The Audacity of Chapel:
Scalable Parallel Programming Done Right (director's cut, with outtakes)

Brad Chamberlain, Chapel Team, Cray Inc.

ACCU 2017, Bristol UK

April 27th, 2017

Safe Harbor Statement

This presentation may contain forward-looking statements that are based on our current expectations. Forward looking statements may include statements about our financial guidance and expected operating results, our opportunities and future potential, our product development and new product introduction plans, our ability to expand and penetrate our addressable markets and other statements that are not historical facts. These statements are only predictions and actual results may materially vary from those projected. Please refer to Cray's documents filed with the SEC from time to time concerning factors that could affect the Company and these forward-looking statements.

Copyright 2017 Cray Inc.

Fair Warning Statement

This keynote contains many slides and no guitar solos ☹

BUT… it does contain thematically relevant

UK rock lyrics trivia!!

Copyright 2017 Cray Inc.

I am the egg man

Introductions

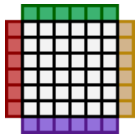
I am the Walrus
The Beatles
Magical Mystery Tour

Copyright 2017 Cray Inc.

("Pssst… Who is this turkey?")

Hi, I'm Brad Chamberlain

- graduate of the University of Washington
  - worked on a data-parallel array language, ZPL



- principal engineer at Cray Inc.
  - founding member and technical lead of the Chapel project

- more of a practical parallel computing guy than a PL expert…

Copyright 2017 Cray Inc.

The Context for My Work

HPC: High Performance Computing

- parallelism at large scales
  - lots of distributed processors and memories
- performance rules
  - and too often, is **all** that matters
- programmers are virtually living in the dark ages

CRAY: The Supercomputer Company

Copyright 2017 Cray Inc.

Recent Highlighted Cray Systems

NNSA National Nuclear Security Administration | Sandia National Laboratories | Los Alamos National Laboratory | Lawrence Livermore National Laboratory

Next-Gen Cray XC
19k compute nodes
40+ PF, 2+ PB
80+PB File System
5200 sq ft

Cray/Intel partnership
50k+ compute nodes
180PF, 7+ PB
150+ PB File System

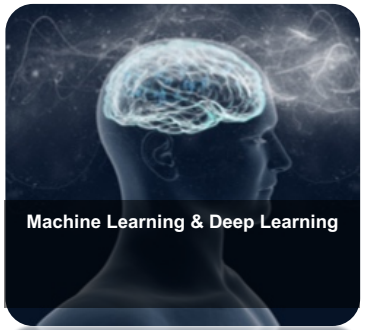
Argonne NATIONAL LABORATORY

CORAL COLLABORATION OAK RIDGE • ARGONNE • LIVERMORE

PGS Petroleum Geo-Services

Copyright 2017 Cray Inc.

Cray Market Segments



Copyright 2017 Cray Inc.

“I don’t really care about HPC programming...”

- **OK, but do you care about parallelism & concurrency?**
  - What about performance?
  - What about scaling up your data set sizes?
  - What about targeting next-generation processors?

Next-generation processors and computations are increasingly resembling traditional HPC.

Copyright 2017 Cray Inc.

If you didn't care what happened to me,
And I didn't care for you,
We would zig zag our way
Through the boredom and pain…

Motivation for Chapel

Pigs on the Wing (part one)
Pink Floyd
Animals

Copyright 2017 Cray Inc.

Motivation from Kathy Yelick (UC Berkeley, LBNL)

Why Consider New Languages at all?

- **Do we need a language? And a compiler?**
  - If higher level syntax is needed for productivity
    - We need a language
  - If static analysis is needed to help with correctness
    - We need a compiler (front-end)
  - If static optimizations are needed to get performance
    - We need a compiler (back-end)

Copyright 2017 Cray Inc.

What is Chapel?

Chapel: A productive parallel programming language

- portable
- open-source
- a collaborative effort

Goals:

- Support general parallel programming
  - "any parallel algorithm on any parallel hardware"
- Make parallel programming at scale far more productive

Copyright 2017 Cray Inc.

What does “Productivity” mean to you?

Recent Graduates:

“something similar to what I used in school: Python, Matlab, Java, …”

Seasoned HPC Programmers:

“that sugary stuff that I don’t need because I ~~was born to suffer~~ want full control to ensure performance”

Computational Scientists:

“something that lets me express my parallel computations without having to wrestle with architecture-specific details”

Chapel Team:

“something that lets computational scientists express what they want, without taking away the control that HPC programmers want, implemented in a language as attractive as recent graduates want.”

Copyright 2017 Cray Inc.

“The Audacity of Chapel”?

audacity (according to Google):

/ɔːˈdasɪti/

noun

1. a willingness to take bold risks.

   “I applaud the *audacity* of the Chapel team in attempting to create a new language given how hard it is for new languages to succeed.”

2. rude or disrespectful behaviour; impudence.

   “I can’t believe the Chapel team has the *audacity* to create a new language when we already have [C++ | Python | …]!”

Copyright 2017 Cray Inc.

This Talk's Thesis

Programming language designers have, to date, largely failed the large-scale parallel computing community.

Copyright 2017 Cray Inc.

This just feels like spinning plates
I'm living in cloud-cuckoo land

The Status Quo in HPC Programming

Like Spinning Plates
Radiohead
Amnesiac

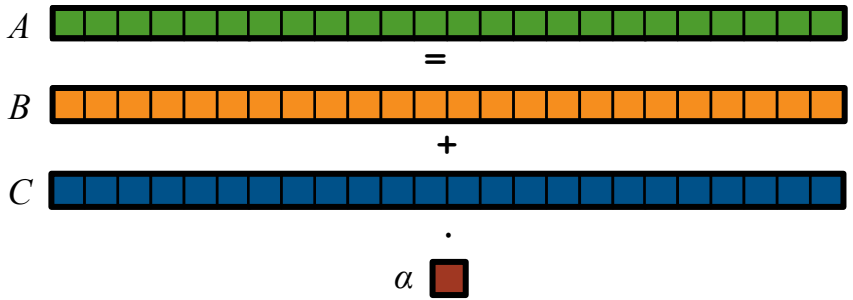
Copyright 2017 Cray Inc.

STREAM Triad: a trivial parallel computation

Given: m-element vectors A, B, C

Compute: $\forall i \in 1..m, A_i = B_i + \alpha \cdot C_i$

In pictures:



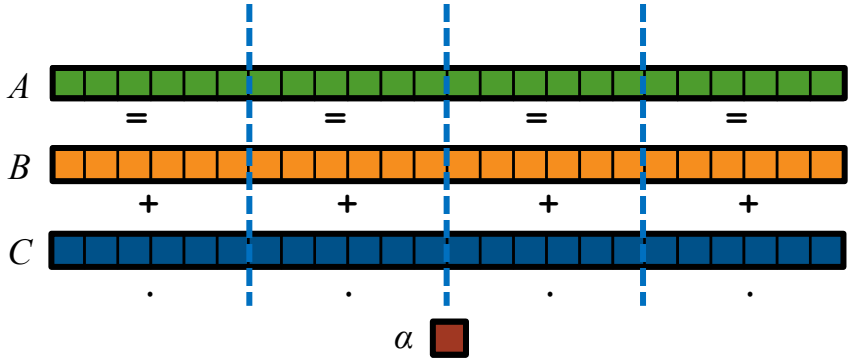
Copyright 2017 Cray Inc.

STREAM Triad: a trivial parallel computation

Given: m-element vectors A, B, C

Compute: $\forall i \in 1..m, A_i = B_i + \alpha \cdot C_i$

In pictures, in parallel (shared memory / multicore):



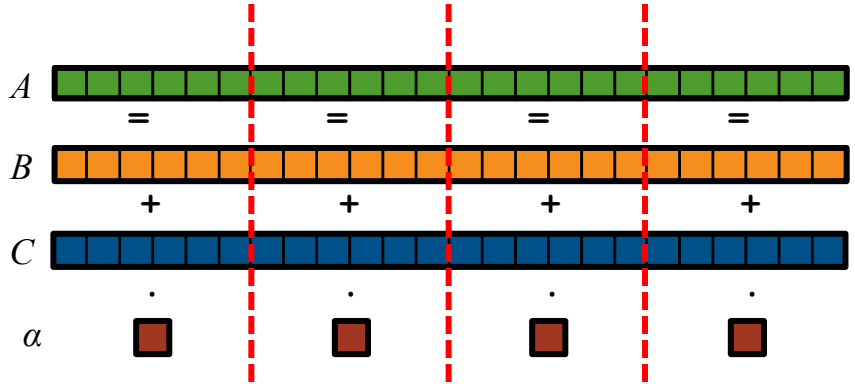
Copyright 2017 Cray Inc.

STREAM Triad: a trivial parallel computation

Given: m-element vectors A, B, C

Compute: $\forall i \in 1..m, A_i = B_i + \alpha \cdot C_i$

In pictures, in parallel (distributed memory):



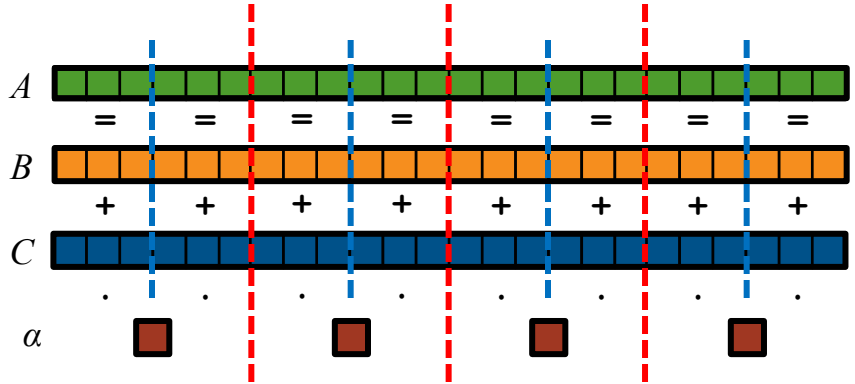
Copyright 2017 Cray Inc.

STREAM Triad: a trivial parallel computation

Given: m-element vectors A, B, C

Compute: $\forall i \in 1..m, A_i = B_i + \alpha \cdot C_i$

In pictures, in parallel (distributed memory multicore):



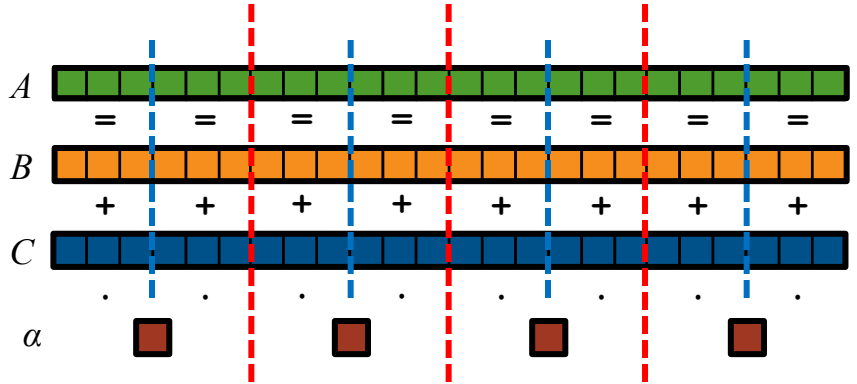
Copyright 2017 Cray Inc.

Scalable Parallel Programming Concerns

Q: What should scalable parallel programmers focus on?

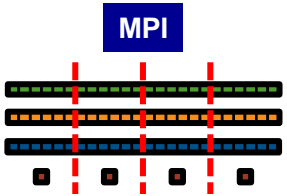
A: *Parallelism:* What should execute simultaneously?

Locality: Where should those tasks execute?



Copyright 2017 Cray Inc.

STREAM Triad: MPI




```
#include <hpcc.h>

static int VectorSize;
static double *a, *b, *c;

int HPCC_StarStream(HPCC_Params *params) {
  int myRank, commSize;
  int rv, errCount;
  MPI_Comm comm = MPI_COMM_WORLD;

  MPI_Comm_size( comm, &commSize );
  MPI_Comm_rank( comm, &myRank );

  rv = HPCC_Stream( params, 0 == myRank);
  MPI_Reduce( &rv, &errCount, 1, MPI_INT, MPI_SUM,
    0, comm );

  return errCount;
}

int HPCC_Stream(HPCC_Params *params, int doIO) {
  register int j;
  double  scalar;

  VectorSize = HPCC_LocalVectorSize( params, 3,
    sizeof(double), 0 );

  a = HPCC_XMALLOC( double, VectorSize );
  b = HPCC_XMALLOC( double, VectorSize );
  c = HPCC_XMALLOC( double, VectorSize );
```

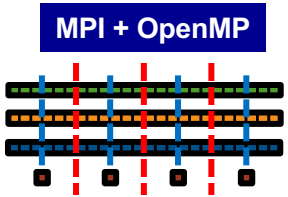
```
if (!a || !b || !c) {
  if (c) HPCC_free(c);
  if (b) HPCC_free(b);
  if (a) HPCC_free(a);
  if (doIO) {
    fprintf( outFile, "Failed to
      allocate memory (%d).\n",
      VectorSize );
    fclose( outFile );
  }
  return 1;
}
```

```
for (j=0; j<VectorSize; j++) {
  b[j] = 2.0;
  c[j] = 1.0;
}
scalar = 3.0;
```

```
for (j=0; j<VectorSize; j++)
  a[j] = b[j]+scalar*c[j];

HPCC_free(c);
HPCC_free(b);
HPCC_free(a);

return 0;  }
```


Copyright 2017 Cray Inc.

STREAM Triad: MPI+OpenMP

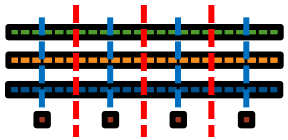



```
#include <hpcc.h>
#ifdef _OPENMP
#include <omp.h>
#endif

static int VectorSize;
static double *a, *b, *c;

int HPCC_StarStream(HPCC_Params *params) {
  int myRank, commSize;
  int rv, errCount;
  MPI_Comm comm = MPI_COMM_WORLD;

  MPI_Comm_size( comm, &commSize );
  MPI_Comm_rank( comm, &myRank );

  rv = HPCC_Stream( params, 0 == myRank);
  MPI_Reduce( &rv, &errCount, 1, MPI_INT, MPI_SUM,
    0, comm );

  return errCount;
}

int HPCC_Stream(HPCC_Params *params, int doIO) {
  register int j;
  double  scalar;

  VectorSize = HPCC_LocalVectorSize( params, 3,
    sizeof(double), 0 );

  a = HPCC_XMALLOC( double, VectorSize );
  b = HPCC_XMALLOC( double, VectorSize );
  c = HPCC_XMALLOC( double, VectorSize );

  if (!a || !b || !c) {
    if (c) HPCC_free(c);
    if (b) HPCC_free(b);
    if (a) HPCC_free(a);
    if (doIO) {
      fprintf( outFile, "Failed to
        allocate memory (%d).\n",
        VectorSize );
      fclose( outFile );
    }
    return 1;
  }

#ifdef _OPENMP
#pragma omp parallel for
#endif
  for (j=0; j<VectorSize; j++) {
    b[j] = 2.0;
    c[j] = 1.0;
  }
  scalar = 3.0;

#ifdef _OPENMP
#pragma omp parallel for
#endif
  for (j=0; j<VectorSize; j++)
    a[j] = b[j]+scalar*c[j];

  HPCC_free(c);
  HPCC_free(b);
  HPCC_free(a);

  return 0;  }
```


Copyright 2017 Cray Inc.

