of machines
- Provide a reliable communication interface

History

- Defined in 1992 by a large consortium (60 individuals, 40 organizations)
- Widely adopted
  - Many implementations, including vendor-specific implementations
  - Widely used

53

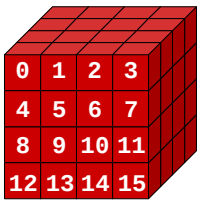
The Basic Model

Distributed memory

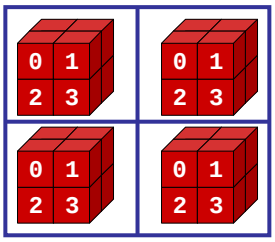
- Each process sees only a local address space
- Processes send messages to communicate with other processes

Data structures

- Described as a collection of Fragmented Views, rather than a single Global View
- Programmer must make the mapping



Global View



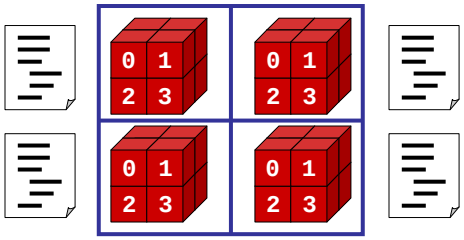
Fragmented View (4 processes)

54

Basic Model (cont)

SPMD code

- Write one piece of code that executes on each processor



Fragmented View (4 processes)

55

Communication in MPI

Two types

- Collective communication
- Point-to-point communication

56

Collective Communication

Barriers
- Pure synchronization

Gather
- Collect data from all processes to a single process

Scatter
- Spread data from one process to all other processes

Reductions
- Compute max, min, sum of values that reside on multiple processes
- Can also compute some user-defined function

Scans
- Parallel prefix

57

Communication in MPI

Two types
- Collective communication
- Point-to-point communication

58

MPI Example: Point-to-Point Communication

```
do {
    /* do somethi                    g
    if (rank==0) {
        scanf ("%d", &value);
        MPI_Send (&value, 1, MPI_INT, rank+1, 0,
                  MPI_COMM_
    }
    else {
        MPI_Recv (&value, 1, MPI_INT, rank-1, 0,
                  MPI_COMM_WORLD);
        if (rank < size-1)
            MPI_Send (&value, 1, MPI_INT, rank+1, 0,
                  MPI_COMM_WORLD);
    }
    printf ("Process %d got %d\n", rank, value);
} while (value >= 0);
```

The address of the data to send

The type of the data

Message tag

The length of the data to send

Message destination

59

MPI Example: Point-to-Point Communication

```
do {
    /* do something interesting
    if (rank==0) {
        scanf ("%d", &value);
        MPI_Send (&value, 1, MPI_INT, rank+1, 0,
                  MPI_COMM_WORLD);
    }
    else {
        MPI_Recv (&value, 1, MPI_INT, rank-1, 0,
                  MPI_COMM_WORLD);
        if (rank < size-1)
            MPI_Send (&value, 1, MPI_INT, rank+1, 0,
    }
    printf (
} while (val
```

What does this code do?

Observations

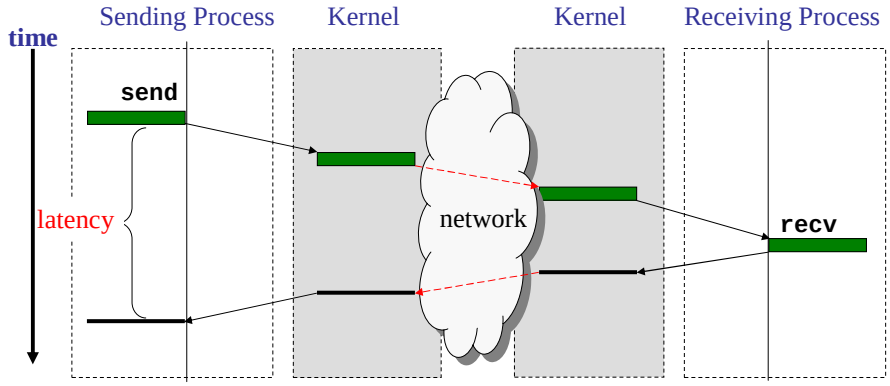
- Communication code is large and cumbersome
- Fragmented View requires more mental effort
  ⇒ more difficult to reason about correctness

60

Round Trip Message Latency

Latency

– Much copying and synchronization, eg, **`MPI_Bsend()`**

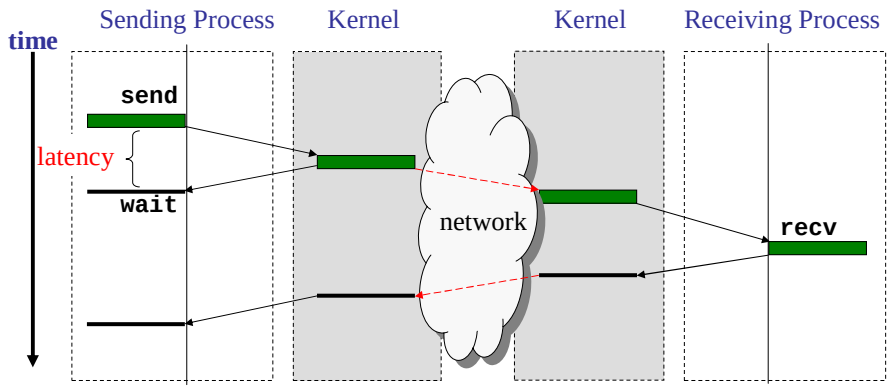


61

Lower Latency Communication

Non-buffered sends

– The send returns when data has been copied to the kernel

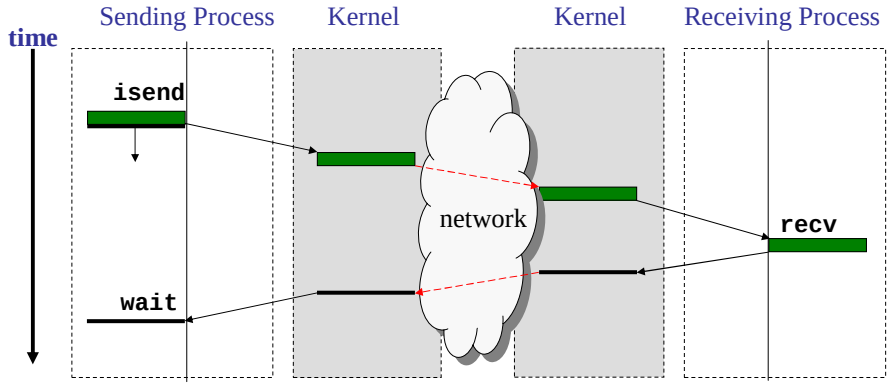


62

The Cost of Non-Buffered Communication

Danger

- Buffer might be overwritten before being copied to the kernel
- The wrong data will be sent!



63

Design Issue

The dilemma

- Buffered sends are safer but slower
- Non-buffered sends are faster but more dangerous
- Which form of point-to-point communication should MPI provide?

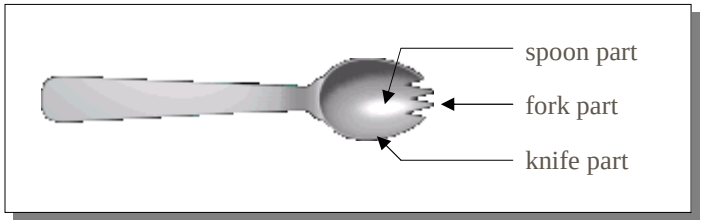
64

The Challenge

The challenge facing MPI

- Different clients have different needs
- No single implementation is ideal for all situations

Real World Analogy: The Spork



65

The Solution: Interface Bloat

Common approach

- Create lots of specialized routines
- Let user choose the appropriate routine

66

Interface Bloat in MPI

Short term problems:
- Complex interface
- Specialized routines can be difficult to use

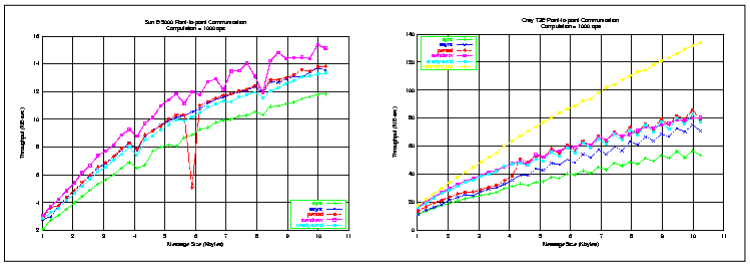
12 ways (modes) to perform point-to-point communication:

	Normal	Sync	Ready	Buffered
Normal	MPI_Send	MPI_Ssend	MPI_Rsend	MPI_Bsend
Nonblock	MPI_Isend	MPI_Issend	MPI_Irsend	MPI_Ibsend
Persistent	MPI_Send_init	MPI_Ssend_init	MPI_Rsend_init	MPI_Bsend_init

67

Interface Bloat in MPI

Long term problems:
- No performance portability
- Application becomes less general



68

Problems with Interface Bloat

Premature optimization

- Requires manual changes to application source
- Embeds optimizations into application source

Long Term Problems

- Complicates maintenance
- Defeats portability

69

Summary

Pthreads and MPI

- Difficult to use
  - Each has its own issues
  - Both too low-level
- Both widely used

OpenMP

- Easier to use
- Supports both data and task parallelism
- Much more limited in its support for data parallelism

70

Outline

1. State of the art
   - Pthreads
   - OpenMP
   - MPI

2. Moving forward ⬅
   - High level languages
   - PGAS languages
   - Global view abstractions

3. Modern languages
   - Chapel

71

Moving Forward

What are our goals?