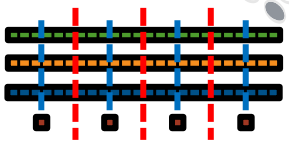
STREAM Triad: MPI+OpenMP

MPI + OpenMP



```
#include <hpcc.h>
#ifdef _OPENMP
#include <omp.h>
#endif

static int VectorSize;
static double *a, *b, *c;

int HPCC_StarStream(HPCC_Params *params) {
  int myRank, commSize;
  int rv, errCount;
  MPI_Comm comm = MPI_COMM_WORLD;

  MPI_Comm_size( comm, &commSize );
  MPI_Comm_rank( comm, &myRank );

  rv = HPCC_Stream( params, 0 == myRank);
  MPI_Reduce( &rv, &errCount, 1, MPI_INT, MPI_SUM,
    0, comm );

  return errCount;
}

int HPCC_Stream(HPCC_Params *params, int doIO) {
  register int j;
  double  scalar;

  VectorSize = HPCC_LocalVectorSize( params, 3,
    sizeof(double), 0 );

  a = HPCC_XMALLOC( double, VectorSize );
  b = HPCC_XMALLOC( double, VectorSize );
  c = HPCC_XMALLOC( double, VectorSize );

  if (!a || !b || !c) {
    if (c) HPCC_free(c);
    if (b) HPCC_free(b);
    if (a) HPCC_free(a);
    if (doIO) {
      fprintf( outFile, "Failed to
        allocate memory (%d).\n",
        VectorSize );
      fclose( outFile );
    }
    return 1;
  }

#ifdef _OPENMP
#pragma omp parallel for
#endif
  for (j=0; j<VectorSize; j++) {
    b[j] = 2.0;
    c[j] = 1.0;
  }
  scalar = 3.0;

#ifdef _OPENMP
#pragma omp parallel for
#endif
  for (j=0; j<VectorSize; j++)
    a[j] = b[j]+scalar*c[j];

  HPCC_free(c);
  HPCC_free(b);
  HPCC_free(a);

  return 0;
}
```

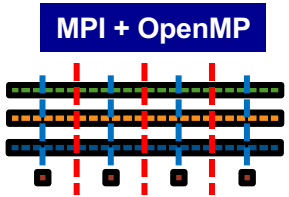
CUDA



```
#define N 2000000

int main() {
  float *d_a, *d_b, *d_c;
  float scalar;

  cudaMalloc((void**)&d_a, sizeof(float)*N);
  cudaMalloc((void**)&d_b, sizeof(float)*N);
  cudaMalloc((void**)&d_c, sizeof(float)*N);

  dim3 dimBlock(128);
  dim3 dimGrid(N/dimBlock.x );
  if( N % dimBlock.x != 0 ) dimGrid

  set_array<<<dimGrid,dimBlock>>>(d_b, .5f, N);
  set_array<<<dimGrid,dimBlock>>>(d_c, .5f, N);

  scalar=3.0f;
  STREAM_Triad<<<dimGrid,dimBlock>>>(d_b, d_c, d_a, scalar,  N);
  cudaThreadSynchronize();

  cudaFree(d_a);
  cudaFree(d_b);
  cudaFree(d_c);

__global__ void set_array(float *a,  float value, int len) {
  int idx = threadIdx.x + blockIdx.x * blockDim.x;
  if (idx < len) a[idx] = value;
}

__global__ void STREAM_Triad( float *a, float *b, float *c,
                              float scalar, int len) {
  int idx = threadIdx.x + blockIdx.x * blockDim.x;
  if (idx < len) c[idx] = a[idx]+scalar*b[idx];  }
```


Copyright 2017 Cray Inc.

STREAM Triad: MPI+OpenMP

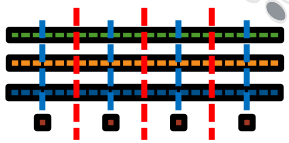



```
#include <hpcc.h>
#ifdef _OPENMP
#include <omp.h>
#endif

static int VectorSize;
static double *a, *b, *c;

int HPCC_StarStream(HPCC_Params *params) {
  int myRank, commSize;
```

```
  MPI_Reduce( &rv, &errCount, 1, MPI_INT, MPI_SUM,
    0, comm );

  return errCount;
}

int HPCC_Stream(HPCC_Params *params, int doIO) {
  register int j;
  double  scalar;

  VectorSize = HPCC_LocalVectorSize( params, 3,
    sizeof(double), 0 );

  a = HPCC_XMALLOC( double, VectorSize );
  b = HPCC_XMALLOC( double, VectorSize );
  c = HPCC_XMALLOC( double, VectorSize );
```

```
if (!a || !b || !c) {
  if (c) HPCC_free(c);
  if (b) HPCC_free(b);
  if (a) HPCC_free(a);
  if (doIO) {
    fprintf( outFile, "Failed to
      allocate memory (%d).\n",
      VectorSize );
    fclose( outFile );
  }
```

```
  for (j=0; j<VectorSize; j++) {
    b[j] = 2.0;
    c[j] = 1.0;
  }
  scalar = 3.0;

#ifdef _OPENMP
#pragma omp parallel for
#endif
  for (j=0; j<VectorSize; j++)
    a[j] = b[j]+scalar*c[j];

  HPCC_free(c);
  HPCC_free(b);
  HPCC_free(a);

  return 0;  }
```




```
#define N 2000000

int main() {
  float *d_a, *d_b, *d_c;
  float scalar;

  cudaMalloc((void**)&d_a, sizeof(float)*N);
  cudaMalloc((void**)&d_b, sizeof(float)*N);
  cudaMalloc((void**)&d_c, sizeof(float)*N);

  dim3 dimBlock(128);
```

```
  STREAM_Triad<<<dimGrid,dimBlock>>>(d_b, d_c, d_a, scalar,  N);
  cudaThreadSynchronize();

  cudaFree(d_a);
  cudaFree(d_b);
  cudaFree(d_c);


__global__ void set_array(float *a,  float value, int len) {
  int idx = threadIdx.x + blockIdx.x * blockDim.x;
  if (idx < len) a[idx] = value;
}

__global__ void STREAM_Triad( float *a, float *b, float *c,
                              float scalar, int len) {
  int idx = threadIdx.x + blockIdx.x * blockDim.x;
  if (idx < len) c[idx] = a[idx]+scalar*b[idx];  }
```

HPC suffers from too many distinct notations for expressing parallelism and locality. This tends to be a result of bottom-up language design.

Copyright 2017 Cray Inc.

Why so many programming models?

HPC tends to approach programming models bottom-up:

Given a system and its core capabilities…

…provide features that permit users to access the available performance.

- portability? generality? programmability? These are second- or third-order concerns, if that.

Type of HW Parallelism	Programming Model	Unit of Parallelism
Inter-node	MPI	executable
Intra-node/multicore	OpenMP / pthreads	iteration/task
Instruction-level vectors/threads	pragmas	iteration
GPU/accelerator	CUDA / Open[MP|CL|ACC]	SIMD function/task

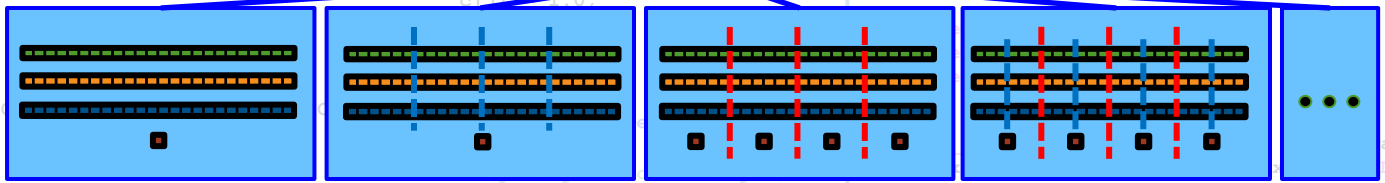
benefits: lots of control; decent generality; easy to implement
downsides: lots of user-managed detail; brittle to changes

Copyright 2017 Cray Inc.

STREAM Triad: Chapel

```
use …;

config const m = 1000,
             alpha = 3.0;

const ProblemSpace = {1..m} dmapped …;

var A, B, C: [ProblemSpace] real;

B = 2.0;
C = 1.0;

A = B + alpha * C;
```

The special sauce:
How should this index set—and any arrays and computations over it—be mapped to the system?



Philosophy: Good, *top-down* language design can tease system-specific implementation details away from an algorithm, permitting the compiler, runtime, applied scientist, and HPC expert to each focus on their strengths.

Copyright 2017 Cray Inc.

This Talk's Takeaways

If you design a parallel programming language…

…don't tie yourself to low-level, architecture-specific mechanisms

- yet don't make them inaccessible either…
  - permit interoperating with such mechanisms
  - or support them as the "assembly" to your higher-level features

Copyright 2017 Cray Inc.

It's so easy to laugh, it's so easy to hate
It takes guts to be gentle and kind

SPMD Programming Models like MPI

I Know It's Over
The Smiths
The Queen is Dead

Copyright 2017 Cray Inc.

HPC's Status Quo: SPMD Programming

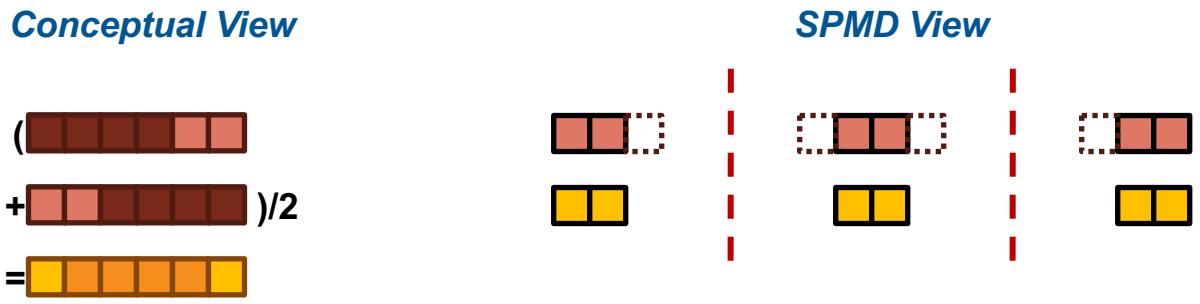
SPMD: Single Program, Multiple Data

- concept: write one program, run multiple copies of it in parallel
- a "bottom-up" programming model design
  - "HPC systems can run lots of programs, so let's get parallelism that way"
- often clumsy in practice

Copyright 2017 Cray Inc.

SPMD by Example (in pictures)

“Apply a 3-Point Stencil to a vector”

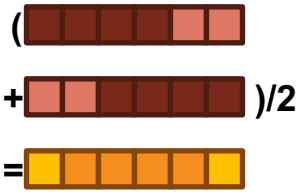


Copyright 2017 Cray Inc.

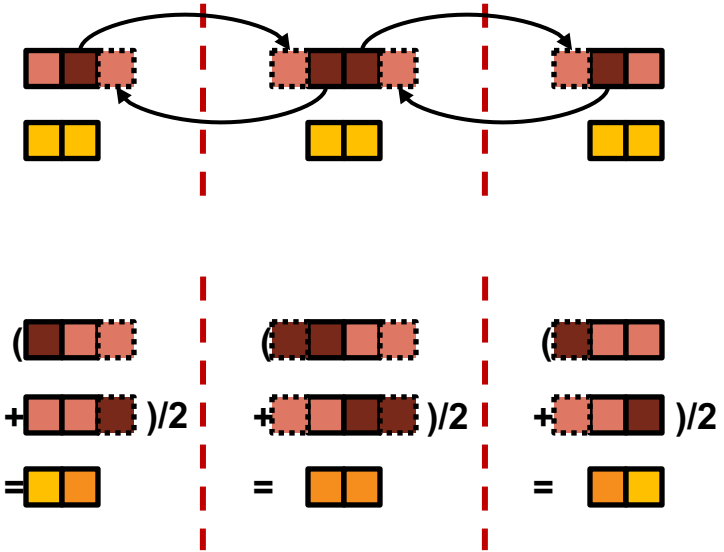
SPMD by Example (in pictures)

“Apply a 3-Point Stencil to a vector”

Conceptual View



SPMD View

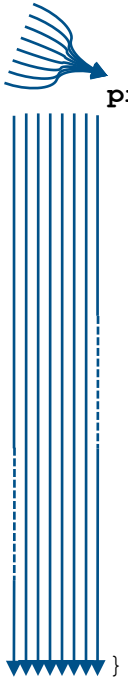


Copyright 2017 Cray Inc.

SPMD by Example (in code)

“Apply a 3-Point Stencil to a vector”

SPMD pseudo-code



```
proc main() {
  var n = 1000;
  var p = numProcs(),
      me = myProc(),
      myN = n/p,
  var A, B: [0..myN+1] real;

  if (me < p-1) {
    send(me+1, A[myN]);
    recv(me+1, A[myN+1]);
  }
  if (me > 0) {
    send(me-1, A[1]);
    recv(me-1, A[0]);
  }
  forall i in 1..myN do
    B[i] = (A[i-1] + A[i+1])/2;
}
```


Copyright 2017 Cray Inc.

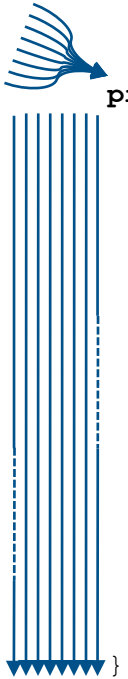
SPMD by Example (in code)

"Apply a 3-Point Stencil to a vector"

Global-view code (Chapel)

```
proc main() {
  const n = 1000,
        D = {1..n} dmapped Block(…);
  var A, B: [1..n] real;

  forall i in 2..n-1 do
    B[i] = (A[i-1] + A[i+1])/2;
}
```

SPMD pseudo-code



```
proc main() {
  var n = 1000;
  var p = numProcs(),
      me = myProc(),
      myN = n/p,
  var A, B: [0..myN+1] real;

  if (me < p-1) {
    send(me+1, A[myN]);
    recv(me+1, A[myN+1]);
  }
  if (me > 0) {
    send(me-1, A[1]);
    recv(me-1, A[0]);
  }
  forall i in 1..myN do
    B[i] = (A[i-1] + A[i+1])/2;
}
```


Copyright 2017 Cray Inc.

SPMD by Example (in code)

"Apply a 3-Point Stencil to a vector"

Global-view code (Chapel)

```
proc main() {
  const n = 1000,
        D = {1..n} dmapped Block(…);
  var A, B: [1..n] real;

  forall i in 2..n-1 do
}
```

Bug: Refers to uninitialized values at ends of A

SPMD pseudo-code

```
proc main() {
  var n = 1000;
  var p = numProcs(),
      me = myProc(),
      myN = n/p,
  var A, B: [0..myN+1] real;

  if (me < p-1) {
    send(me+1, A[myN]);
    recv(me+1, A[myN+1]);
  }
  if (me > 0) {
    send(me-1, A[1]);
    recv(me-1, A[0]);
  }
  forall i in 1..myN do
    B[i] = (A[i-1] + A[i+1])/2;
}
```


Copyright 2017 Cray Inc.

SPMD by Example (in code)

"Apply a 3-Point Stencil to a vector"

Global-view code (Chapel)

```
proc main() {
  const n = 1000,
        D = {1..n} dmapped Block(…);
  var A, B: [1..n] real;

  forall
    B[i]
}
```

Communication becomes geometrically more complex for higher-dimensional arrays

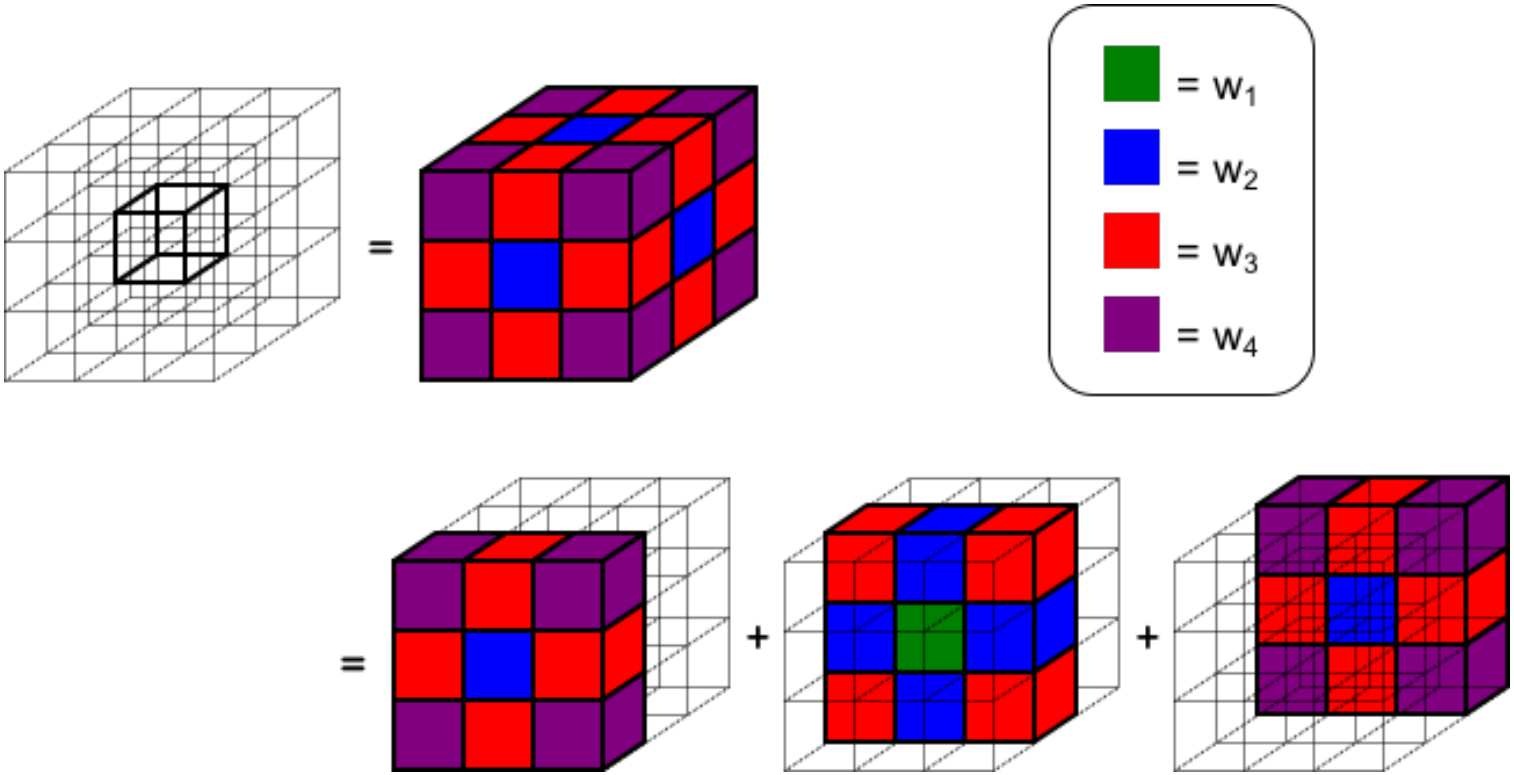
SPMD pseudo-code

```
proc main() {
  var n = 1000;
  var p = numProcs(),
      me = myProc(),
      myN = n/p,
      myLo = 1,
      myHi = myN;
  var A, B: [0..myN+1] real;

  if (me < p-1) {
    send(me+1, A[myN]);
    recv(me+1, A[myN+1]);
  } else
    myHi = myN-1;
  if (me > 0) {
    send(me-1, A[1]);
    recv(me-1, A[0]);
  } else
    myLo = 2;
  forall i in myLo..myHi do
    B[i] = (A[i-1] + A[i+1])/2;
}
```

Assumes p divides n

Copyright 2017 Cray Inc.

27-point stencils (*rprj3* from NAS MG)



Copyright 2017 Cray Inc.

rprj3 in Fortran + MPI

```
      subroutine rprj3(r,m1k,m2k,m3k,s,m1j,m2j,m3j,k)
      implicit none
      include 'cafnpb.h'
      include 'globals.h'

      integer m1k, m2k, m3k, m1j, m2j, m3j,k

      double precision r(m1k,m2k,m3k), s(m1j,m2j,m3j)
      integer j3, j2, j1, i3, i2, i1, d1, d2, d3, j
      double precision x1(m), y1(m), x2,y2

      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)
            s(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(s,m1j,m2j,m3j,j)
        return
        end
```


Copyright 2017 Cray Inc.

rprj3 in Fortran + MPI

```
subroutine rprj3(r,m1k,m2k,m3k,s,m1j,m2j,m3j,k)
implicit none
include 'cafnpb.h'
include 'globals.h'

integer m1k, m2k, m3k, m1j, m2j, m3j,k

double precision r(m1k,m2k,m3k), s(m1j,m2j,m3j)
integer j3, j2, j1, i3, i2, i1, d1, d2, d3, j
double precision x1(m), y1(m), x2,y2

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)
      s(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(s,m1j,m2j,m3j,j)
  return
  end
```


Copyright 2017 Cray Inc.

comm3: the communication for *rprj3*

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

implicit none

include 'cafnpb.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( nproces .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 commlp( 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 'cafnpb.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(axi
     s,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

   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.  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  i2=1,n2
            do  i1=1,n1
               indx = indx + 1
               u(i1,n2,i3) =
     buff(indx, buff_id )
            enddo
         enddo

      endif
   endif

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

         do  i3=2,n3-1
            do  i1=1,n1
               indx = indx + 1
               u(i1,i2,3) =
     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,n3) =
     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  i2=1,n2
            do  i1=1,n1
               indx = indx + 1
               u(i1,i2,n3) =
     buff(indx, buff_id )
            enddo
         enddo

      endif
   endif

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

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

      endif
   endif

   if( axis .eq.  3 )then
      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

      else 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

      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(i1,1,i3) =
     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

   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.  3 )then
      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

      else if( dir .eq. +1 ) then

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

      endif
   endif

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

         do  i1=1,n1
            do  i2=1,n2
               indx = indx + 1
               u(i1,1,i3) =
     buff(indx, buff_id )
            enddo
         enddo

      else if( dir .eq. +1 ) then

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

      endif
   endif

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

         do  i1=1,n1
            do  i2=1,n2
               indx = indx + 1
               u(i1,i2,i3) =
     buff(indx, buff_id )
            enddo
         enddo

      else if( dir .eq. +1 ) then

         do  i1=1,n1
            do  i2=1,n2
               indx = indx + 1
               u(i1,i2,i3) =
     buff(indx, buff_id )
            enddo
         enddo

      endif
   endif
```


Copyright 2017 Cray Inc.

Being Gutsy, Gentle, and Kind to MPI

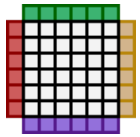
- **It's enabled the vast majority of HPC results for the past ~20 years**
- **It's very analogous to assembly programming**
  - explicitly move data from memory to registers

    vs.

    explicitly move data between compute nodes' memories
- **Just like assembly, it's an important technology**
  - for programming at low levels
  - for enabling higher-level technologies
- **Yet, as with assembly, we should develop higher-level alternatives**

Copyright 2017 Cray Inc.

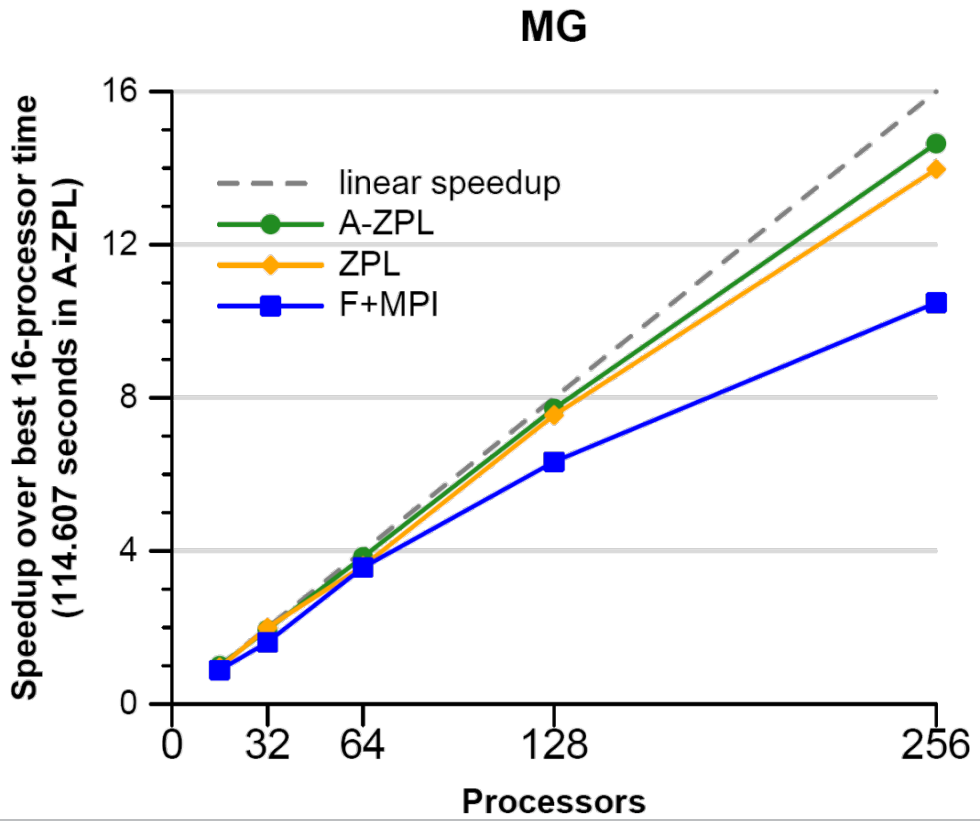
rprj3 in ZPL



```
procedure rprj3(var S,R: [,,] double;
                d: array [] of direction);
begin
  S := 0.5000 * R +
       0.2500 * (R@^d[ 1, 0, 0] + R@^d[ 0, 1, 0] + R@^d[ 0, 0, 1] +
                 R@^d[-1, 0, 0] + R@^d[ 0,-1, 0] + R@^d[ 0, 0,-1] +
       0.1250 * (R@^d[ 1, 1, 0] + R@^d[ 1, 0, 1] + R@^d[ 0, 1, 1] +
                 R@^d[ 1,-1, 0] + R@^d[ 1, 0,-1] + R@^d[ 0, 1,-1] +
                 R@^d[-1, 1, 0] + R@^d[-1, 0, 1] + R@^d[ 0,-1, 1] +
                 R@^d[-1,-1, 0] + R@^d[-1, 0,-1] + R@^d[ 0,-1,-1])+
       0.0625 * (R@^d[ 1, 1, 1] + R@^d[ 1, 1,-1] +
                 R@^d[ 1,-1, 1] + R@^d[ 1,-1,-1] +
                 R@^d[-1, 1, 1] + R@^d[-1, 1,-1] +
                 R@^d[-1,-1, 1] + R@^d[-1,-1,-1]);
end;
```

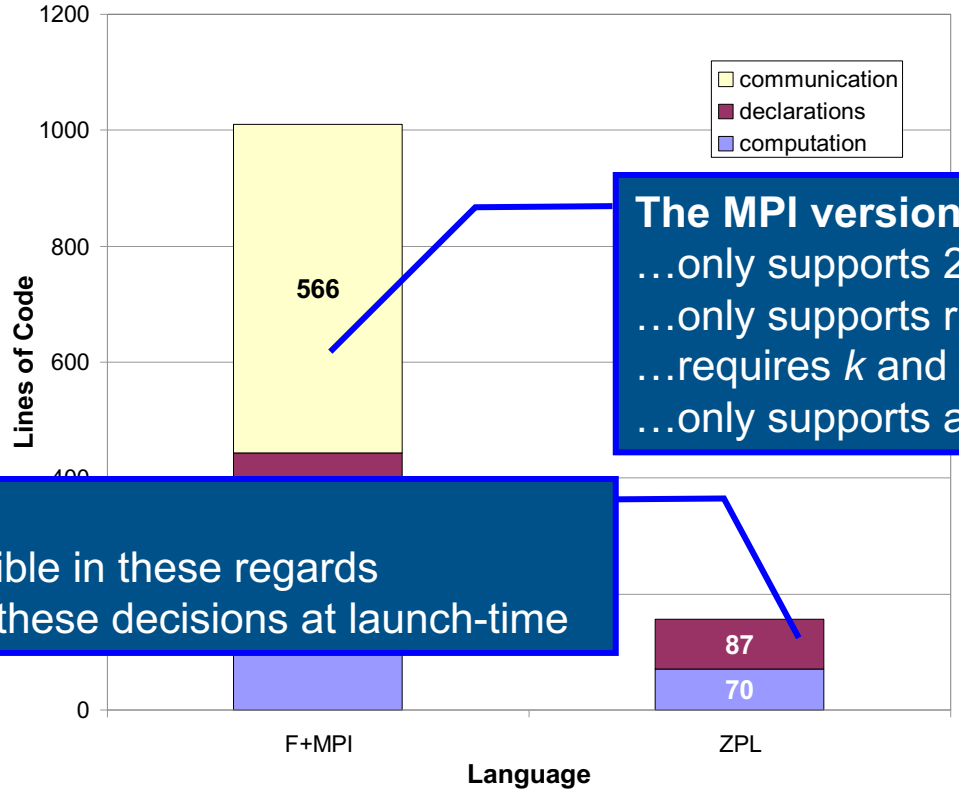
Copyright 2017 Cray Inc.

NAS MG Speedup: Cray T3E



Copyright 2017 Cray Inc.

Code Size Comparison



The MPI version…
…only supports 2^k problem sizes
…only supports running on 2^p nodes
…requires *k* and *p* to be specified statically
…only supports a single 3D distribution

The ZPL version…
…is completely flexible in these regards
…supports making these decisions at launch-time

Copyright 2017 Cray Inc.

This Talk's Takeaways

If you design a parallel programming language…

…don't base your model for parallelism and locality on SPMD

- instead, support a global view of parallelism and locality (like ZPL)

Copyright 2017 Cray Inc.

Epilogue: So why was ZPL not adopted?

- **Too restricted in terms of generality:**
  - only a single level of array-based data parallelism
  - only a single parallel array type: block-distributed
  - lack of “manual overrides” to drop down closer to the system
  - choices that were dated (no OOP, modula-based syntax, …)
- **Great academic project, not a great practical language**
  - however, ZPL’s experiences informed Chapel’s design greatly

Copyright 2017 Cray Inc.


And now for something
completely different.

[Instrumental]

Chapel Characteristics

The Liberty Bell March
John Philip Sousa
(unreleased?)

Copyright 2017 Cray Inc.

Chapel's Goal

To create a language that is…

…as productive as Python

…as fast as Fortran

…as portable as C

…as scalable as MPI

…as fun as [*insert your favorite language here*]

Copyright 2017 Cray Inc.

The Challenge

Q: So why don't we already have such a language already?

A: ~~Technical challenges?~~

- while they exist, we don't think this is the main issue…

A: Due to a lack of…

…long-term efforts

…resources

…community will

…co-design between developers and users

…patience

Chapel is our attempt to reverse this trend

Copyright 2017 Cray Inc.

Chapel is Portable

- **Chapel's design and implementation are hardware-independent**
- **The current release requires:**
  - a C/C++ compiler
  - a *NIX environment: Linux, Mac OS X, Windows 10 w/ bash, Cygwin, …
  - POSIX threads
  - UDP, MPI, or RDMA (if distributed memory execution is desired)
- **Chapel runs on…**

  …laptops and workstations
  …commodity clusters
  …the cloud
  …HPC systems from Cray and other vendors
  …modern processors like Intel Xeon Phi, GPUs*, etc.

* = academic work only; not yet supported in the official release

Copyright 2017 Cray Inc.

Chapel is Open-Source

- **Chapel's development is hosted at GitHub**
  - https://github.com/chapel-lang
- **Chapel is licensed as Apache v2.0 software**
- **Instructions for download + install are online**
  - see http://chapel.cray.com/download.html to get started

Copyright 2017 Cray Inc.

The Chapel Team at Cray (May 2016)

14 full-time employees + 2 summer interns + occasional visiting academics
(one of each started after photo taken)

Copyright 2017 Cray Inc.

Chapel Community R&D Efforts

(and several others…)

http://chapel.cray.com/collaborations.html

Copyright 2017 Cray Inc.

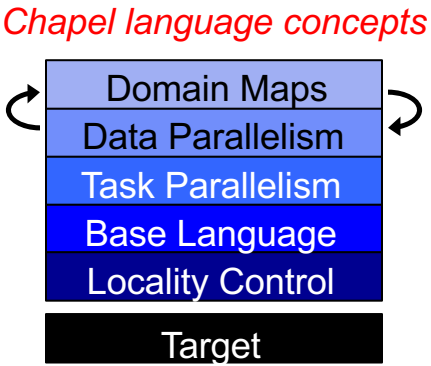
You say you got a real solution
Well, you know
We'd all love to see the plan

Chapel in a Nutshell

Revolution
The Beatles
The Beatles

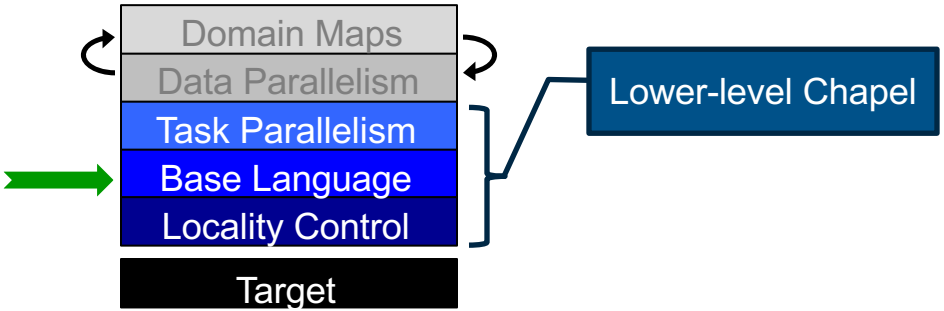
Copyright 2017 Cray Inc.

Chapel language feature areas



Copyright 2017 Cray Inc.

Base Language



Copyright 2017 Cray Inc.

Base Language Features, by example

```
iter fib(n) {
  var current = 0,
      next = 1;

  for i in 1..n {
    yield current;
    current += next;
    current <=> next;
  }
}
```

```
config const n = 10;

for f in fib(n) do
  writeln(f);
```

```
0
1
1
2
3
5
8
...
```


Copyright 2017 Cray Inc.

Base Language Features, by example

Modern iterators

```
iter fib(n) {
  var current = 0,
      next = 1;

  for i in 1..n {
    yield current;
    current += next;
    current <=> next;
  }
}
```

```
config const n = 10;

for f in fib(n) do
  writeln(f);
```

```
0
1
1
2
3
5
8
...
```


Copyright 2017 Cray Inc.

Base Language Features, by example

Configuration declarations
(to avoid command-line argument parsing)
`./a.out --n=1000000`