- **Correctness:** Help programmers write correct code
- **Performance:** Help programmers write efficient code
- **Portability:** Help programmers write portable code

Is portability important?

- Long-lived software needs to run efficiently on many different platforms
- Why?
  - There's a wide variety of hardware platforms
  - Hardware continues to change rapidly

Why not just restrict ourselves to multi-core?

- As the number of cores per chip grows, the architecture needs to change
  - eg. On-chip latencies grow
  - eg. Cost of cache coherence grows with the number of cores

72

Recall: Premature Optimization

The root of all evil

- Requires manual changes to the application source code
- Embeds optimizations into the source code

Long term implications

- Complicates maintenance
- Defeats portability

What's the fundamental problem?

- MPI is too low level
- MPI over-specifies the communication
  - It specifies what to send, when to send it, and how to send it by specifying details of the implementation, such as the marshalling of data, synchronization, and buffering

Solution?

- Use a high level language (HLL)

73

Compiling High Level Languages

Strategy

- Compile to MPI to get portability

Problem

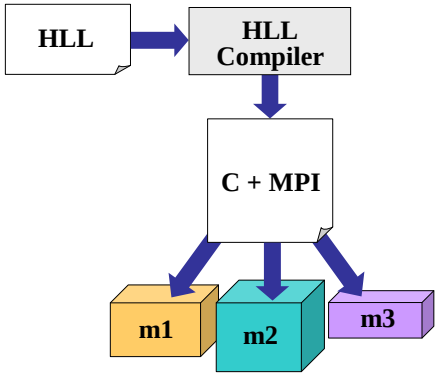
- Different machines favor the use of different MPI routines

74

Compiling Higher Level Languages (HLL)

Option 1: Portable compiler

– Compile to an intermediate language, such as C+MPI



Advantages

– Intermediate code is portable

– Compiler has a single backend

Disadvantages

– Favors portability over performance

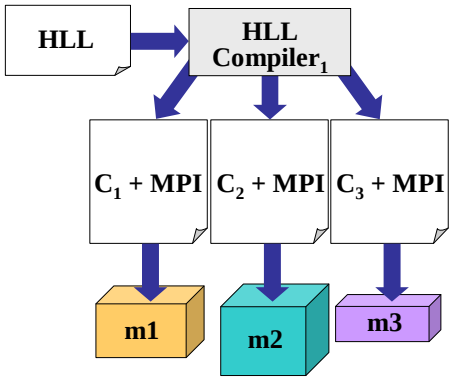
– We're still using the MPI interface, so we have the same performance portability problems that an MPI programmer faces

75

Compiling Higher Level Languages

Option 2: Machine-specific compiler

– Create multiple backends for multiple target platforms



Advantages

– Can exploit machine assumptions

Disadvantages

– Intermediate code is not portable

– Lots of work in building backends

How can we resolve this conflict between portability and performance?

76

Ironman Interface [Chamberlain, et al, 1997]

A communications interface used by ZPL

- A set of four calls which define constraints about possible communication
- Individually, each call has little meaning
- Collectively, they can be bound to different mechanisms for different machines

The name is not based on the comic book

- It's a reference to Strawman, Woodman, Tinman and Ironman, . . .

  which were different versions of the Ada language specification

77

The Ironman Interface

DR– Destination Ready

- Earliest point at which the destination can receive data

SR– Source Ready

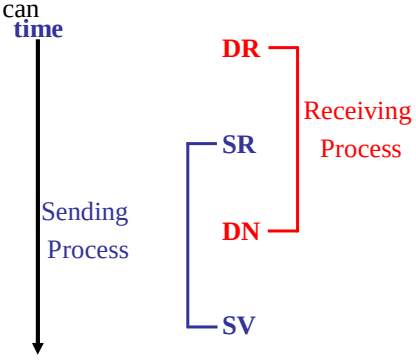
- Earliest point at which the sender can transmit data

DN– Destination Needed

- Latest point at which destination can receive data

SV– Sender Volatile

- Latest point by which data must be transmitted from the sender



78

The Ironman Interface (cont)

DR– Destination Ready

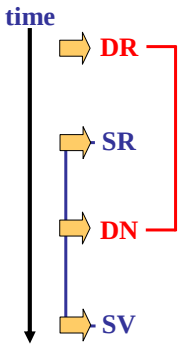
- Assuming that the destination receives data into a buffer, this receive cannot occur until the buffer has been allocated, and it cannot occur while the buffer's data is in use

SR– Source Ready

- The data cannot be sent until its been computed by the sender

DN– Destination Needed

- The point at which the destination needs to use the data that it's receiving



SV– Source Volatile

- If the sender is re-using the buffer, then this is the point at which the source's data is no longer valid

79

Example Bindings

Synchronous Sends

Effect at P_1	SPMD code	Effect at P_2
-	DR()	-
Send data from P_1	SR()	-
-	DN()	Receive data in P_2
-	SV()	-

Q: Can we bind DR() to a receive?

A: No. It would be legal from P_2's point of view, but it would cause deadlock in an SPMD program in which processes both send and receive data

80

Example Bindings II

Non-blocking Sends and non-blocking Receives

Effect at P_1	SPMD code	Effect at P_2
-	DR()	Non-blocking receive in P_2
Non-blocking send from P_1	SR()	-
-	DN()	Wait for receive at P_2
Wait for send to complete	SV()	-

81

Example Bindings III

User-Defined Callback Routines

Effect at P_1	SPMD code	Effect at P_2
Synchronize	DR()	Post receive callback
Send data	SR()	-
-	DN()	Wait for receive to complete
-	SV()	-

Usage

- This binding is similar to the use of non-blocking receives, but when the message is complete, a user-defined callback routine is called to un-marshall the data as it arrives

82

Example Bindings IV

One-sided Communication

Effect at P_1	SPMD code	Effect at P_2
Synchronize	DR()	Synchronize
Put data into destination	SR()	-
Synchronize	DN()	Synchronize
-	SV()	-

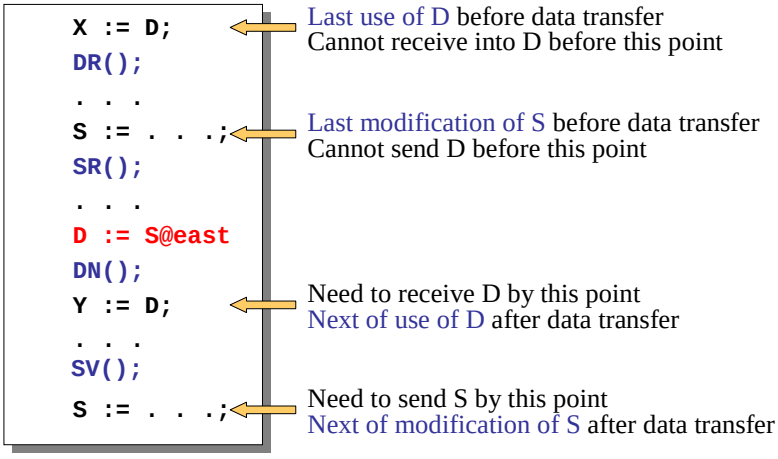
Usage

- Some hardware allows one processor to Put data onto another processor's memory
- This mechanism is one-sided because the destination process is not involved

83

Static Analysis– Identify Uses and Defs

Example HLL code



84

Static Analysis (cont)

Example HLL code

```
X := D;
DR();
. . .
S := . . .;
SR();
. . .
D := S@east;
DN();
Y := D;
. . .
SV();
S := . . .;
```

Overall compilation scheme

- Identify the need for communication
- Use dependence analysis to identify Defs and Uses, which define the four points of interest
- Perform code motion to push the four locations apart
- Assign static Communication Tags to each set of Ironman calls
  - These tags are used to maintain state across calls at runtime
- Insert parameters to each call

Array language semantics help by reducing control flow

85

Performance Summary for ZPL Using Ironman

Extra procedure call overhead

- Less than 1%

On an Intel Paragon

- Can use MPI, which maps well to Intel's NX message passing library

On the Cray T3E

- One-sided communication is 60-66% faster than MPI

Key benefit

- Ironman produces code that is both portable and efficient

86

The Larger Lessons?

Higher level languages

- Can use richer and more complicated interfaces
- No human would want to use the Ironman interface

Abstract interfaces

- Abstract interfaces can convey more information than lower-level interfaces
- Abstract interfaces can be both portable and efficient—but they need to convey the right information
- In the case of communication, they should specify what and when to transfer data and nothing more

87

Outline

1. State of the art

- Pthreads
- OpenMP
- MPI

2. Moving forward

- High level languages
- PGAS languages 
- Global view abstractions

3. Modern languages

- Chapel

88

High Level Languages

What should our HLL look like?