```
iter fib(n) {
  var current = 0,
      next = 1;

  for i in 1..n {
    yield current;
    current += next;
    current <=> next;
  }
}
```

```
config const n = 10;

for f in fib(n) do
  writeln(f);
```

```
0
1
1
2
3
5
8
...
```


Copyright 2017 Cray Inc.

Base Language Features, by example

Static type inference for:
- arguments
- return types
- variables

```
iter fib(n) {
  var current = 0,
      next = 1;

  for i in 1..n {
    yield current;
    current += next;
    current <=> next;
  }
}
```

```
config const n = 10;

for f in fib(n) do
  writeln(f);
```

```
0
1
1
2
3
5
8
...
```


Copyright 2017 Cray Inc.

Base Language Features, by example

```
iter fib(n) {
  var current = 0,
      next = 1;

  for i in 1..n {
    yield current;
    current += next;
    current <=> next;
  }
}
```

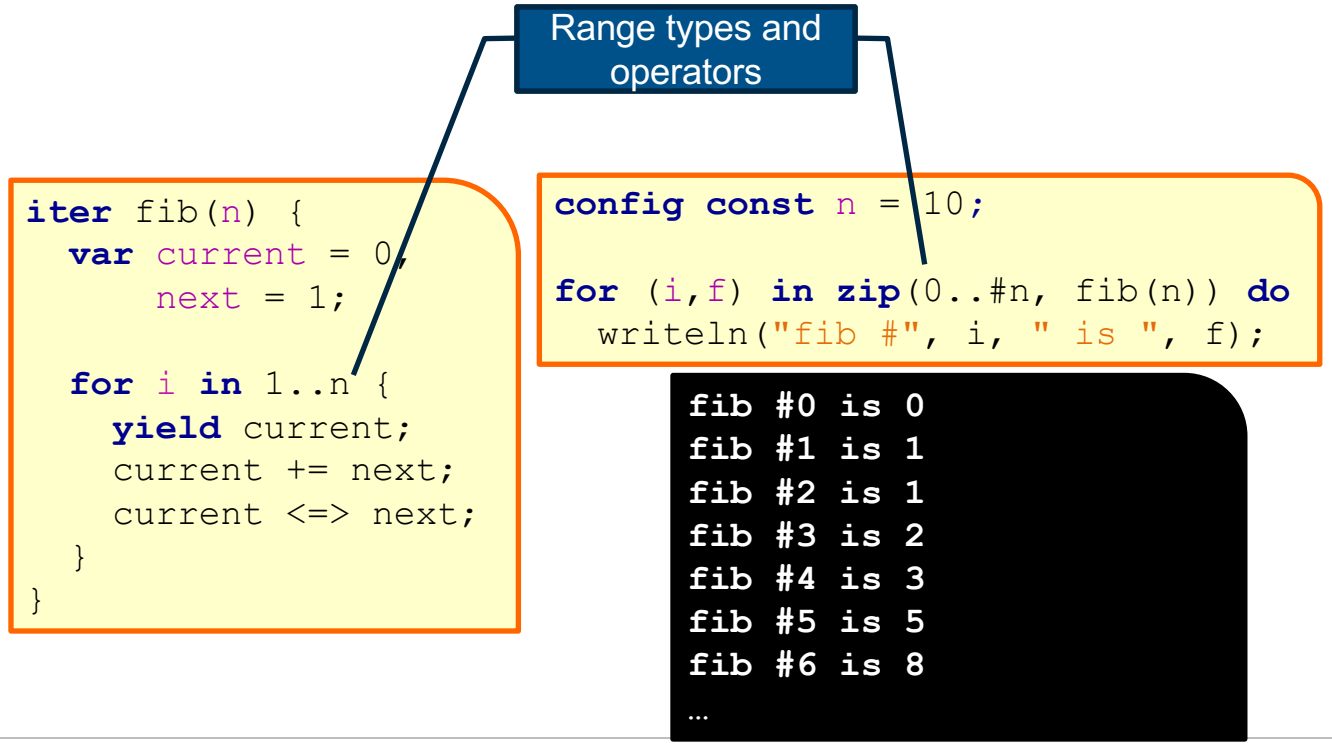
Zippered iteration

```
config const n = 10;

for (i,f) in zip(0..#n, fib(n)) do
  writeln("fib #", i, " is ", f);
```

```
fib #0 is 0
fib #1 is 1
fib #2 is 1
fib #3 is 2
fib #4 is 3
fib #5 is 5
fib #6 is 8
…
```

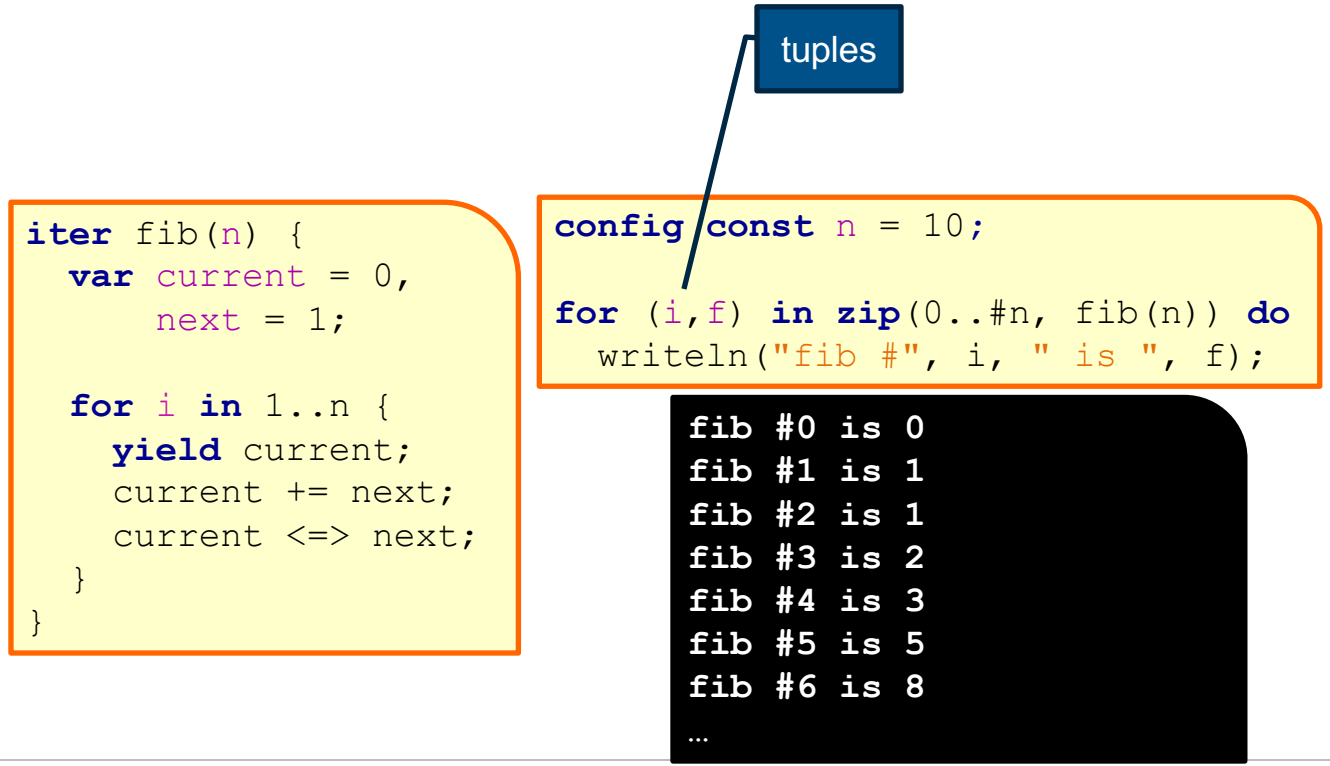

Copyright 2017 Cray Inc.

Base Language Features, by example



Copyright 2017 Cray Inc.

Base Language Features, by example



Copyright 2017 Cray Inc.

Base Language Features, by example

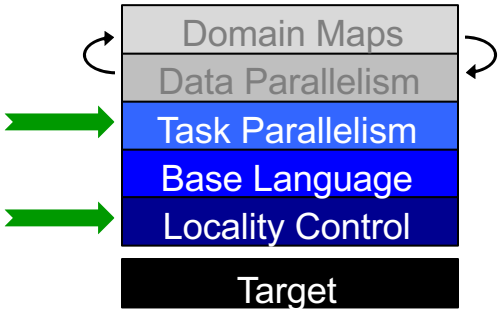
```
iter fib(n) {
  var current = 0,
      next = 1;

  for i in 1..n {
    yield current;
    current += next;
    current <=> next;
  }
}
```

```
config const n = 10;

for (i,f) in zip(0..#n, fib(n)) do
  writeln("fib #", i, " is ", f);
```

```
fib #0 is 0
fib #1 is 1
fib #2 is 1
fib #3 is 2
fib #4 is 3
fib #5 is 5
fib #6 is 8
…
```


Copyright 2017 Cray Inc.

Task Parallelism and Locality Control



Copyright 2017 Cray Inc.

Task Parallelism and Locality, by example

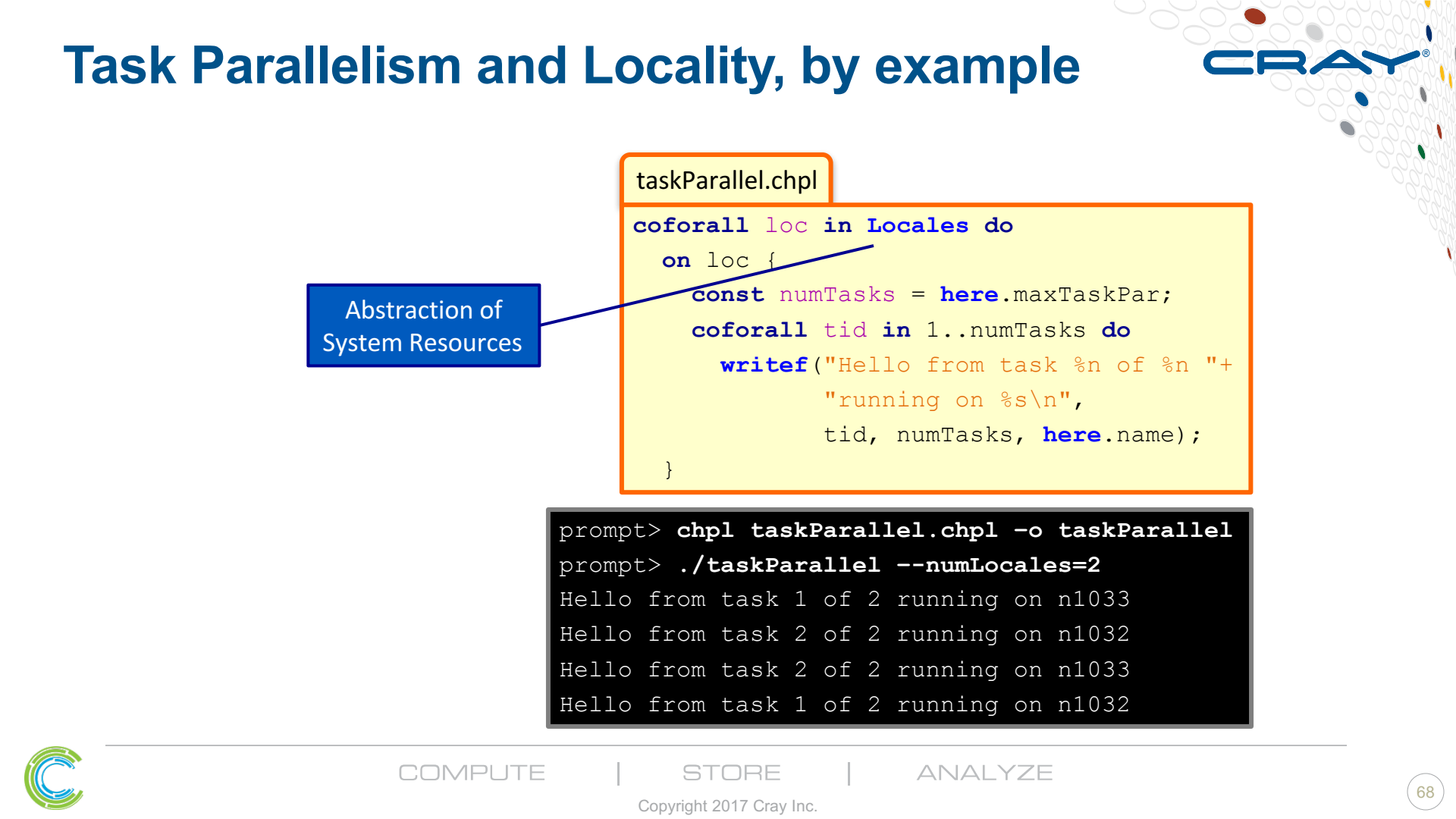
taskParallel.chpl

```
coforall loc in Locales do
  on loc {
    const numTasks = here.maxTaskPar;
    coforall tid in 1..numTasks do
      writef("Hello from task %n of %n "+
             "running on %s\n",
             tid, numTasks, here.name);
  }
```

```
prompt> chpl taskParallel.chpl -o taskParallel
prompt> ./taskParallel --numLocales=2
Hello from task 1 of 2 running on n1033
Hello from task 2 of 2 running on n1032
Hello from task 2 of 2 running on n1033
Hello from task 1 of 2 running on n1032
```


Copyright 2017 Cray Inc.

Task Parallelism and Locality, by example



taskParallel.chpl

```
coforall loc in Locales do
  on loc {
    const numTasks = here.maxTaskPar;
    coforall tid in 1..numTasks do
      writef("Hello from task %n of %n "+
             "running on %s\n",
             tid, numTasks, here.name);
  }
```

```
prompt> chpl taskParallel.chpl -o taskParallel
prompt> ./taskParallel --numLocales=2
Hello from task 1 of 2 running on n1033
Hello from task 2 of 2 running on n1032
Hello from task 2 of 2 running on n1033
Hello from task 1 of 2 running on n1032
```


Copyright 2017 Cray Inc.

Task Parallelism and Locality, by example

High-Level Task Parallelism

taskParallel.chpl

```
coforall loc in Locales do
  on loc {
    const numTasks = here.maxTaskPar;
    coforall tid in 1..numTasks do
      writef("Hello from task %n of %n "+
             "running on %s\n",
             tid, numTasks, here.name);
  }
```

```
prompt> chpl taskParallel.chpl -o taskParallel
prompt> ./taskParallel --numLocales=2
Hello from task 1 of 2 running on n1033
Hello from task 2 of 2 running on n1032
Hello from task 2 of 2 running on n1033
Hello from task 1 of 2 running on n1032
```


Copyright 2017 Cray Inc.

Task Parallelism and Locality, by example

Control of Locality/Affinity

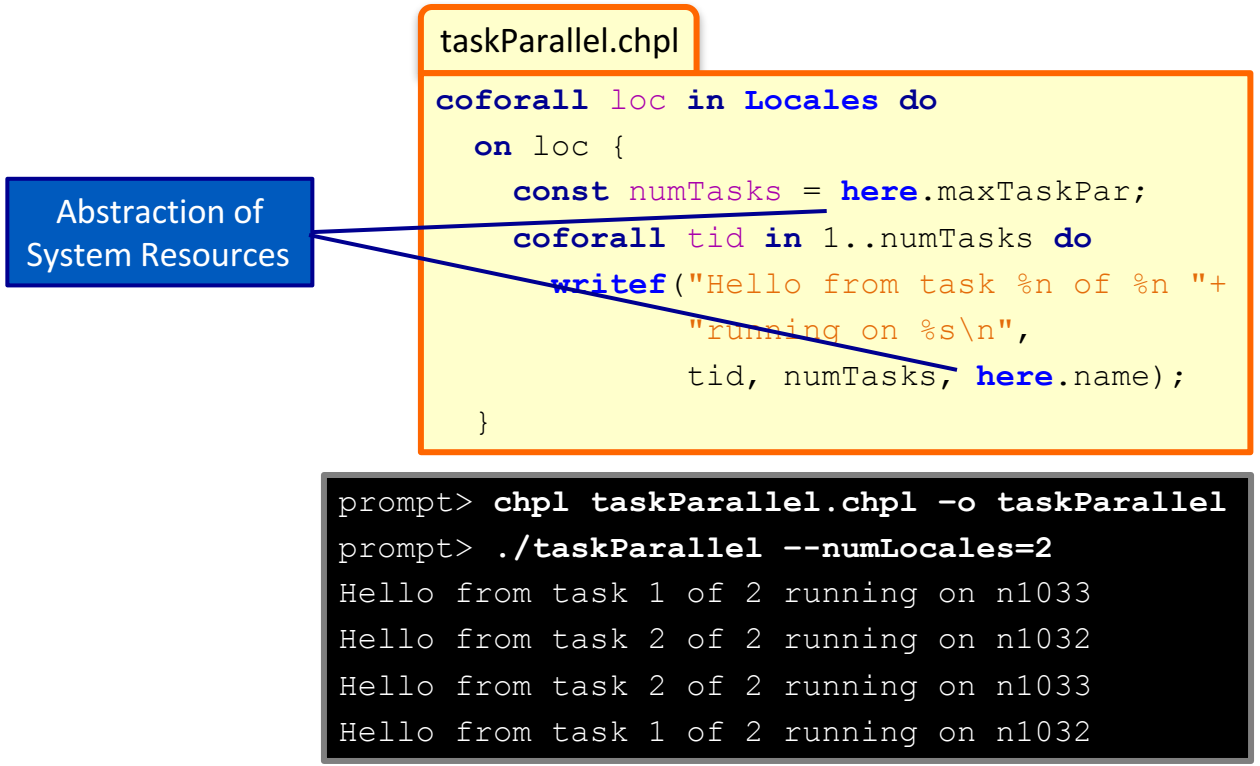
taskParallel.chpl

```
coforall loc in Locales do
  on loc {
    const numTasks = here.maxTaskPar;
    coforall tid in 1..numTasks do
      writef("Hello from task %n of %n "+
             "running on %s\n",
             tid, numTasks, here.name);
  }
```

```
prompt> chpl taskParallel.chpl -o taskParallel
prompt> ./taskParallel --numLocales=2
Hello from task 1 of 2 running on n1033
Hello from task 2 of 2 running on n1032
Hello from task 2 of 2 running on n1033
Hello from task 1 of 2 running on n1032
```


Copyright 2017 Cray Inc.

Task Parallelism and Locality, by example



Copyright 2017 Cray Inc.

Task Parallelism and Locality, by example

High-Level Task Parallelism

taskParallel.chpl

```
coforall loc in Locales do
  on loc {
    const numTasks = here.maxTaskPar;
    coforall tid in 1..numTasks do
      writef("Hello from task %n of %n "+
             "running on %s\n",
             tid, numTasks, here.name);
  }
```

```
prompt> chpl taskParallel.chpl -o taskParallel
prompt> ./taskParallel --numLocales=2
Hello from task 1 of 2 running on n1033
Hello from task 2 of 2 running on n1032
Hello from task 2 of 2 running on n1033
Hello from task 1 of 2 running on n1032
```


Copyright 2017 Cray Inc.

Task Parallelism and Locality, by example

taskParallel.chpl

```
coforall loc in Locales do
  on loc {
    const numTasks = here.maxTaskPar;
    coforall tid in 1..numTasks do
      writef("Hello from task %n of %n "+
             "running on %s\n",
             tid, numTasks, here.name);
  }
```

Not seen here:

Data-centric task coordination via atomic and full/empty vars

```
prompt> chpl taskParallel.chpl -o taskParallel
prompt> ./taskParallel --numLocales=2
Hello from task 1 of 2 running on n1033
Hello from task 2 of 2 running on n1032
Hello from task 2 of 2 running on n1033
Hello from task 1 of 2 running on n1032
```

Copyright 2017 Cray Inc.

Task Parallelism and Locality, by example

taskParallel.chpl

```
coforall loc in Locales do
  on loc {
    const numTasks = here.maxTaskPar;
    coforall tid in 1..numTasks do
      writef("Hello from task %n of %n "+
             "running on %s\n",
             tid, numTasks, here.name);
  }
```

```
prompt> chpl taskParallel.chpl -o taskParallel
prompt> ./taskParallel --numLocales=2
Hello from task 1 of 2 running on n1033
Hello from task 2 of 2 running on n1032
Hello from task 2 of 2 running on n1033
Hello from task 1 of 2 running on n1032
```


Copyright 2017 Cray Inc.

Parallelism and Locality: Distinct in Chapel

- This is a **parallel**, but local program:

```
coforall i in 1..msgs do
  writeln("Hello from task ", i);
```

- This is a **distributed**, but serial program:

```
writeln("Hello from locale 0!");
on Locales[1] do writeln("Hello from locale 1!");
on Locales[2] do writeln("Hello from locale 2!");
```

- This is a **distributed parallel** program:

```
coforall i in 1..msgs do
  on Locales[i%numLocales] do
    writeln("Hello from task ", i,
            " running on locale ", here.id);
```


Copyright 2017 Cray Inc.

Partitioned Global Address Space (PGAS) Languages
(Or more accurately: partitioned global namespace languages)

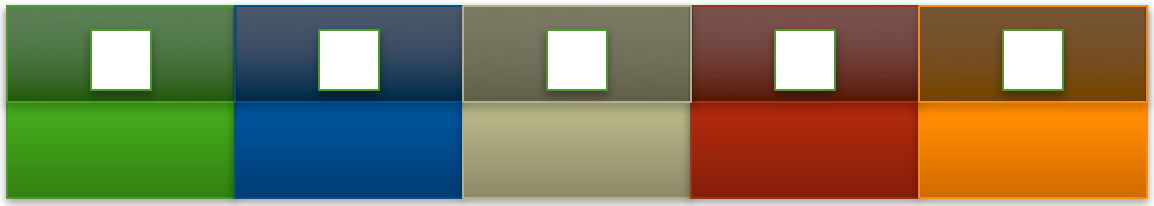
- **abstract concept:**
  - support a shared namespace on distributed memory
    - permit parallel tasks to access remote variables by naming them
  - establish a strong sense of ownership
    - every variable has a well-defined location
    - local variables are cheaper to access than remote ones
- **traditional PGAS languages have been SPMD in nature**
  - best-known examples: Fortran 2008's co-arrays, Unified Parallel C (UPC)

Copyright 2017 Cray Inc.

SPMD PGAS Languages (using a pseudo-language, not Chapel)

```
shared int i(*);          // declare a shared variable i
```

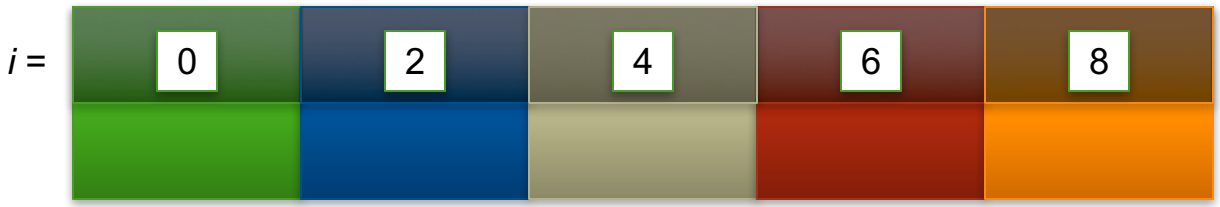
i =



Copyright 2017 Cray Inc.

SPMD PGAS Languages (using a pseudo-language, not Chapel)

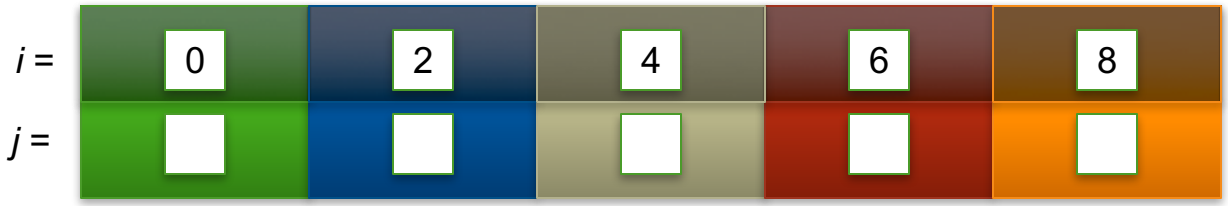
```
shared int i(*);          // declare a shared variable i
function main() {
  i = 2*this_image();     // each image initializes its copy
```



Copyright 2017 Cray Inc.

SPMD PGAS Languages (using a pseudo-language, not Chapel)

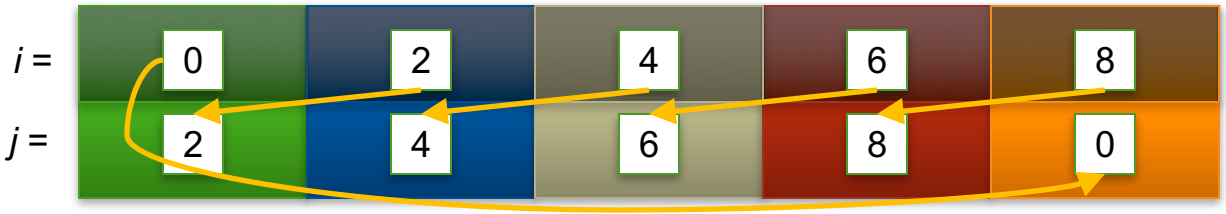
```
shared int i(*);          // declare a shared variable i
function main() {
  i = 2*this_image();     // each image initializes its copy

  private int j;          // declare a private variable j
```



Copyright 2017 Cray Inc.

SPMD PGAS Languages (using a pseudo-language, not Chapel)

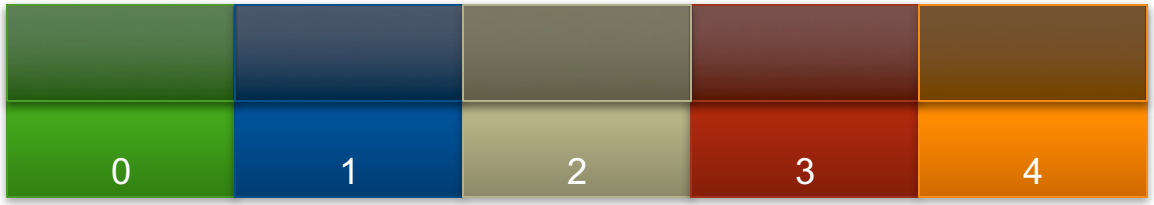
```
shared int i(*);          // declare a shared variable i
function main() {
  i = 2*this_image();     // each image initializes its copy
  barrier();
  private int j;          // declare a private variable j
  j = i( (this_image()+1) % num_images() );
    // ^^ access our neighbor's copy of i; compiler and runtime implement the communication
    // Q: How did we know our neighbor had an i?
    // A: Because it's SPMD – we're all running the same program so if we have an i, so do they.
```



Copyright 2017 Cray Inc.

Chapel and PGAS

- **Chapel is PGAS, but unlike most, it's not inherently SPMD**
  - never think about "the other copies of the program"
  - "global name/address space" comes from lexical scoping
    - as in traditional languages, each declaration yields one variable
    - variables are stored on the locale where the task declaring it is executing

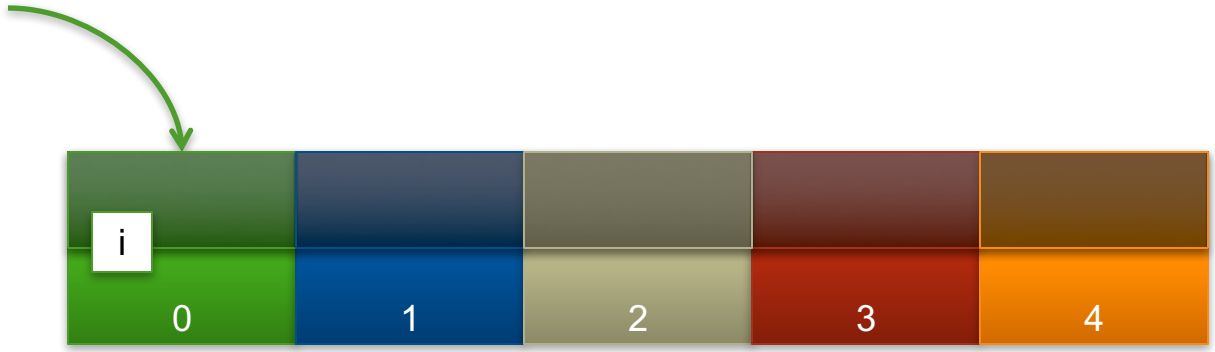


Locales (think: "compute nodes")

Copyright 2017 Cray Inc.

Chapel: Scoping and Locality

```
var i: int;
```

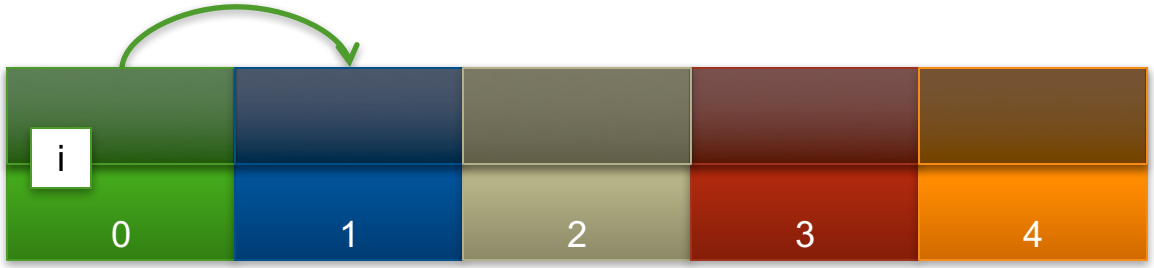


Locales (think: “compute nodes”)

Copyright 2017 Cray Inc.

Chapel: Scoping and Locality

```
var i: int;
on Locales[1] {
```

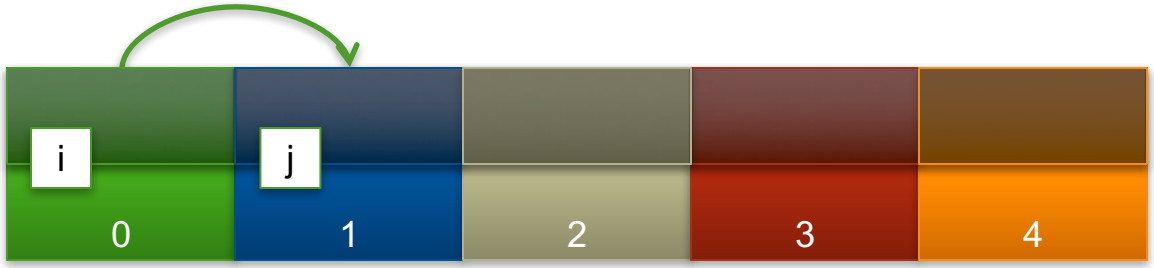


Locales (think: “compute nodes”)

Copyright 2017 Cray Inc.

Chapel: Scoping and Locality

```
var i: int;
on Locales[1] {
  var j: int;
```

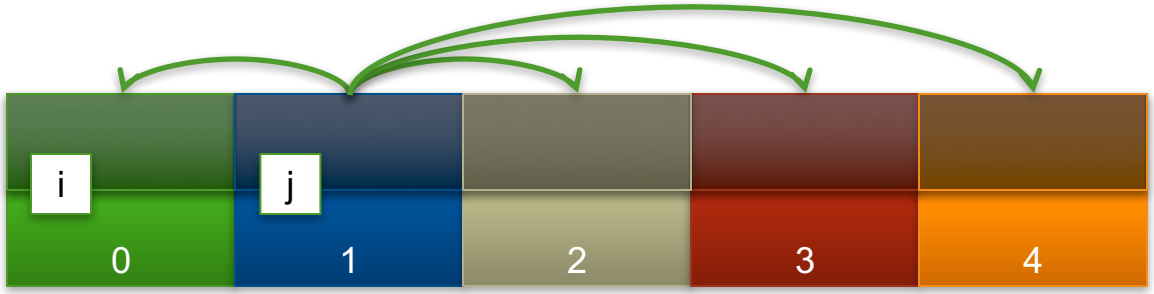


Locales (think: “compute nodes”)

Copyright 2017 Cray Inc.

Chapel: Scoping and Locality

```
var i: int;
on Locales[1] {
  var j: int;
  coforall loc in Locales {
    on loc {
```

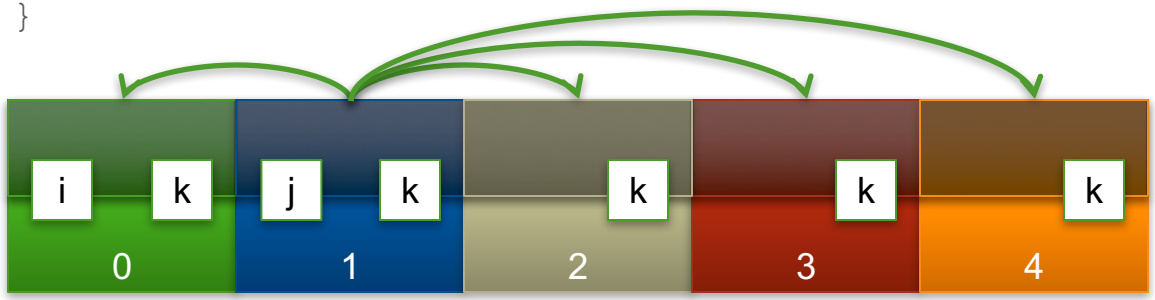


Locales (think: “compute nodes”)

Copyright 2017 Cray Inc.

Chapel: Scoping and Locality