89

Observations

Global address space

- Makes it easier to reason about correctness

Partitioned data

- Essential for reasoning about locality, which is essential for obtaining good performance

PGAS languages

- Provide a Partitioned Global Address Space
- Offers the best of both worlds
- Raise the level of abstraction over Pthreads and MPI

90

First Generation PGAS Languages

Three languages

- Co-Array Fortran (CAF, formerly known as F--) http://www.co-array.org/

- Unified Parallel C (UPC) http://upc.lbl.gov/
- Titanium (Ti) Titanium http://titanium.cs.berkeley.edu/

Shared basic idea

- Extend a sequential language with support for distributed arrays
- Remove low-level communication details

Much more convenient than MPI

Co-Array Fortran (CAF)

Philosophy

- What is the smallest change required to make Fortran 90 an effective parallel language?

Distributed vs. local array

- Usual Fortran90 syntax
- Use square brackets to

4D data structure

- Distinguishes local data from remote data
- Provides a Fragmented View of data, so programmer must still make mental mapping to what should be a single 2D array

```
real, dimension(n,n)[p,*] :: a,b
do k=1,n
   do q=1,p
      a(i,j)[myP, myQ] = b(i,k)[myP,q]
```

Unified Parallel C

Basic idea

- Extends C
- Data can be either private or shared

```
int myCount;
shared int count;
```

- Four cases for pointers:
  - Private pointer pointing to private data
  - Private pointer pointing to shared data
  - Shared pointer pointing to private data
  - Shared pointer pointing to shared data

```
int *p1;
shared int *p2;
int *shared p3;
shared int *shared p4;
```

93

Titanium

Object oriented

- Extends Java
- Provides region-based memory allocation as well as garbage collected memory
- Restricts various other OO features

94

Limitations of First Generation PGAS Languages

Two issues

- They focus on array-based data-parallelism
  - No support for irregular data structures
  - No support for task-parallelism
- They provide a Fragmented View of data

95

Types of Parallelism

Data parallelism

- Create parallelism by dividing the data and performing roughly the same operations on each piece of the data

Task parallelism

- Create parallelism by dividing the functions and applying them to the data at the same time
- eg. pipeline parallelism

Data parallelism is often more scalable

- Example: The latest presidential inauguration
  - Cost: $160M
  - Solution: parallelize the cooking of the meals

96

Parallel Chefs

Data parallelism

– Scales with the number of guests

Task parallelism

– Scales with the number of tasks

We often want both

97

Limitations of First Generation PGAS Languages

Two issues

– They focus on array-based data-parallelism
  – No support for irregular data structures
  – No support for task-parallelism
– They provide a Fragmented View of data
  – Can we articulate the problems that this causes?

98

Outline

1. State of the art
   - Pthreads
   - OpenMP
   - MPI
2. Moving forward
   - High level languages
   - PGAS languages
   - Global view abstractions 
3. **Modern languages**
   - Chapel

Programmer Productivity

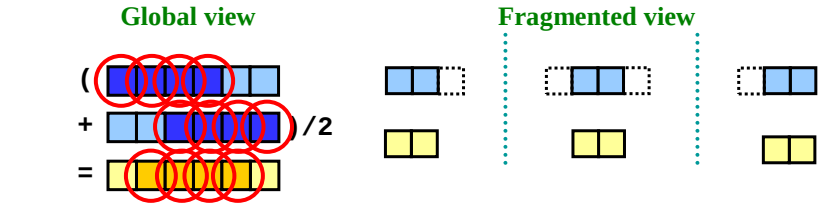
Global View abstractions

- Language constructs that produce the same result regardless of the number of processors that is used
- Allows programmers to debug sequentially
- Leads to more clear and concise code

Global View vs. Fragmented View

Example

– 3 point stencil of a vector



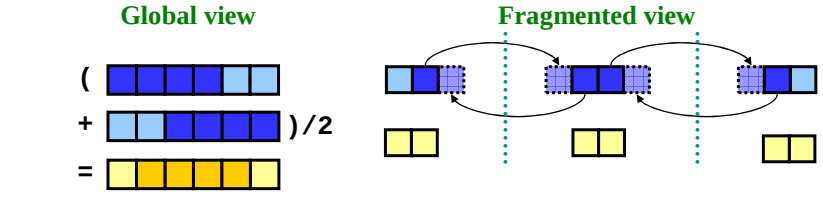

```
B = (A@east + A@west)/2;
```

101

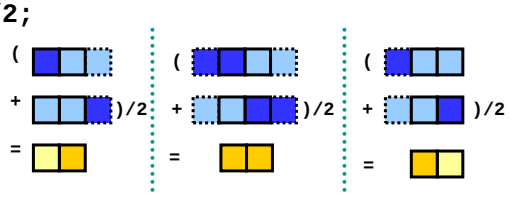
Global View vs. Fragmented View

Example

– 3 point stencil of a vector




```
B = (A@east + A@west)/2;
```



102

Global View vs. Fragmented View

Example

– 3 point stencil of a vector

Global View

```
begin
region R = [1..n];
var    n : int;
    A, B : [R] real;
procedure main()
[R] begin
    B := (A@west+A@east)/2;
    end;
end;
```

Fragmented View

```
def main() {
  var n: int = 1000;
  var locN: int = n/numProcs;
  var a, b: [0..locN+1] real;

  if (iHaveRightNeighbor) {
    send(right, a(locN));
    recv(right, a(locN+1));
  }
  if (iHaveLeftNeighbor) {
    send(left, a(1));
    recv(left, a(0));
  }
  forall i in 1..locN {
    b(i) = (a(i-1) + a(i+1))/2;
  }
}
```

103

Global View vs. Fragmented View

Assumes *numProcs* divides *n*; a more general version would require additional effort

Example

– 3 point stencil of a vector

Global View

```
begin
region R = [1..n];
var    n : int;
    A, B : [R] real;
procedure main()
[R] begin
    B := (A@west+A@east)/2;
    end;
end;
```

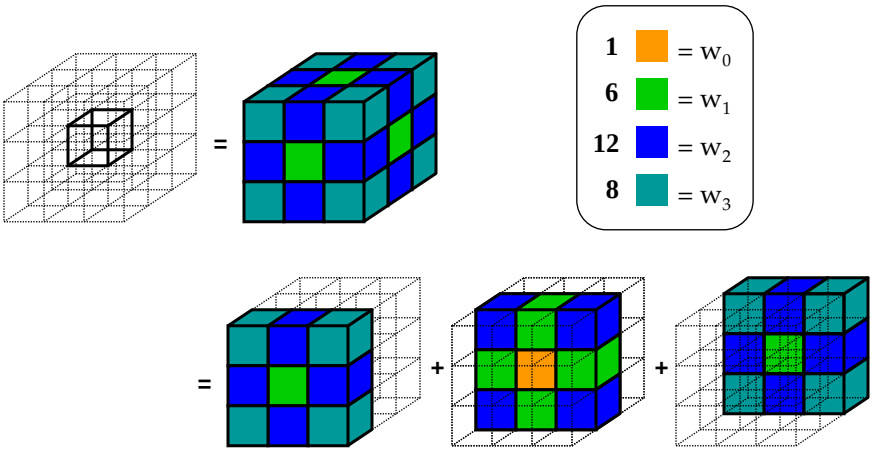
Fragmented View

```
def main() {
  var n: int = 1000;
  var locN: int = n/numProcs;
  var a, b: [0..locN+1] real;
  var innerLo: int = 1;
  var innerHi: int = locN;

  if (iHaveRightNeighbor) {
    send(right, a(locN));
    recv(right, a(locN+1));
  } else {
    innerHi = locN-1;
  }
  if (iHaveLeftNeighbor) {
    send(left, a(1));
    recv(left, a(0));
  } else {
    innerLo = 2;
  }
  forall i in innerLo..innerHi {
    b(i) = (a(i-1) + a(i+1))/2;
  }
}
```

Communication becomes geometrically more complex for higher-dimensional arrays

104

Consider the *rprj3* stencil from NAS MG



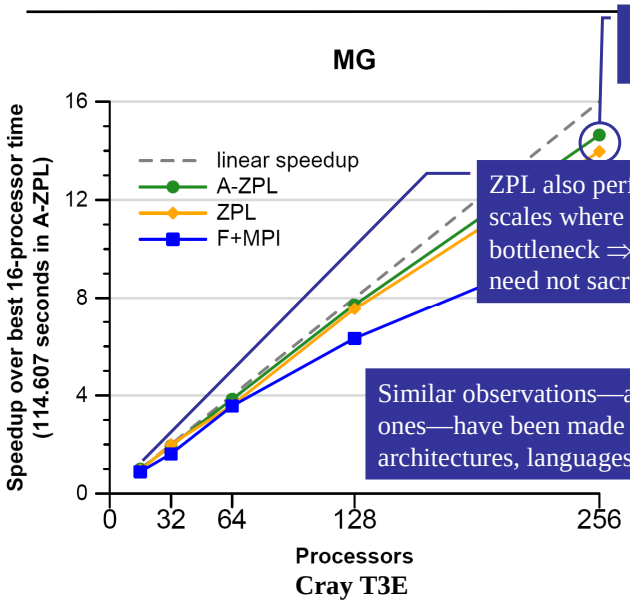
105

NAS MG *rprj3* stencil in Fortran+MPI

```
subroutine comm3(u,n1,n2,n3,kk)
use caf_intrinsics

implicit none

include 'cafmpb.h'
include 'globals.h'

integer n1, n2, n3, kk
double precision u(n1,n2,n3)
integer axis

if( .not. dead(kk) )then
   do axis = 1, 3
      if( nprocs(axis) .ne. 1) then
         call sync_all()
         call give3( axis, +1, u, n1, n2, n3, kk )
         call give3( axis, -1, u, n1, n2, n3, kk )
         call sync_all()
         call take3( axis, -1, u, n1, n2, n3 )
         call take3( axis, +1, u, n1, n2, n3 )
      else
         call comm1p( axis, u, n1, n2, n3, kk )
      endif
   enddo
else
   do  axis = 1, 3
       call sync_all()
       call sync_all()
   enddo
   call zero3(u,n1,n2,n3)
endif
return
end


subroutine give3( axis, dir, u, n1, n2, n3, k )
use caf_intrinsics


implicit none

include 'cafmpb.h'
include 'globals.h'

integer axis, dir, n1, n2, n3, k, ierr
double precision u( n1, n2, n3 )

integer i3, i2, i1, buff_len,buff_id

buff_id = 2 + dir
buff_len = 0

if( axis .eq. 1 )then
   if( dir .eq. -1 )then

      do  i3=2,n3-1
         do  i2=2,n2-1
            buff_len = buff_len + 1
            buff(buff_len,buff_id ) = u( 2,  i2,i3)
         enddo
      enddo

      buff(1:buff_len,buff_id+1)[nbr(axis,dir,k)] =
>     buff(1:buff_len,buff_id)

   else if( dir .eq. +1 ) then

      do  i3=2,n3-1
         do  i2=2,n2-1
            buff_len = buff_len + 1
            buff(buff_len, buff_id ) = u( n1-1, i2,i3)
         enddo
      enddo

      buff(1:buff_len,buff_id+1)[nbr(axis,dir,k)] =
>     buff(1:buff_len,buff_id)

   endif
endif

if( axis .eq.  2 )then
   if( dir .eq. -1 )then
      do  i3=2,n3-1
         do  i1=1,n1
            buff_len = buff_len + 1
            buff(buff_len, buff_id ) = u( i1,  2,i3)
         enddo
      enddo

  buff(1:buff_len,buff_id+1)[nbr(axis,dir,k)] =
>     buff(1:buff_len,buff_id)

else if( dir .eq. +1 ) then

      do  i3=2,n3-1
         do  i1=1,n1
            buff_len = buff_len + 1
            buff(buff_len,  buff_id )= u( i1,n2-1,i3)
         enddo
      enddo

      buff(1:buff_len,buff_id+1)[nbr(axis,dir,k)] =
>     buff(1:buff_len,buff_id)

   endif
endif

if( axis .eq.  3 )then
   if( dir .eq. -1 )then

      do  i2=1,n2
         do  i1=1,n1
            buff_len = buff_len + 1
            buff(buff_len, buff_id ) = u( i1,i2,2)
         enddo
      enddo

      buff(1:buff_len,buff_id+1)[nbr(axis,dir,k)] =
>     buff(1:buff_len,buff_id)

   else if( dir .eq. +1 ) then

      do  i2=1,n2
         do  i1=1,n1
            buff_len = buff_len + 1
            buff(buff_len, buff_id ) = u( i1,i2,n3-1)
         enddo
      enddo

      buff(1:buff_len,buff_id+1)[nbr(axis,dir,k)] =
>     buff(1:buff_len,buff_id)

   endif
endif

return
end


subroutine take3( axis, dir, u, n1, n2, n3 )
use caf_intrinsics

implicit none

include 'cafmpb.h'
include 'globals.h'

integer axis, dir, n1, n2, n3
double precision u( n1, n2, n3 )

integer buff_id, indx

integer i3, i2, i1

buff_id = 3 + dir
indx = 0

if( axis .eq.  1 )then
   if( dir .eq. -1 )then

      do  i3=2,n3-1
         do  i2=2,n2-1
            indx = indx + 1
            u(n1,i2,i3) = buff(indx, buff_id )
         enddo
      enddo

   else if( dir .eq. +1 ) then

      do  i3=2,n3-1
         do  i2=2,n2-1
            indx = indx + 1
            u(1,i2,i3) = buff(indx, buff_id )
         enddo
      enddo

   endif
endif

if( axis .eq.  2 )then
   if( dir .eq. -1 )then

      do  i3=2,n3-1
         do  i1=1,n1
            indx = indx + 1
            u(i1,n2,i3) = buff(indx, buff_id )
         enddo
      enddo

   else if( dir .eq. +1 ) then

      do  i3=2,n3-1
         do  i1=1,n1
            indx = indx + 1
            u(i1,1,i3) = buff(indx, buff_id )
         enddo
      enddo

   endif
endif

if( axis .eq.  3 )then
   if( dir .eq. -1 )then

      do  i2=1,n2
         do  i1=1,n1
            indx = indx + 1
            u(i1,i2,n3) = buff(indx, buff_id )
         enddo
      enddo

   else if( dir .eq. +1 ) then

      do  i2=1,n2
         do  i1=1,n1
            indx = indx + 1
            u(i1,i2,1) = buff(indx, buff_id )
         enddo
      enddo

   endif
endif

return
end

subroutine comm1p( axis, u, n1, n2, n3, kk )
use caf_intrinsics


implicit none

include 'cafmpb.h'
include 'globals.h'

integer axis, dir, n1, n2, n3
double precision u( n1, n2, n3 )

integer i3, i2, i1, buff_len,buff_id
integer i, kk, indx

dir = -1

buff_id = 3 + dir
buff_len = nm2

do  i=1,nm2
   buff(i,buff_id) = 0.0D0
enddo

dir = +1

buff_id = 3 + dir
buff_len = nm2

do  i=1,nm2
   buff(i,buff_id) = 0.0D0
enddo

dir = +1

buff_id = 2 + dir
buff_len = 0

if( axis .eq.  1 )then
   do  i3=2,n3-1
      do  i2=2,n2-1
         buff_len = buff_len + 1
         buff(buff_len, buff_id ) = u( n1-1,
     i2,i3)
      enddo
   enddo
endif

if( axis .eq.  2 )then
   do  i3=2,n3-1
      do  i1=1,n1
         buff_len = buff_len + 1
         buff(buff_len,  buff_id )= u( i1,n2-
     1,i3)
      enddo
   enddo
endif

if( axis .eq.  3 )then
   do  i2=1,n2
      do  i1=1,n1
         buff_len = buff_len + 1
         buff(buff_len, buff_id ) = u( i1,i2,n3-
     1)
      enddo
   enddo
endif

dir = -1

buff_id = 2 + dir
buff_len = 0

if( axis .eq.  1 )then
   do  i3=2,n3-1
      do  i2=2,n2-1
         buff_len = buff_len + 1
         buff(buff_len,buff_id ) = u( 2,  i2,i3)
      enddo
   enddo
endif

if( axis .eq.  2 )then
   do  i3=2,n3-1
      do  i1=1,n1
         buff_len = buff_len + 1
         buff(buff_len, buff_id ) = u( i1,
     2,i3)
      enddo
   enddo
endif

if( axis .eq.  3 )then
   do  i2=1,n2
      do  i1=1,n1
         buff_len = buff_len + 1
         buff(buff_len, buff_id ) = u( i1,i2,2)
      enddo
   enddo
endif

do  i=1,nm2
   buff(i,4) = buff(i,3)
   buff(i,2) = buff(i,1)
enddo

dir = -1

buff_id = 3 + dir
indx = 0

if( axis .eq.  1 )then
   do  i3=2,n3-1
      do  i2=2,n2-1
         indx = indx + 1
         u(n1,i2,i3) = buff(indx, buff_id )
      enddo
   enddo
endif

if( axis .eq.  2 )then
   do  i3=2,n3-1
      do  i1=1,n1
         indx = indx + 1
         u(i1,n2,i3) = buff(indx, buff_id )
      enddo
   enddo
endif

if( axis .eq.  3 )then
   do  i2=1,n2
      do  i1=1,n1
         indx = indx + 1
         u(i1,i2,n3) = buff(indx, buff_id )
      enddo
   enddo
endif

dir = +1

buff_id = 3 + dir
indx = 0

if( axis .eq.  1 )then
   do  i3=2,n3-1
      do  i2=2,n2-1
         indx = indx + 1
         u(1,i2,i3) = buff(indx, buff_id )
      enddo
   enddo
endif

if( axis .eq.  2 )then
   do  i3=2,n3-1
      do  i1=1,n1
         indx = indx + 1
         u(i1,1,i3) = buff(indx, buff_id )
      enddo
   enddo
endif

if( axis .eq.  3 )then
   do  i2=1,n2
      do  i1=1,n1
         indx = indx + 1
         u(i1,i2,1) = buff(indx, buff_id )
      enddo
   enddo
endif

if(mik.eq.3)then
   di = 2
else
   di = 1
endif

if(mjk.eq.3)then
   dj = 2
else
   dj = 1
endif

if(mik.eq.3)then
   i3 = 2*i3-d3
   do  i2=2,m2j-1
      i2 = 2*j2-d2
      do i1=2,m1j
         i1 = 2*j1-d1
         x1(i1-1) = r(i1-1,i2-1,i3  ) + r(i1-1,i2+1,i3  )
>              + r(i1-1,i2,  i3-1) + r(i1-1,i2,  i3+1)
         y1(i1-1) = r(i1-1,i2-1,i3-1) + r(i1-1,i2-1,i3+1)
>              + r(i1-1,i2+1,i3-1) + r(i1-1,i2+1,i3+1)
      enddo
      do  j1=2,m1j-1
         i1 = 2*j1-d1
         y2 = r(i1,  i2-1,i3-1) + r(i1,  i2-1,i3+1)
>             + r(i1,  i2+1,i3-1) + r(i1,  i2+1,i3+1)
         x2 = r(i1,  i2-1,i3  ) + r(i1,  i2+1,i3  )
>             + r(i1,  i2,  i3-1) + r(i1,  i2,  i3+1)
         x(j1,j2,j3) =
>             0.5D0 * r(i1,i2,i3)
>          + 0.25D0 * (r(i1-1,i2,i3) + r(i1+1,i2,i3) + x2)
>          + 0.125D0 * ( x1(i1-1) + x1(i1+1) + y2)
>          + 0.0625D0 * ( y1(i1-1) + y1(i1+1) )
      enddo
   enddo
enddo
j = k-1
call comm3(x,m1j,m2j,m3j,j)
return
end
```