```
var i: int;
on Locales[1] {
  var j: int;
  coforall loc in Locales {
    on loc {
      var k: int;
      …
    }
  }
}
```

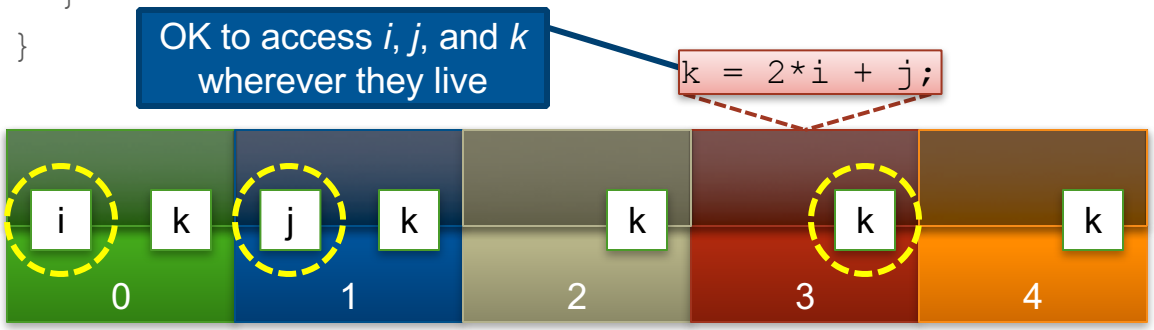


Locales (think: “compute nodes”)

Copyright 2017 Cray Inc.

Chapel: Scoping and Locality

```
var i: int;
on Locales[1] {
  var j: int;
  coforall loc in Locales {
    on loc {
      var k: int;
      k = 2*i + j;
    }
  }
}
```



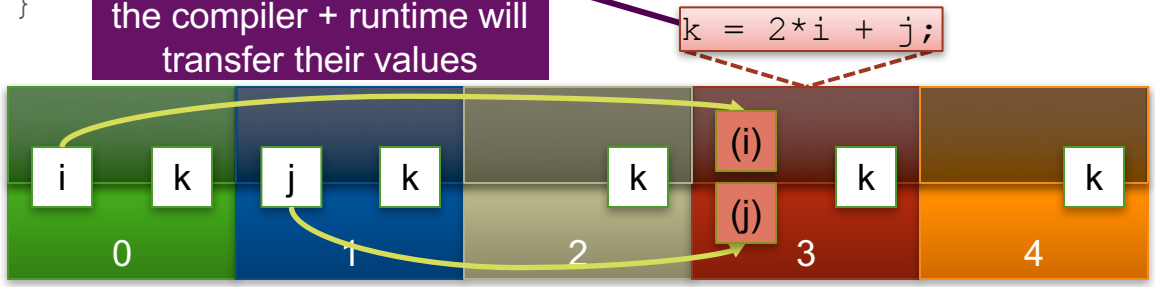
Locales (think: "compute nodes")

Copyright 2017 Cray Inc.

Chapel: Scoping and Locality

```
var i: int;
on Locales[1] {
  var j: int;
  coforall loc in Locales {
    on loc {
      var k: int;
      k = 2*i + j;
    }
  }
}
```

here, *i* and *j* are remote, so the compiler + runtime will transfer their values

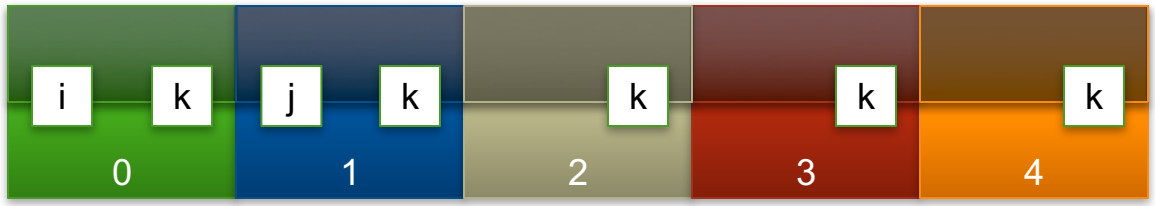


Locales (think: "compute nodes")

Copyright 2017 Cray Inc.

Chapel: Locality queries

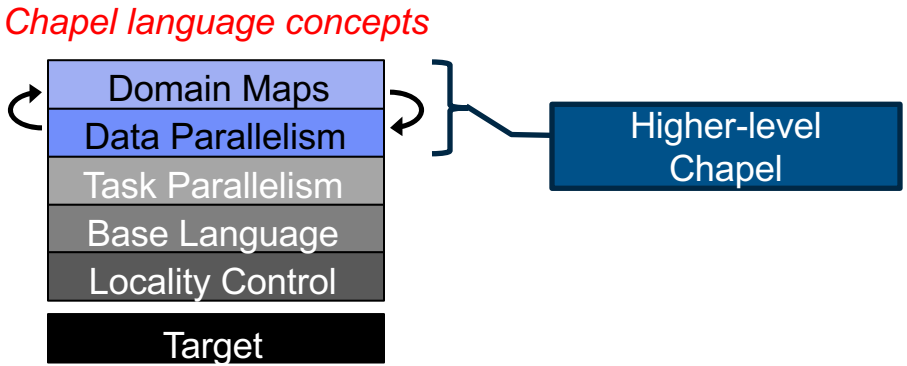
```
var i: int;
on Locales[1] {
  var j: int;
  coforall loc in Locales {
    on loc {
      var k: int;

      …here…          // query the locale on which this task is running
      …j.locale…      // query the locale on which j is stored
    }
}
```



Locales (think: “compute nodes”)

Copyright 2017 Cray Inc.

Higher-Level Features



Copyright 2017 Cray Inc.

Data Parallelism, by example

dataParallel.chpl

```
config const n = 1000;
var D = {1..n, 1..n};

var A: [D] real;
forall (i,j) in D do
  A[i,j] = i + (j - 0.5)/n;
writeln(A);
```

```
prompt> chpl dataParallel.chpl -o dataParallel
prompt> ./dataParallel --n=5
1.1 1.3 1.5 1.7 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
```


Copyright 2017 Cray Inc.

Data Parallelism, by example

Domains (Index Sets)

dataParallel.chpl

```
config const n = 1000;
var D = {1..n, 1..n};

var A: [D] real;
forall (i,j) in D do
  A[i,j] = i + (j - 0.5)/n;
writeln(A);
```

```
prompt> chpl dataParallel.chpl -o dataParallel
prompt> ./dataParallel --n=5
1.1 1.3 1.5 1.7 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
```


Copyright 2017 Cray Inc.

Data Parallelism, by example

Arrays

dataParallel.chpl

```
config const n = 1000;
var D = {1..n, 1..n};

var A: [D] real;
forall (i,j) in D do
  A[i,j] = i + (j - 0.5)/n;
writeln(A);
```

```
prompt> chpl dataParallel.chpl -o dataParallel
prompt> ./dataParallel --n=5
1.1 1.3 1.5 1.7 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
```


Copyright 2017 Cray Inc.

Data Parallelism, by example

Data-Parallel Forall Loops

dataParallel.chpl

```
config const n = 1000;
var D = {1..n, 1..n};

var A: [D] real;
forall (i,j) in D do
  A[i,j] = i + (j - 0.5)/n;
writeln(A);
```

```
prompt> chpl dataParallel.chpl -o dataParallel
prompt> ./dataParallel --n=5
1.1 1.3 1.5 1.7 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
```


Copyright 2017 Cray Inc.

Distributed Data Parallelism, by example

Domain Maps
(Map Data Parallelism to the System)

dataParallel.chpl

```
use CyclicDist;
config const n = 1000;
var D = {1..n, 1..n}
        dmapped Cyclic(startIdx = (1,1));
var A: [D] real;
forall (i,j) in D do
  A[i,j] = i + (j - 0.5)/n;
writeln(A);
```

```
prompt> chpl dataParallel.chpl -o dataParallel
prompt> ./dataParallel --n=5 --numLocales=4
1.1 1.3 1.5 1.7 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
```


Copyright 2017 Cray Inc.

Distributed Data Parallelism, by example

dataParallel.chpl

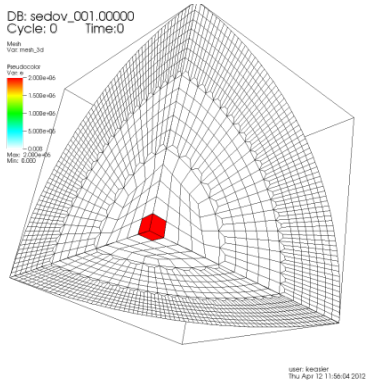
```
use CyclicDist;
config const n = 1000;
var D = {1..n, 1..n}
        dmapped Cyclic(startIdx = (1,1));
var A: [D] real;
forall (i,j) in D do
  A[i,j] = i + (j - 0.5)/n;
writeln(A);
```

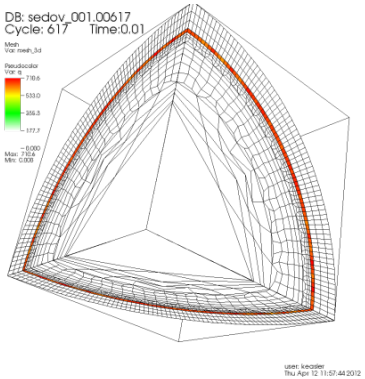
```
prompt> chpl dataParallel.chpl -o dataParallel
prompt> ./dataParallel --n=5 --numLocales=4
1.1 1.3 1.5 1.7 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
```

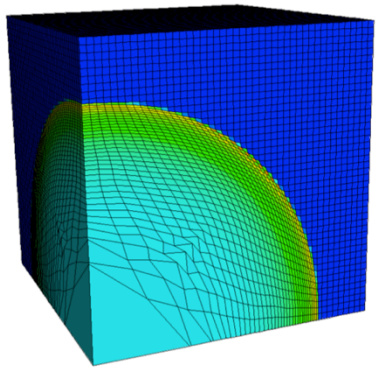

Copyright 2017 Cray Inc.

LULESH: a DOE Proxy Application

Goal: Solve one octant of the spherical Sedov problem (blast wave) using Lagrangian hydrodynamics for a single material







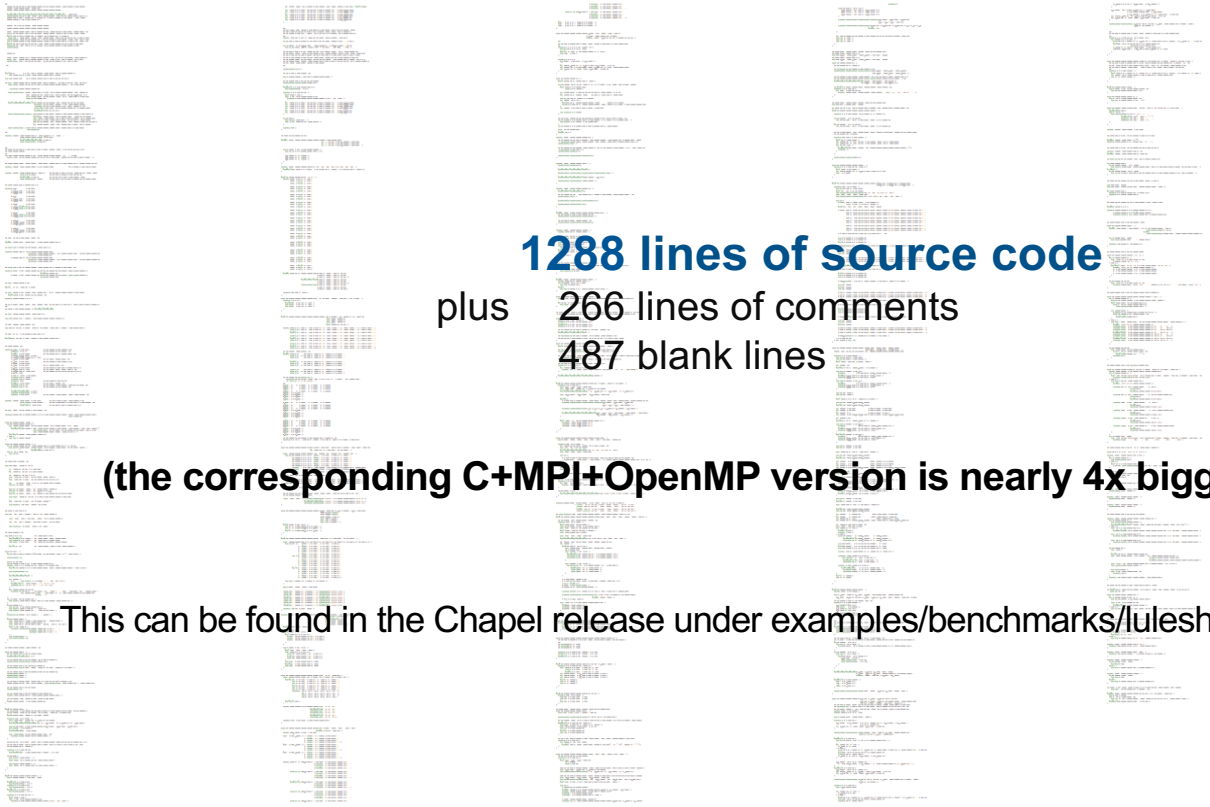
pictures courtesy of Rob Neely, Bert Still, Jeff Keasler, LLNL

Copyright 2017 Cray Inc.

LULESH in Chapel

Copyright 2017 Cray Inc.

LULESH in Chapel



1288 lines of source code

plus 266 lines of comments

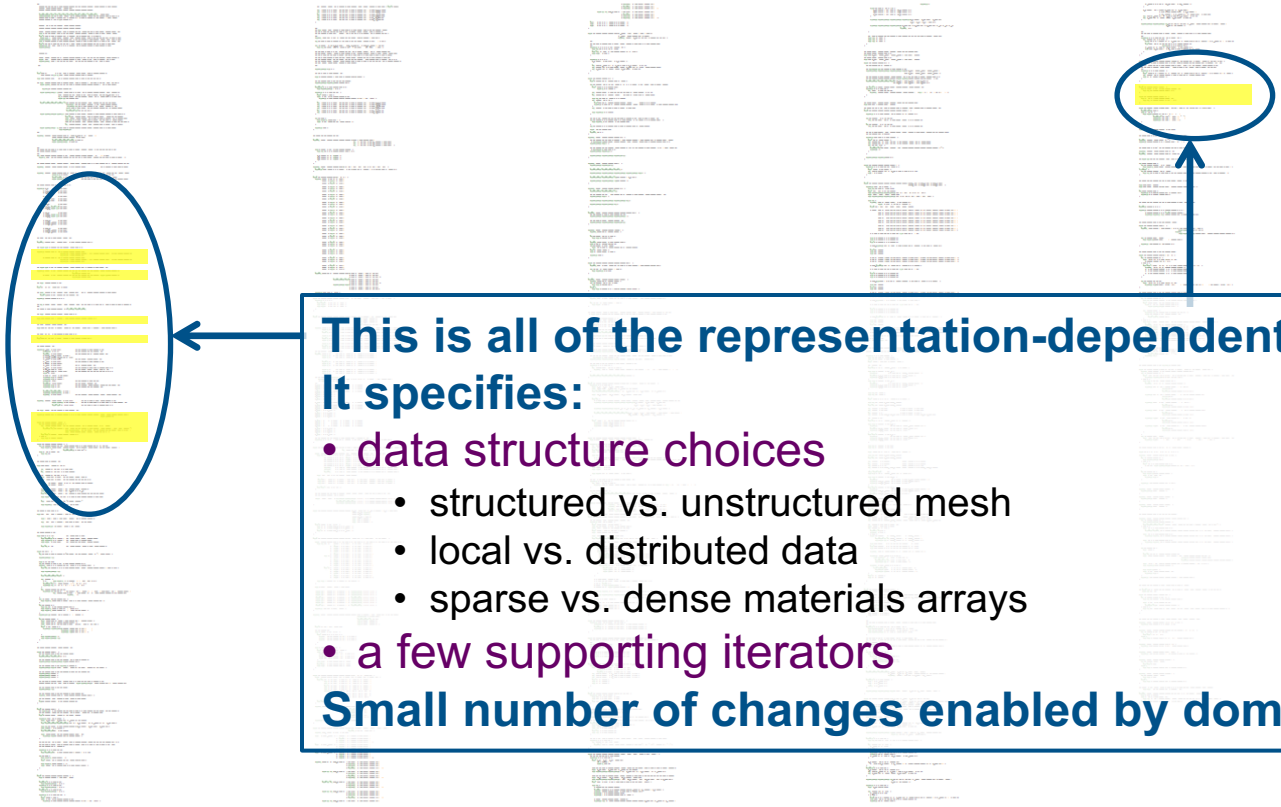
487 blank lines

(the corresponding C+MPI+OpenMP version is nearly 4x bigger)

This can be found in the Chapel release under examples/benchmarks/lulesh/*.chpl

Copyright 2017 Cray Inc.

LULESH in Chapel



This is all of the representation-dependent code.
It specifies:

- data structure choices
  - structured vs. unstructured mesh
  - local vs. distributed data
  - sparse vs. dense materials arrays
- a few supporting iterators

Small number of changes enabled by domain maps

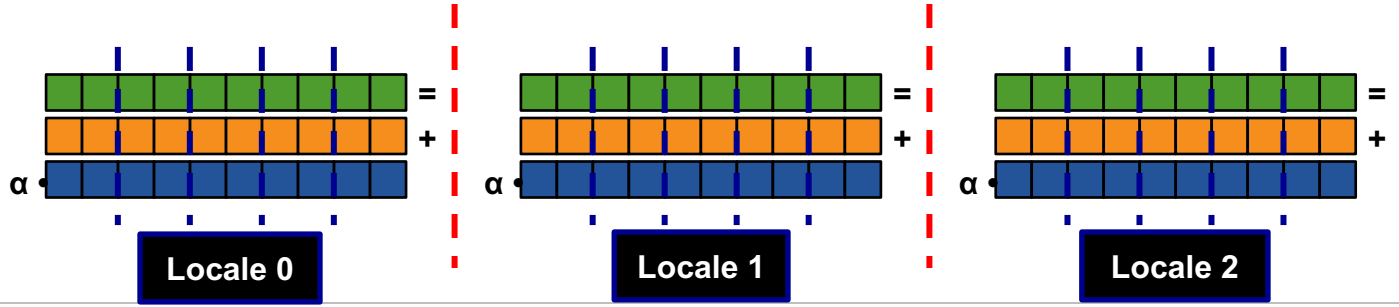
Copyright 2017 Cray Inc.

Domain Maps

Domain maps are “recipes” that instruct the compiler how to map the global view of a computation…