106

Performance Notes



Here we see the advantage of ZPL's Ironman interface

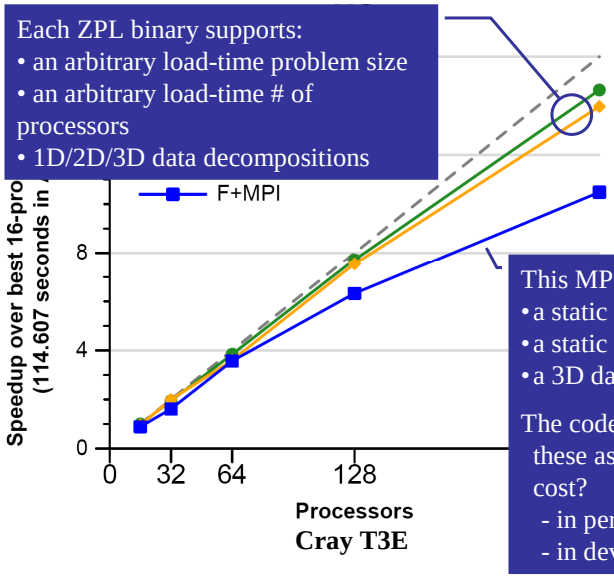
ZPL also performs better at smaller scales where communication is not the bottleneck ⇒ higher-level languages need not sacrifice performance

Similar observations—and more dramatic ones—have been made using more recent architectures, languages, and benchmarks

107

Generality Notes

Each ZPL binary supports:
- an arbitrary load-time problem size
- an arbitrary load-time # of processors
- 1D/2D/3D data decompositions



This MPI binary only supports:
- a static 2^k problem size
- a static 2^j # of processors
- a 3D data decomposition

The code could be rewritten to relax these assumptions, but at what cost?
- in performance?
- in development effort?

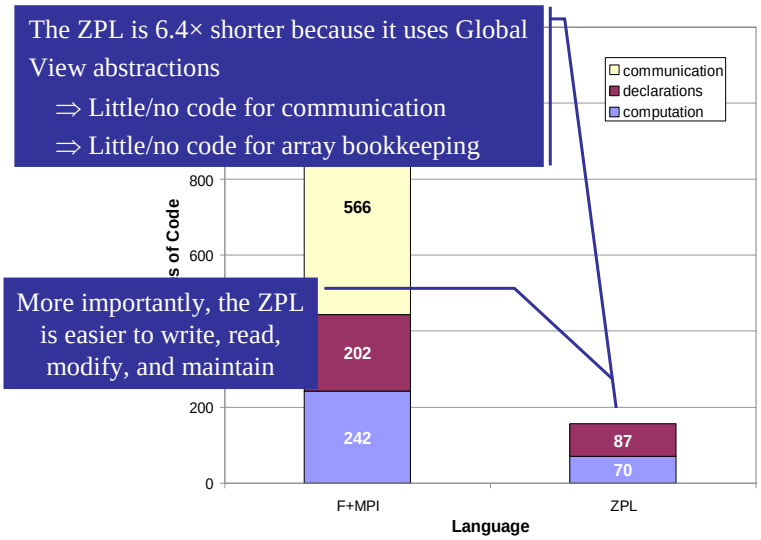
Code Size Notes

The ZPL is 6.4× shorter because it uses Global View abstractions

⇒ Little/no code for communication

⇒ Little/no code for array bookkeeping

More importantly, the ZPL is easier to write, read, modify, and maintain



109

Critiquing ZPL

Strengths

- Concise
- Global View abstractions
- Helps programmers reason about performance

Weaknesses

- Focuses on regular data-parallelism
- Is it too restrictive?
- Unfamiliar to many programmers

110

Outline

1. State of the art
   - Pthreads
   - OpenMP
   - MPI
2. Moving forward
   - High level languages
   - PGAS languages
   - Global view abstractions
3. **Modern languages**
   - Chapel 

Chapel Goals

Goals

- Support general parallel programming
- Provide global view abstractions
- Provide support for locality
- Reduce gap between mainstream languages and parallel languages

History– High Productivity Computing Systems Program

DARPA HPCS Program (2002)

- Realization that programmer productivity is critical
- Sought to increase programmer productivity by 10× by 2010
  - Productivity = Performance + Programmability + Portability + Robustness

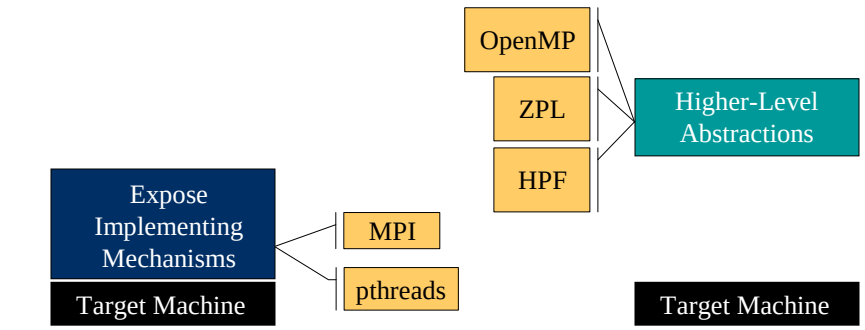
Includes both hardware and language design

Last two languages standing

- Chapel (Cray)
- X10 (IBM)

113

Previous Languages– Two Extremes



"Why is everything so painful?"

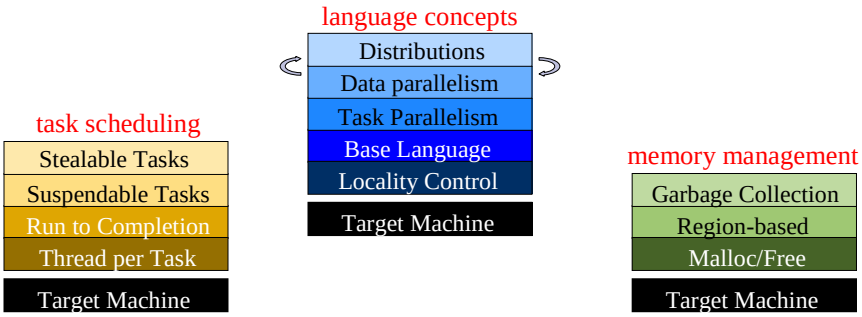
"Why do my hands feel tied?"

114

Multi-Resolution Language Design

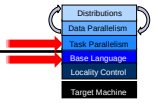
The Chapel approach

- Allow the language to be used at multiple levels of abstraction
  - Provide high-level features and automation for convenience
  - Provide the ability to drop down to lower, more manual levels
  - Use appropriate separation of concerns to keep these layers clean



115

Chapel In a Nutshell



Rich base language

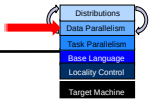
- **Standard stuff:** types, expressions, statements, functions, modules
- **Object-orientation:** value- and reference-based classes (optional)
- **Iterators:** functions that generate a stream of return values
- **Latent types:** ability to omit types of variables, arguments, etc.

Task parallelism

- **Task creation:** structured and unstructured task creation
- **Synchronization:** through sync variables, transactional memory

116

Chapel In a Nutshell (cont)



Data parallelism

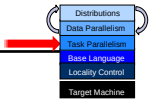
- **Data structures:** global view of dense, sparse, associative arrays
- **Operators:** forall loops, promotion of scalar operators/functions, …

Locality

- **Locales:** language concept for reasoning about machine locality
- **On clauses:** ability to place tasks, variables on specific locales
- **Distributions:** recipes for implementing distributed arrays on locales

117

Task Parallelism



Basic features

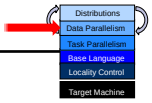
- Spawn a task
  - `Begin`, `cobegin`, `coforall`
- Synchronize tasks
  - Synch, full/empty bits, atomic sections, . . .

Example of task creation:

```
begin DoThisTask();
WhileContinuing();
TheOriginalThread();
```

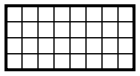
118

Domains



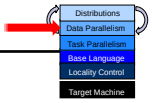
domain: a first-class index set

```
var m = 4, n = 8;

var D: domain(2) = [1..m, 1..n];
```



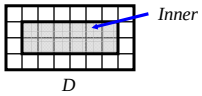
D

Domains



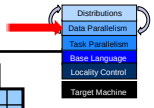
domain: a first-class index set

```
var m = 4, n = 8;

var D: domain(2) = [1..m, 1..n];
var Inner: subdomain(D) = [2..m-1, 2..n-1];
```



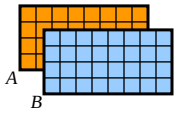
D

Domains: Some Uses



Declaring arrays:

```
var A, B: [D] real;
```

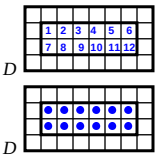


Iteration (sequential or parallel):

```
for    ij in Inner { … }
```

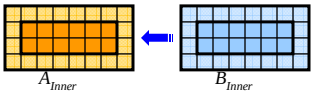
or:

```
forall ij in Inner { … }
```



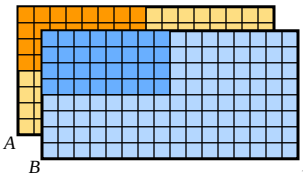
Array Slicing:

```
A[Inner] = B[Inner];
```



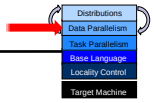
Array reallocation:

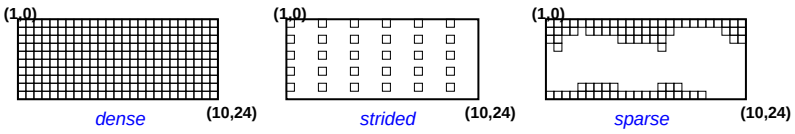
```
D = [1..2*m, 1..2*n];
```

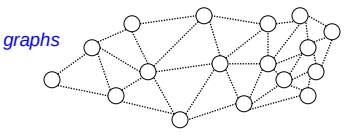


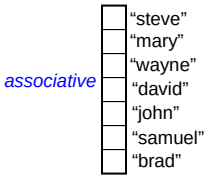
121

Data Parallelism: Other Domains



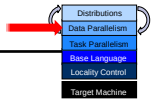




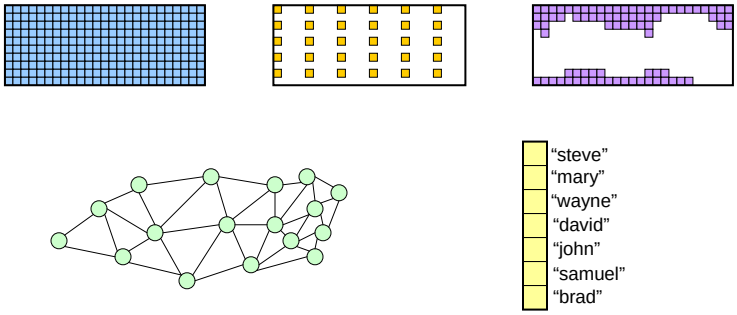


122

Data Parallelism: Domain Uses

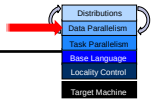


Domains are used to declare arrays…



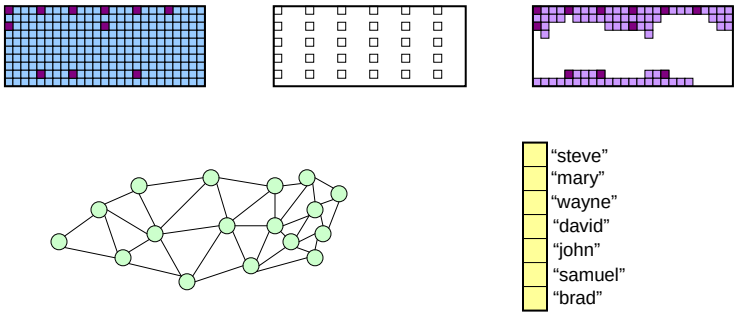
123

Data Parallelism: Domain Uses



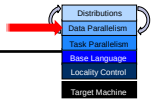
…to iterate over index sets…

```
forall ij in StrDom {
  DnsArr(ij) += SpsArr(ij);
}
```



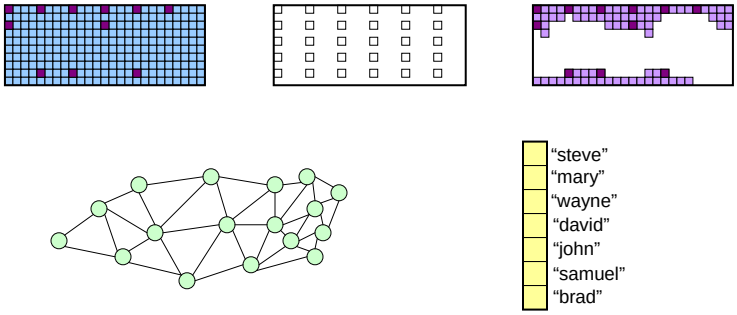
124

Data Parallelism: Domain Uses



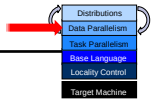
…to slice arrays…

```
DnsArr[StrDom] += SpsArr[StrDom];
```



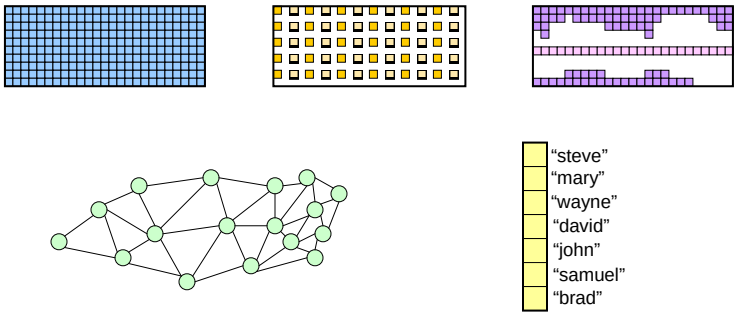
125

Data Parallelism: Domain Uses



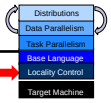
…and to reallocate arrays

```
StrDom = DnsDom by (2,2);
SpsDom += genEquator();
```



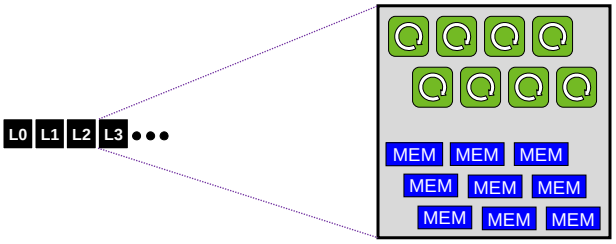
126

Locality: Locales



locale: **architectural unit of locality**

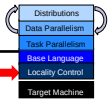
- Represents both a processor and local memory
- Threads within a locale have ~uniform access to local memory
- Memory within other locales is accessible, but at a price
- *e.g.*, a multicore processor or SMP node could be a locale



- Programmers can declare, manipulate, and use locales

127

Locality: Task Placement



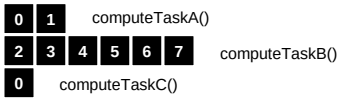
on clauses: **indicate where tasks should execute**

Either in a data-driven manner…

```
computePivot(lo, hi, data);
cobegin {
  on data(lo)    do Quicksort(lo, pivot, data);
  on data(pivot) do Quicksort(pivot, hi, data);
}
```

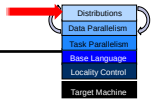
…or by naming locales explicitly

```
cobegin {
  on TaskALocs do computeTaskA(…);
  on TaskBLocs do computeTaskB(…);
  on Locales(0) do computeTaskC(…);
}
```



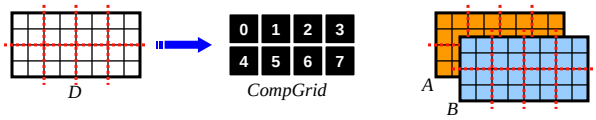
128

Locality: Domain Distribution



Domains may be distributed across locales

```
var D: domain(2) distributed Block on CompGrid = …;
```



A distribution implies…

…ownership of the domain's indices (and its arrays' elements)

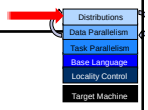
…the default work ownership for operations on the domains/arrays

Chapel provides…

…a standard library of distributions (Block, Recursive Bisection, …)

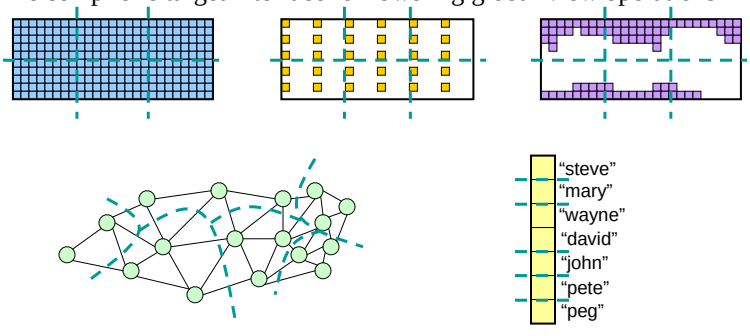
…the means for advanced users to author their own distributions

Locality: Domain Distributions

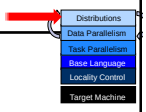


A distribution must implement

- The mapping from indices to locales
- The per-locale representation of domain indices and array elements
- The compiler's target interface for lowering global-view operations



Locality: Distributions Overview



Distributions define a mapping

- From the user's global view operations to the fragmented implementation for a distributed memory machine

Users can implement custom distributions

- Written using task parallel features, on clauses, domains/arrays
- Must implement standard interface:
  - **Allocation/reallocation** of domain indices and array elements
  - **Mapping functions** (*e.g.*, index-to-locale, index-to-value)
  - **Iterators:** parallel/serial × global/local
  - Optionally, communication idioms

Chapel's standard library of distributions

- Written using the same mechanism as user-defined distributions
- Tuned for different platforms to maximize performance

131

Distributions vs. Domains

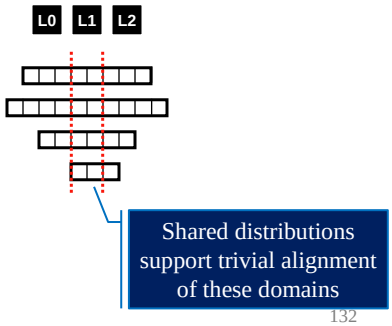
Q1: Why distinguish between distributions and domains?