```
A = B + alpha * C;
```

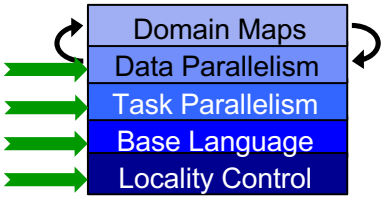
…to the target locales’ memory and processors:



Copyright 2017 Cray Inc.

Chapel's Domain Map Philosophy

1. **Chapel provides a library of standard domain maps**
   - to support common array implementations effortlessly
2. **Expert users can write their own domain maps in Chapel**
   - to cope with any shortcomings in our standard library



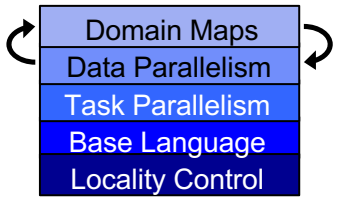
3. **Chapel's standard domain maps are written using the end-user framework**
   - to avoid a performance cliff between "built-in" and user-defined cases
   - in fact *every* Chapel array is implemented using this framework

Copyright 2017 Cray Inc.

Chapel's Multiresolution Philosophy

Multiresolution Design: **Support multiple tiers of features**

- higher levels for programmability, productivity
- lower levels for greater degrees of control



- build the higher-level concepts in terms of the lower
- permit users to intermix layers arbitrarily

Copyright 2017 Cray Inc.

Two Other Multiresolution Features

1) **parallel iterators:** User-specified forall-loop implementations
   - how many tasks to use and where they run
   - how iterations are divided between the tasks
   - how to zipper with other parallel iterators

2) **locale models:** User-specified locale types for new node architectures
   - how do I manage memory, create tasks, communicate, …

Like domain maps, these are…

…written in Chapel by expert users using lower-level features

…available to the end-user via higher-level abstractions

Copyright 2017 Cray Inc.

This Talk's Takeaways

If you design a parallel programming language…

…create attractive ways of expressing parallelism and locality

- these are key concerns—shouldn't be yet another library call / pragma

…support first-class index sets

- very expressive concept for users (not seen here, today)
- enable the implementation to reason about alignment (ditto)

…support multiresolution features

- give users the ability to develop their own parallel policies

Copyright 2017 Cray Inc.

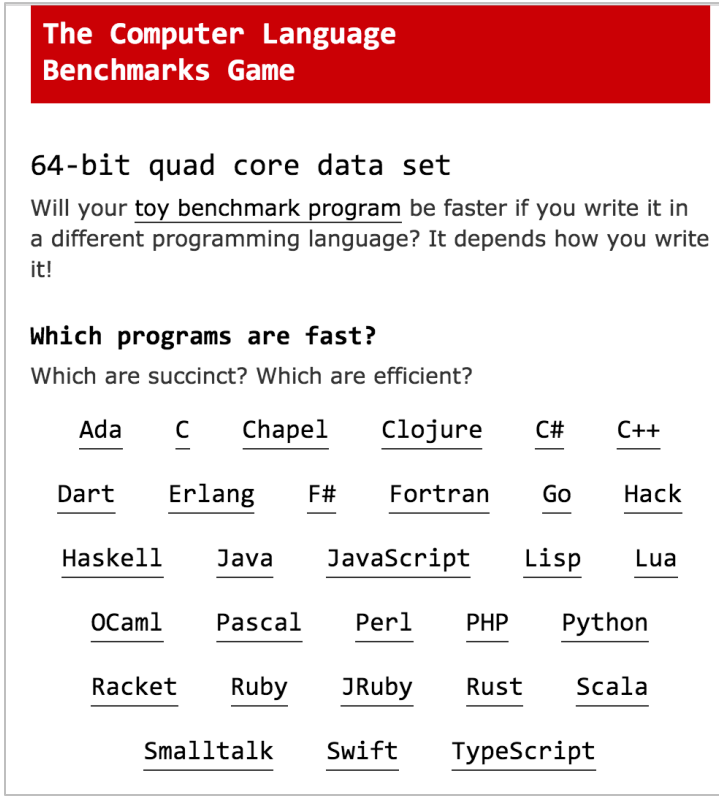
Puttin' me down for thinking of someone new
Always the same
Playin' your game
Drive me insane…

Computer Language Benchmarks Game Results

Your Time is Gonna Come
Led Zeppelin
Led Zeppelin

Copyright 2017 Cray Inc.

Computer Language Benchmarks Game (CLBG)



Website supporting cross-language comparisons

- 13 toy benchmark programs
  - exercise key computational idioms
  - specific approach prescribed

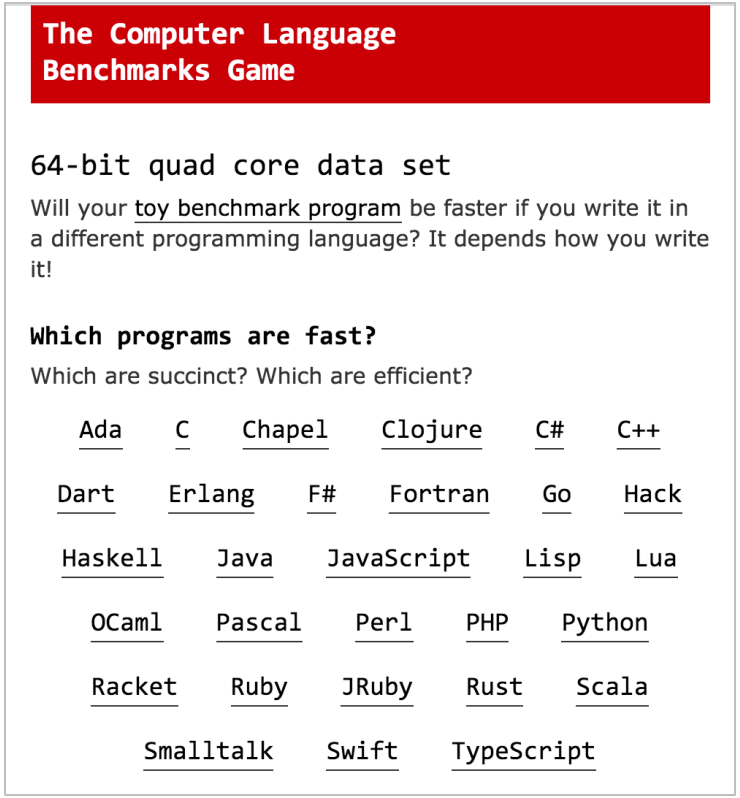
Take results with a grain of salt

- your mileage may vary

That said, it is one of the only such games in town…

Copyright 2017 Cray Inc.

Computer Language Benchmarks Game (CLBG)

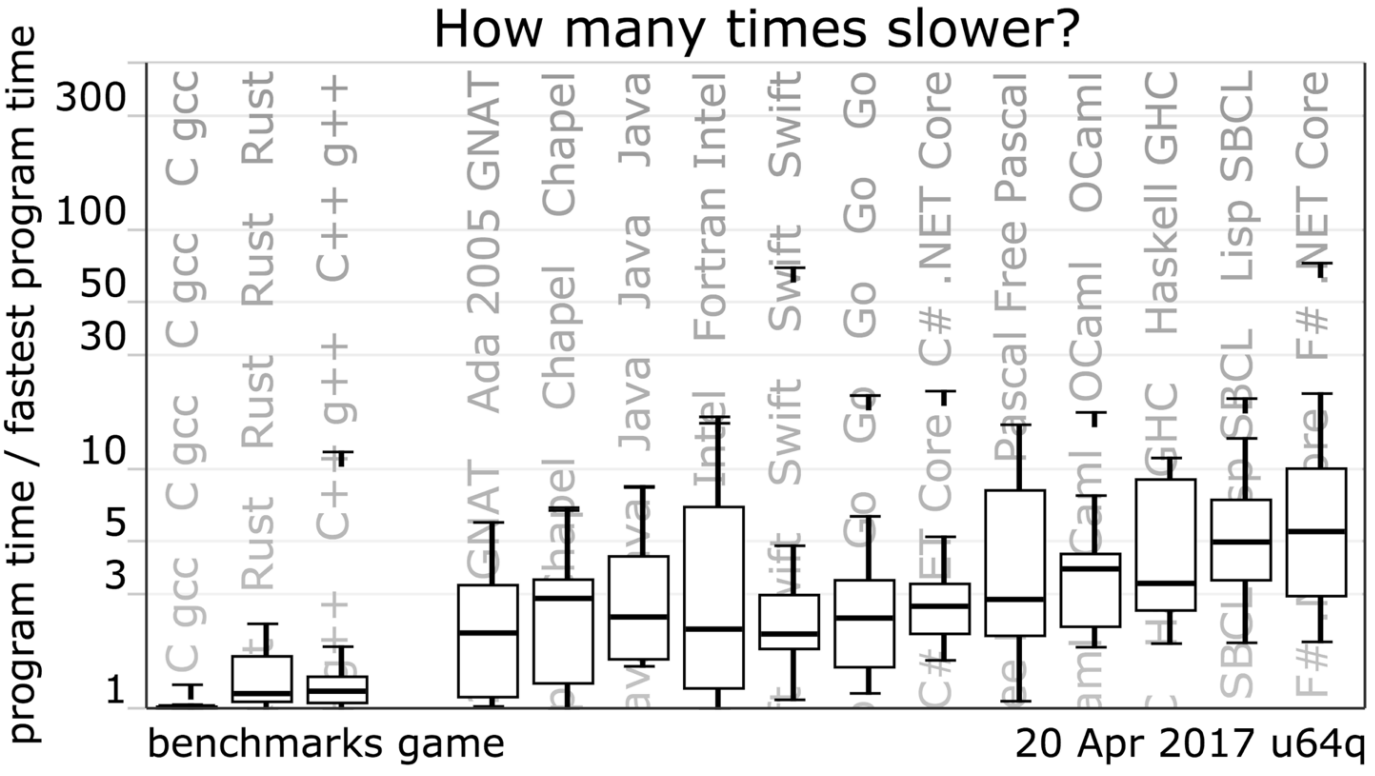


Chapel's approach to the CLBG:

- striving for elegance over heroism
  - ideally: "Want to learn how program xyz works? Read the Chapel version."

Copyright 2017 Cray Inc.

CLBG: Relative Performance Summary



Copyright 2017 Cray Inc.

CLBG: Website

Can sort results by execution time, code size, memory or CPU use:

The Computer Language Benchmarks Game

chameneos-redux

description

program source code, command-line and measurements

×	source	secs	mem	gz	cpu	cpu load
1.0	**C** gcc #5	**0.60**	820	2863	2.37	100% 100% 98% 100%
1.2	**C++** g++ #5	**0.70**	3,356	1994	2.65	100% 100% 91% 92%