Q2: Why do distributions map an index *space* rather than a fixed index set?

A: To permit several domains to share a single distribution

- Amortizes the overheads of storing a distribution
- Supports trivial domain/array alignment and compiler optimizations

```
const D     : …distributed B1 = [1..8],
      outerD: …distributed B1 = [0..9],
      innerD: subdomain(D)    = [2..7],
      slideD: subdomain(D)    = [4..6];
```



132

Distributions vs. Domains

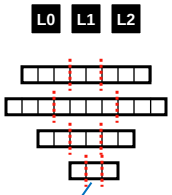
Q1: Why distinguish between distributions and domains?

Q2: Why do distributions map an index *space* rather than a fixed index set?

A: To permit several domains to share a single distribution

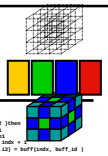
- Amortizes the overheads of storing a distribution
- Supports trivial domain/array alignment and compiler optimizations

```
const D     : …distributed B1 = [1..8],
      outerD: …distributed B1 = [0..9],
      innerD: subdomain(D)    = [2..7],
      slideD: subdomain(D)    = [4..6];
```



When each domain is given its own distribution, the compiler cannot reason about alignment of indices

Recall the NAS MG *rprj3* stencil in Fortran+MPI

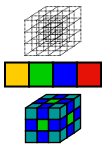


```
subroutine comm3(u,n1,n2,n3,kk)
use caf_intrinsics

implicit none

include 'cafmpb.h'
include 'globals.h'

integer n1, n2, n3, kk
double precision u(n1,n2,n3)
integer axis

if( .not. dead(kk) )then
   do axis = 1, 3
      if( nprocs(axis) .ne. 1) then
         call give3( axis, +1, u, n1, n2, n3, kk )
         call give3( axis, -1, u, n1, n2, n3, kk )
         call sync_all()
         call take3( axis, -1, u, n1, n2, n3 )
         call take3( axis, +1, u, n1, n2, n3 )
      else
         call comm1p( axis, u, n1, n2, n3, kk )
      endif
   enddo
else
   do  axis = 1, 3
      call sync_all()
      call sync_all()
   enddo
   call zero3(u,n1,n2,n3)
endif
returns
end


subroutine give3( axis, dir, u, n1, n2, n3, k )
use caf_intrinsics


implicit none

include 'cafmpb.h'
include 'globals.h'

integer axis, dir, n1, n2, n3, k, ierr
double precision u( n1, n2, n3 )

integer i3, i2, i1, buff_len,buff_id

buff_id = 2 + dir
buff_len = 0

if( axis .eq. 1 )then
   if( dir .eq. -1 )then

      do  i3=2,n3-1
         do  i2=2,n2-1
            buff_len = buff_len + 1
            buff(buff_len,buff_id ) = u( 2,  i2,i3)
         enddo
      enddo

      buff(1:buff_len,buff_id+1)[nbr(axis,dir,k)] =
>     buff(1:buff_len,buff_id)

   else if( dir .eq. +1 ) then

      do  i3=2,n3-1
         do  i2=2,n2-1
            buff_len = buff_len + 1
            buff(buff_len, buff_id ) = u( n1-1, i2,i3)
         enddo
      enddo

      buff(1:buff_len,buff_id+1)[nbr(axis,dir,k)] =
>     buff(1:buff_len,buff_id)

   endif
endif

if( axis .eq.  2 )then
   if( dir .eq. -1 )then
      do  i3=2,n3-1
         do  i1=1,n1
            buff_len = buff_len + 1
            buff(buff_len, buff_id ) = u( i1,  2,i3)
         enddo
      enddo

  buff(1:buff_len,buff_id+1)[nbr(axis,dir,k)] =
>     buff(1:buff_len,buff_id)

else if( dir .eq. +1 ) then

      do  i3=2,n3-1
         do  i1=1,n1
            buff_len = buff_len + 1
            buff(buff_len, buff_id ) = u( i1,i2,n2-1)
         enddo
      enddo

      buff(1:buff_len,buff_id+1)[nbr(axis,dir,k)] =
>     buff(1:buff_len,buff_id)

   endif
endif

return
end


subroutine take3( axis, dir, u, n1, n2, n3 )
use caf_intrinsics

implicit none

include 'cafmpb.h'
include 'globals.h'

integer axis, dir, n1, n2, n3
double precision u( n1, n2, n3 )

integer buff_id, indx

integer i3, i2, i1

buff_id = 3 + dir
indx = 0

if( axis .eq.  1 )then
   if( dir .eq. -1 )then

      do  i3=2,n3-1
         do  i2=2,n2-1
            indx = indx + 1
            u(n1,i2,i3) = buff(indx, buff_id )
         enddo
      enddo

   else if( dir .eq. +1 ) then

      do  i3=2,n3-1
         do  i2=2,n2-1
            indx = indx + 1
            u(1,i2,i3) = buff(indx, buff_id )
         enddo
      enddo

   endif
endif

if( axis .eq.  2 )then
   if( dir .eq. -1 )then

      do  i3=2,n3-1
         do  i1=1,n1
            indx = indx + 1
            u(i1,n2,i3) = buff(indx, buff_id )
         enddo
      enddo

   else if( dir .eq. +1 ) then

      do  i3=2,n3-1
         do  i1=1,n1
            indx = indx + 1
            u(i1,1,i3) = buff(indx, buff_id )
         enddo
      enddo

   endif
endif

if( axis .eq.  3 )then
   if( dir .eq. -1 )then

      do  i2=1,n2
         do  i1=1,n1
            indx = indx + 1
            u(i1,i2,n3) = buff(indx, buff_id )
         enddo
      enddo

   else if( dir .eq. +1 ) then

      do  i2=1,n2
         do  i1=1,n1
            indx = indx + 1
            u(i1,i2,1) = buff(indx, buff_id )
         enddo
      enddo

   endif
endif

return
end

subroutine comm1p( axis, u, n1, n2, n3, kk )
use caf_intrinsics


implicit none

include 'cafmpb.h'
include 'globals.h'

integer axis, dir, n1, n2, n3
double precision u( n1, n2, n3 )

integer i3, i2, i1, buff_len,buff_id
integer i, kk, indx

dir = -1

buff_id = 3 + dir
buff_len = nm2

do  i=1,nm2
   buff(i,buff_id) = 0.0D0
enddo

dir = +1

buff_id = 3 + dir
buff_len = nm2

do  i=1,nm2
   buff(i,buff_id) = 0.0D0
enddo

dir = +1

buff_id = 2 + dir
buff_len = 0

if( axis .eq.  1 )then
   if( dir .eq. -1 )then

      do  i3=2,n3-1
         do  i2=2,n2-1
            buff_len = buff_len + 1
            buff(buff_len, buff_id ) = u( 2,  i2,i3)
         enddo
      enddo

   else if( dir .eq. +1 ) then

      do  i3=2,n3-1
         do  i2=2,n2-1
            buff_len = buff_len + 1
            buff(buff_len, buff_id ) = u( n1-1,
     i2,i3)
         enddo
      enddo

   endif
endif

if( axis .eq.  2 )then
   if( dir .eq. -1 )then

      do  i3=2,n3-1
         do  i1=1,n1
            buff_len = buff_len + 1
            buff(buff_len, buff_id ) = u( i1,  2,i3)
         enddo
      enddo

   else if( dir .eq. +1 ) then

      do  i3=2,n3-1
         do  i1=1,n1
            buff_len = buff_len + 1
            buff(buff_len, buff_id ) = u( i1,n2-1,
     i3)
         enddo
      enddo

   endif
endif

if( axis .eq.  3 )then
   if( dir .eq. -1 )then

      do  i2=1,n2
         do  i1=1,n1
            buff_len = buff_len + 1
            buff(buff_len, buff_id ) = u( i1,i2,  2)
         enddo
      enddo

   else if( dir .eq. +1 ) then

      do  i2=1,n2
         do  i1=1,n1
            buff_len = buff_len + 1
            buff(buff_len, buff_id ) = u( i1,i2,n3-1)
         enddo
      enddo

   endif
endif

dir = -1

buff_id = 2 + dir
buff_len = 0

if( axis .eq.  1 )then
   do  i3=2,n3-1
      do  i2=2,n2-1
         indx = indx + 1
         buff(indx,buff_id ) = u( 2,  i2,i3)
      enddo
   enddo
endif

if( axis .eq.  2 )then
   do  i3=2,n3-1
      do  i1=1,n1
         indx = indx + 1
         buff(indx,buff_id ) = u( i1,  2,i3)
      enddo
   enddo
endif

if( axis .eq.  3 )then
   do  i2=1,n2
      do  i1=1,n1
         indx = indx + 1
         buff(indx,buff_id ) = u( i1,i2,  2)
      enddo
   enddo
endif

dir = -1

buff_id = 3 + dir
indx = 0

if( axis .eq.  1 )then
   do  i3=2,n3-1
      do  i2=2,n2-1
         indx = indx + 1
         buff(indx,buff_id ) = u( 2,  i2,i3)
      enddo
   enddo
endif

if( axis .eq.  2 )then
   do  i3=2,n3-1
      do  i1=1,n1
         indx = indx + 1
         buff(indx,buff_id ) = u( i1,  2,i3)
      enddo
   enddo
endif

if( axis .eq.  3 )then
   do  i2=1,n2
      do  i1=1,n1
         indx = indx + 1
         buff(indx,buff_id ) = u( i1,i2,  2)
      enddo
   enddo
endif

dir = +1

buff_id = 3 + dir
indx = 0

if( axis .eq.  1 )then
   do  i3=2,n3-1
      do  i2=2,n2-1
         indx = indx + 1
         buff(indx,buff_id ) = u( i1,i2,n3-
     1)
      enddo
   enddo
endif

dir = -1

buff_id = 2 + dir
buff_len = 0

if( axis .eq.  1 )then
   do  i3=2,n3-1
      do  i2=2,n2-1
         buff_len = buff_len + 1
         buff(buff_len,buff_id ) = u( 2,  i2,i3)
      enddo
   enddo
endif

if( axis .eq.  2 )then
   do  i3=2,n3-1
      do  i1=1,n1
         buff_len = buff_len + 1
         buff(buff_len,buff_id ) = u( i1,i2,
     2,i3)
      enddo
   enddo
endif

if( axis .eq.  3 )then
   do  i2=1,n2
      do  i1=1,n1
         buff_len = buff_len + 1
         buff(buff_len,buff_id ) = u( i1,i2,2)
      enddo
   enddo
endif

do  i=1,nm2
   buff(i,4) = buff(i,3)
   buff(i,2) = buff(i,1)
enddo

dir = -1

buff_id = 3 + dir
indx = 0

if( axis .eq.  1 )then
   do  i3=2,n3-1
      do  i2=2,n2-1
         indx = indx + 1
         buff(indx,i2,i3) = buff(indx, buff_id )
      enddo
   enddo
endif

if( axis .eq.  2 )then
   do  i3=2,n3-1
      do  i1=1,n1
         indx = indx + 1
         buff(indx,i1,i3) = buff(indx, buff_id )
      enddo
   enddo
endif

if( axis .eq.  3 )then
   do  i2=1,n2
      do  i1=1,n1
         indx = indx + 1
         buff(indx,i1,i2) = buff(indx, buff_id )
      enddo
   enddo
endif

dir = +1

buff_id = 3 + dir
indx = 0

if( axis .eq.  1 )then
   do  i3=2,n3-1
      do  i2=2,n2-1
         indx = indx + 1
         buff(indx,i2,i3) = buff(indx, buff_id )
      enddo
   enddo
endif

if( axis .eq.  2 )then
   do  i3=2,n3-1
      do  i1=1,n1
         indx = indx + 1
         buff(indx,i1,i3) = buff(indx, buff_id )
      enddo
   enddo
endif

if( axis .eq.  3 )then
   do  i2=1,n2
      do  i1=1,n1
         indx = indx + 1
         buff(indx,i1,i2) = buff(indx, buff_id )
      enddo
   enddo
endif

dir = +1

buff_id = 2 + dir
buff_len = 0

if( axis .eq.  1 )then
   do  i3=2,n3-1
      do  i2=2,n2-1
         buff_len = buff_len + 1
         buff(buff_len,buff_id ) = u( i1,i2,n3-
     1)
      enddo
   enddo
endif

dir = +1

buff_id = 3 + dir
indx = 0

if( axis .eq.  1 )then
   do  i3=2,n3-1
      do  i2=2,n2-1
         indx = indx + 1
         buff(indx,buff_id ) = u( 2,  i2,i3)
      enddo
   enddo
endif

if( axis .eq.  2 )then
   do  i3=2,n3-1
      do  i1=1,n1
         indx = indx + 1
         buff(indx,buff_id ) = u( i1,  2,i3)
      enddo
   enddo
endif

if( axis .eq.  3 )then
   do  i2=1,n2
      do  i1=1,n1
         indx = indx + 1
         buff(indx,buff_id ) = u( i1,i2,  2)
      enddo
   enddo
endif

dir = +1

buff_id = 3 + dir
indx = 0

if( axis .eq.  1 )then
   do  i3=2,n3-1
      do  i2=2,n2-1
         indx = indx + 1
         buff(indx,buff_id ) = u( n1-1, i2,i3)
      enddo
   enddo
endif

if( axis .eq.  2 )then
   do  i3=2,n3-1
      do  i1=1,n1
         indx = indx + 1
         buff(indx,buff_id ) = u( i1,n2-1,
     i3)
      enddo
   enddo
endif

if( axis .eq.  3 )then
   do  i2=1,n2
      do  i1=1,n1
         indx = indx + 1
         buff(indx,buff_id ) = u( i1,i2,n3-
     1)
      enddo
   enddo
endif

dir = -1

buff_id = 2 + dir
buff_len = 0

if( axis .eq.  1 )then
   do  i3=2,n3-1
      do  i2=2,n2-1
         buff_len = buff_len + 1
         buff(buff_len,buff_id ) = u( 2,  i2,i3)
      enddo
   enddo
endif

if( axis .eq.  2 )then
   do  i3=2,n3-1
      do  i1=1,n1
         buff_len = buff_len + 1
         buff(buff_len,buff_id ) = u( i1,  2,i3)
      enddo
   enddo
endif

if( axis .eq.  3 )then
   do  i2=1,n2
      do  i1=1,n1
         buff_len = buff_len + 1
         buff(buff_len,buff_id ) = u( i1,i2,  2)
      enddo
   enddo
endif

dir = +1

buff_id = 3 + dir
indx = 0

if( axis .eq.  1 )then
   do  i3=2,n3-1
      do  i2=2,n2-1
         indx = indx + 1
         buff(indx,buff_id ) = u( n1-1, i2,i3)
      enddo
   enddo
endif

if( axis .eq.  2 )then
   do  i3=2,n3-1
      do  i1=1,n1
         indx = indx + 1
         buff(indx,buff_id ) = u( i1,n2-1,
     i3)
      enddo
   enddo
endif

if( axis .eq.  3 )then
   do  i2=1,n2
      do  i1=1,n1
         indx = indx + 1
         buff(indx,buff_id ) = u( i1,i2,n3-
     1)
      enddo
   enddo
endif

if(m1k.eq.3)then
   d1 = 2
else
   d1 = 1
endif

if(m2k.eq.3)then
   d2 = 2
else
   d2 = 1
endif

if(m3k.eq.3)then
   d3 = 2
else
   d3 = 1
endif

do  j3=2,m3j-1
   i3 = 2*j3-d3
   do  j2=2,m2j-1
      i2 = 2*j2-d2
      do j1=2,m1j
         i1 = 2*j1-d1
         x1(i1-1) = r(i1-1,i2-1,i3  ) + r(i1-1,i2+1,i3  )
     >            + r(i1-1,i2,  i3-1) + r(i1-1,i2,  i3+1)
         y1(i1-1) = r(i1-1,i2-1,i3-1) + r(i1-1,i2-1,i3+1)
     >            + r(i1-1,i2+1,i3-1) + r(i1-1,i2+1,i3+1)
      enddo
      do  j1=2,m1j-1
         i1 = 2*j1-d1
         y2 = r(i1,  i2-1,i3-1) + r(i1,  i2-1,i3+1)
     >        + r(i1,  i2+1,i3-1) + r(i1,  i2+1,i3+1)
         x2 = r(i1,  i2-1,i3  ) + r(i1,  i2+1,i3  )
     >        + r(i1,  i2,  i3-1) + r(i1,  i2,  i3+1)
         x(j1,j2,j3) =
     >        0.5D0 * r(i1,i2,i3)
     >      + 0.25D0 * (r(i1-1,i2,i3) + r(i1+1,i2,i3) + x2)
     >      + 0.125D0 * ( x1(i1-1) + x1(i1+1) + y2)
     >      + 0.0625D0 * ( y1(i1-1) + y1(i1+1) )
      enddo
   enddo
enddo
j = k-1
call comm3(x,m1j,m2j,m3j,j)
return
end
```

NAS MG *rprj3* stencil in Chapel

Chapel solution

- Exploits first class domains



```
def rprj3(S, R) {
  const Stencil = [-1..1, -1..1, -1..1],
        w: [0..3] real = (0.5, 0.25, 0.125, 0.0625),
        w3d = [(i,j,k) in Stencil] w((i!=0) + (j!=0) + (k!=0));

  forall ijk in S.domain do
    S(ijk) = + reduce [offset in Stencil]
                        (w3d(offset) * R(ijk + offset*R.stride));
}
```

135

Chapel Summary

Generality

- Extends the notion of data parallelism beyond dense arrays
- Supports task parallelism as well as data parallelism

Performance

- Allows programmers to encode locality into their programs

Correctness and programmability

- Raises the level of abstract
- Provides a Global View of data

136

X10

Overview

- Another PGAS language
- Supports both task parallelism and data parallelism
- Support for data parallelism focuses on dense arrays
- Strange memory semantics: Local vs. remote memory

137

Final Thoughts

Exciting times

- Parallelism is ubiquitous
- New willingness to consider new languages
- Chapel and X10 offer nice solutions

Work in parallel computing

- Your choice: You can be the bug or the windshield

138

Thank You!

139