1.7	**Lisp** SBCL #3	**1.01**	55,604	2907	3.93	97% 96% 99% 99%
2.3	**Chapel** #2	**1.39**	76,564	1210	5.43	99% 99% 98% 99%
3.3	**Rust** #2	**2.01**	56,936	2882	7.81	97% 98% 98% 98%
5.6	C++ g++ #2	3.40	1,880	2016	11.88	100% 51% 100% 100%
6.8	Chapel	4.09	66,584	1199	16.25	100% 100% 100% 100%
8.0	**Java** #4	**4.82**	37,132	1607	16.73	98% 98% 54% 99%
8.5	**Haskell** GHC	**5.15**	8,596	989	9.26	79% 100% 2% 2%
10	Java	6.13	53,760	1770	8.78	42% 45% 41% 16%
10	Haskell GHC #4	6.34	6,908	989	12.67	99% 100% 2% 1%
11	**C#** .NET Core	**6.59**	86,076	1400	22.96	99% 82% 78% 91%
11	**Go**	**6.90**	832	1167	24.19	100% 96% 56% 100%
13	Go #2	7.59	1,384	1408	27.65	91% 99% 99% 78%
13	Java #3	7.94	53,232	1267	26.86	54% 96% 98% 94%

The Computer Language Benchmarks Game

chameneos-redux

description

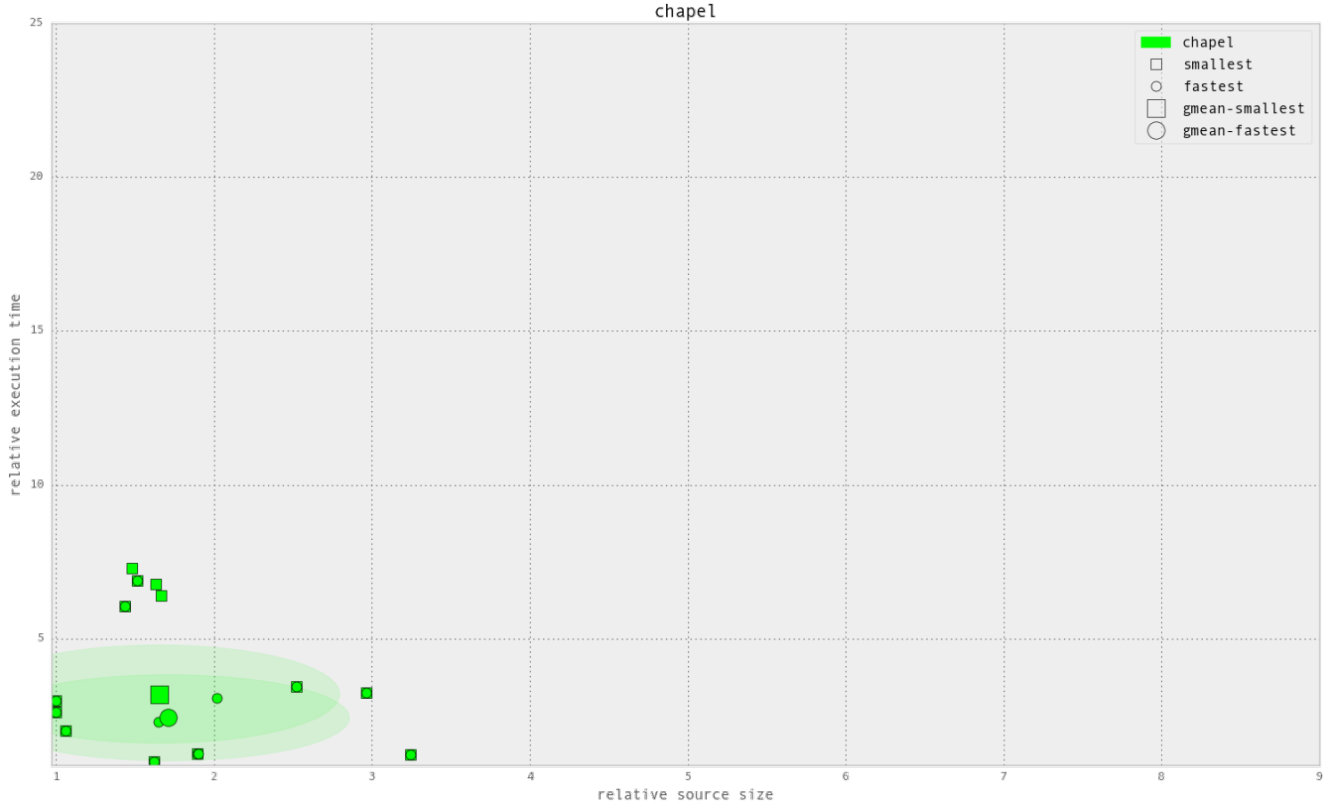
program source code, command-line and measurements

×	source	secs	mem	gz	cpu	cpu load
1.0	**Erlang**	58.90	28,668	**734**	131.19	62% 60% 51% 53%
1.0	**Erlang** HiPE	59.39	25,784	**734**	131.58	60% 56% 56% 54%
1.1	**Perl** #4	5 min	14,084	**785**	7 min	40% 40% 29% 28%
1.1	**Racket**	5 min	132,120	**791**	5 min	1% 0% 0% 100%
1.1	Racket #2	175.88	116,488	842	175.78	100% 1% 1% 0%
1.2	**Python 3** #2	236.84	7,908	**866**	5 min	24% 48% 27% 45%
1.3	**Ruby**	90.52	9,396	**920**	137.53	35% 35% 35% 34%
1.3	**Ruby** JRuby	48.78	628,968	**928**	112.15	65% 60% 49% 58%
1.3	**Go** #5	11.05	832	**957**	32.48	75% 74% 75% 73%
1.3	**Haskell** GHC #4	6.34	6,908	**989**	12.67	99% 100% 2% 1%
1.3	Haskell GHC	5.15	8,596	989	9.26	79% 100% 2% 2%
1.6	**OCaml** #3					32% 38% 37% 39%
1.6	Go					100% 96% 56% 100%
1.6	**Chapel**					0% 100% 100% 100%
1.6	Chapel #2					99% 99% 98% 99%

gz == code size metric
strip comments and extra whitespace, then gzip

Copyright 2017 Cray Inc.

CLBG: Chapel entries

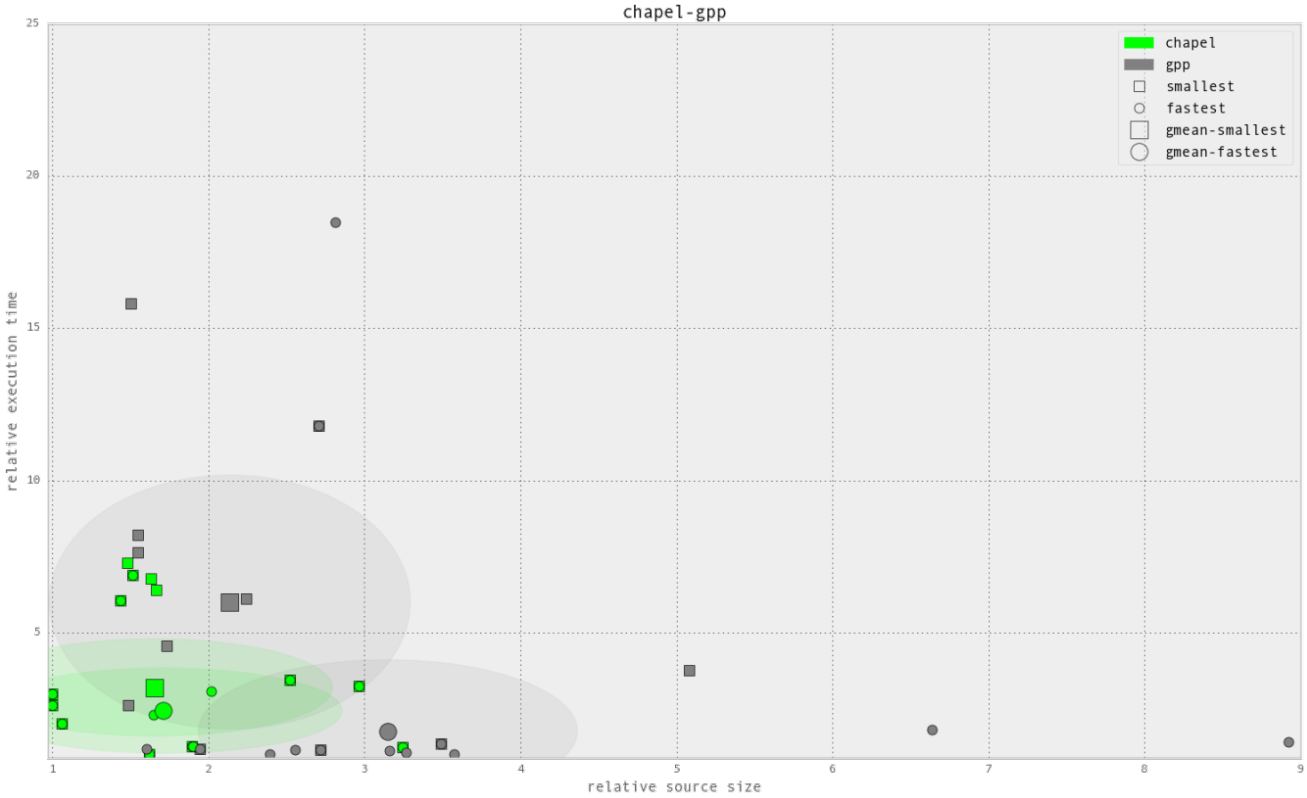


Copyright 2017 Cray Inc.

CLBG: Chapel vs. 9 key languages

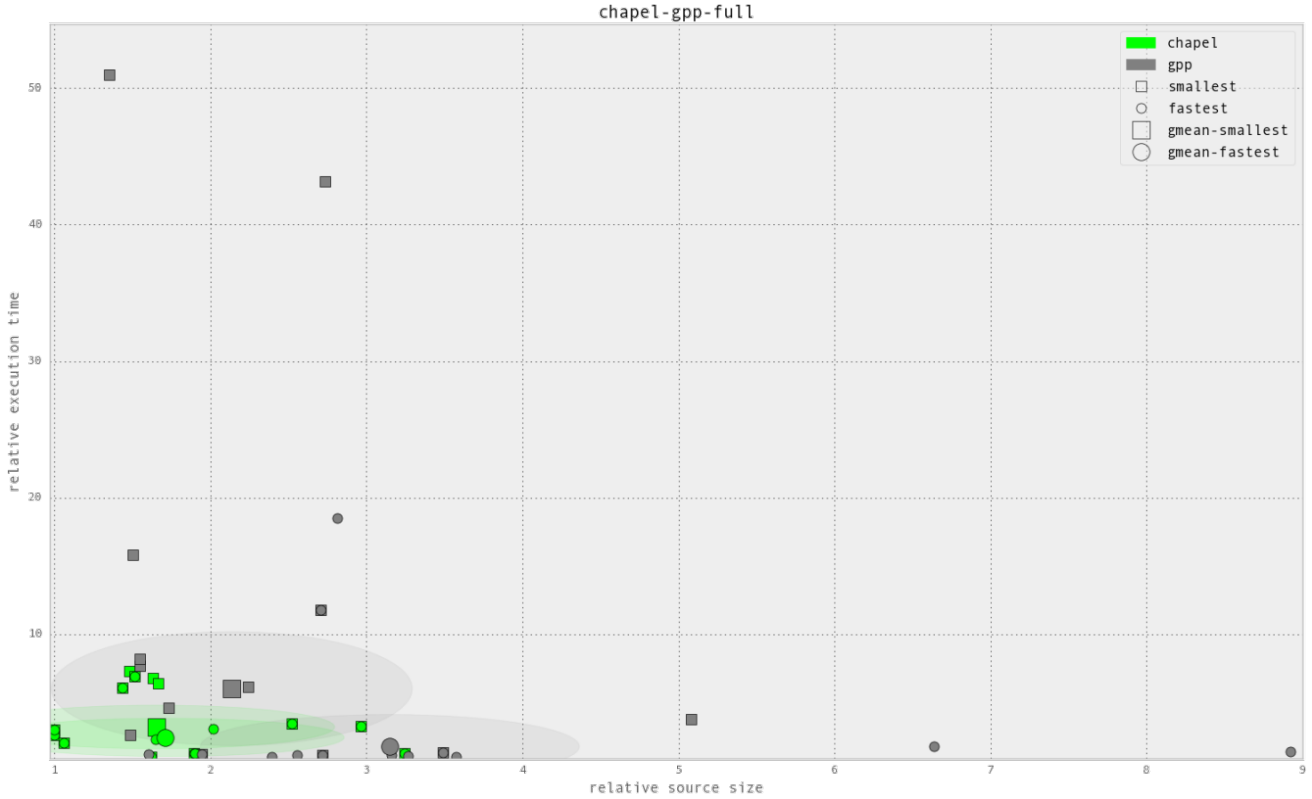
Copyright 2017 Cray Inc.

Chapel vs. C++



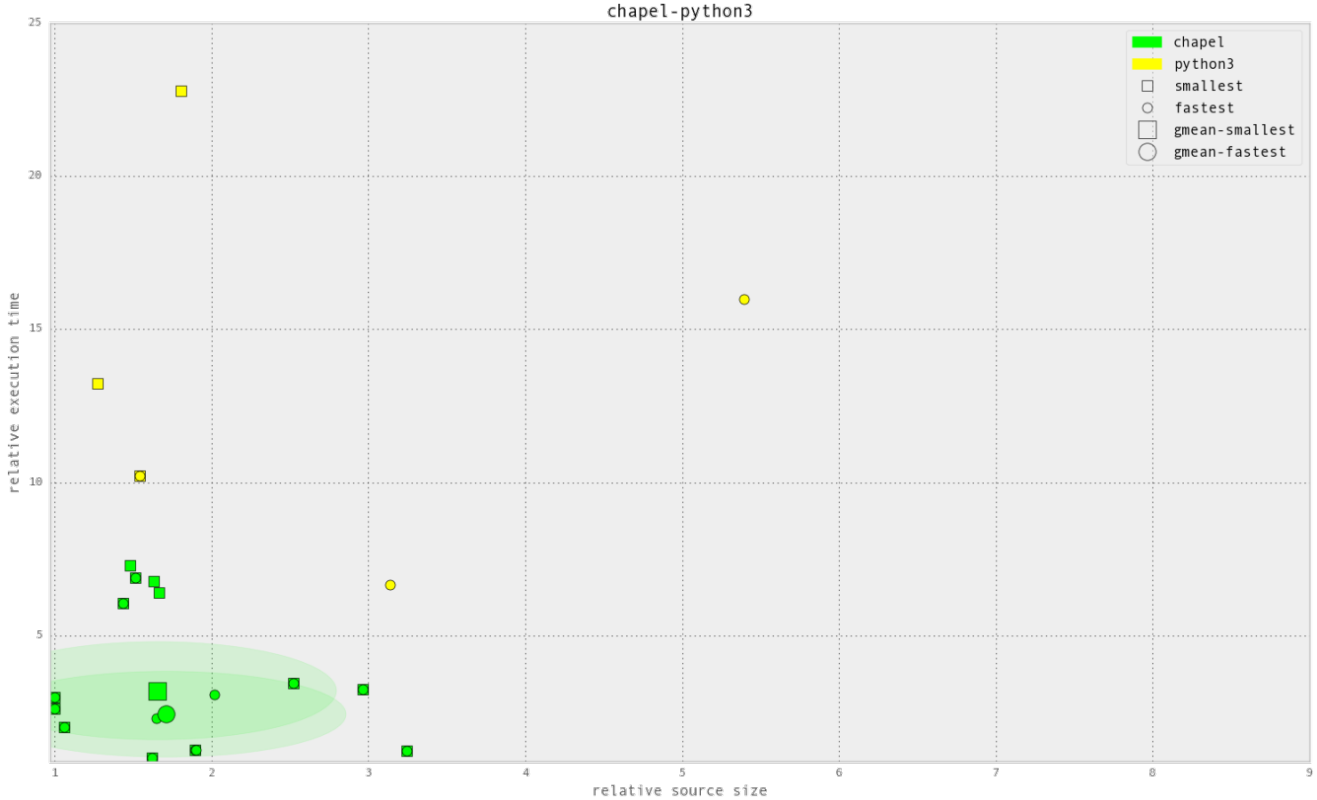
Copyright 2017 Cray Inc.

Chapel vs. C++ (zoomed out)



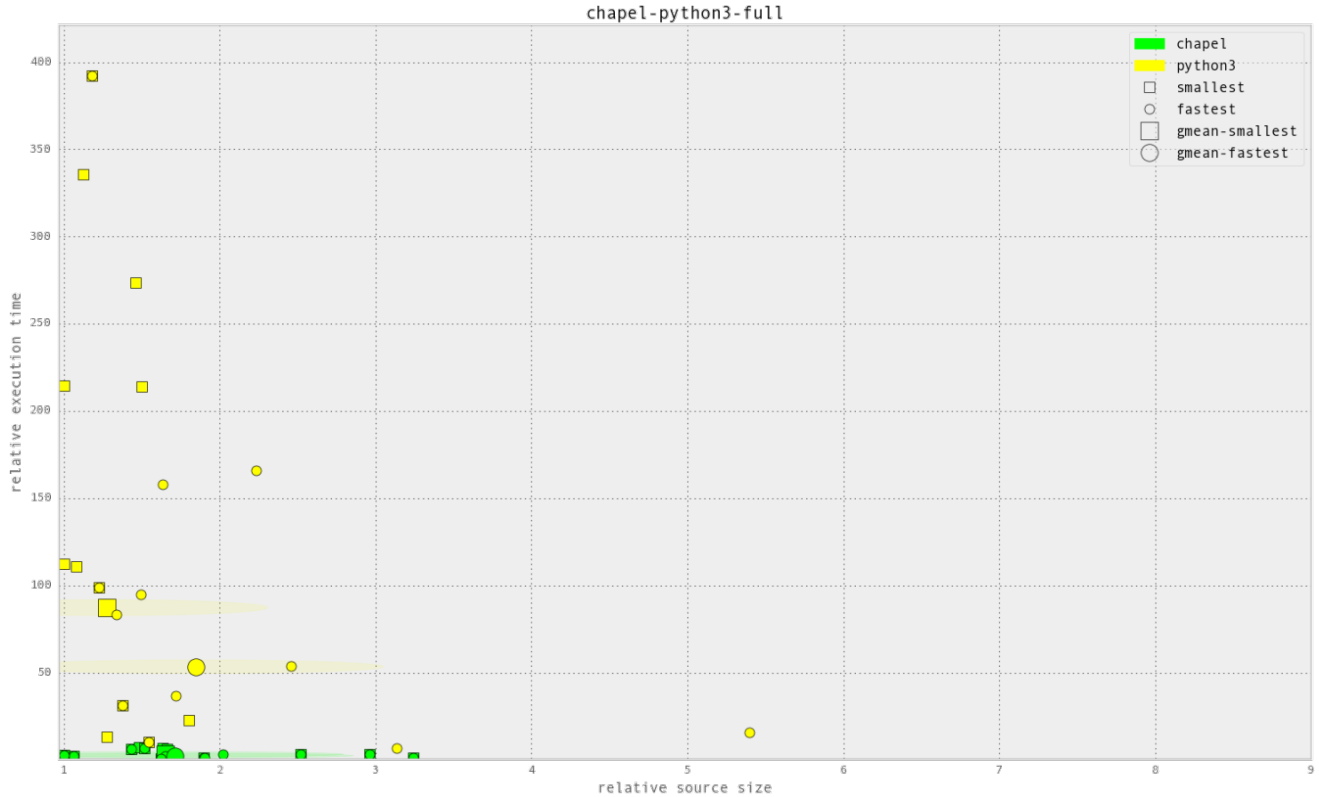
Copyright 2017 Cray Inc.

Chapel vs. Python



Copyright 2017 Cray Inc.

Chapel vs. Python (zoomed out)



Copyright 2017 Cray Inc.

CLBG: Qualitative Comparisons

Can also browse program source code (but this requires actual thought):

```
proc main() {
  printColorEquations();

  const group1 = [i in 1..popSize1] new Chameneos(i, ((i-1)%3):Color);
  const group2 = [i in 1..popSize2] new Chameneos(i, colors10[i]);

  cobegin {
    holdMeetings(group1, n);
    holdMeetings(group2, n);
  }

  print(group1);
  print(group2);

  for c in group1 do delete c;
  for c in group2 do delete c;
}


//
// Print the results of getNewColor() for all color pairs.
//
proc printColorEquations() {
  for c1 in Color do
    for c2 in Color do
      writeln(c1, " + ", c2, " -> ", getNewColor(c1, c2));
  writeln();
}


//
// Hold meetings among the population by creating a shared meeting
// place, and then creating per-chameneos tasks to have meetings.
//
proc holdMeetings(population, numMeetings) {
  const place = new MeetingPlace(numMeetings);

  coforall c in population do            // create a task per chameneos
    c.haveMeetings(place, population);

  delete place;
}
```

excerpt from 1210 gz Chapel entry

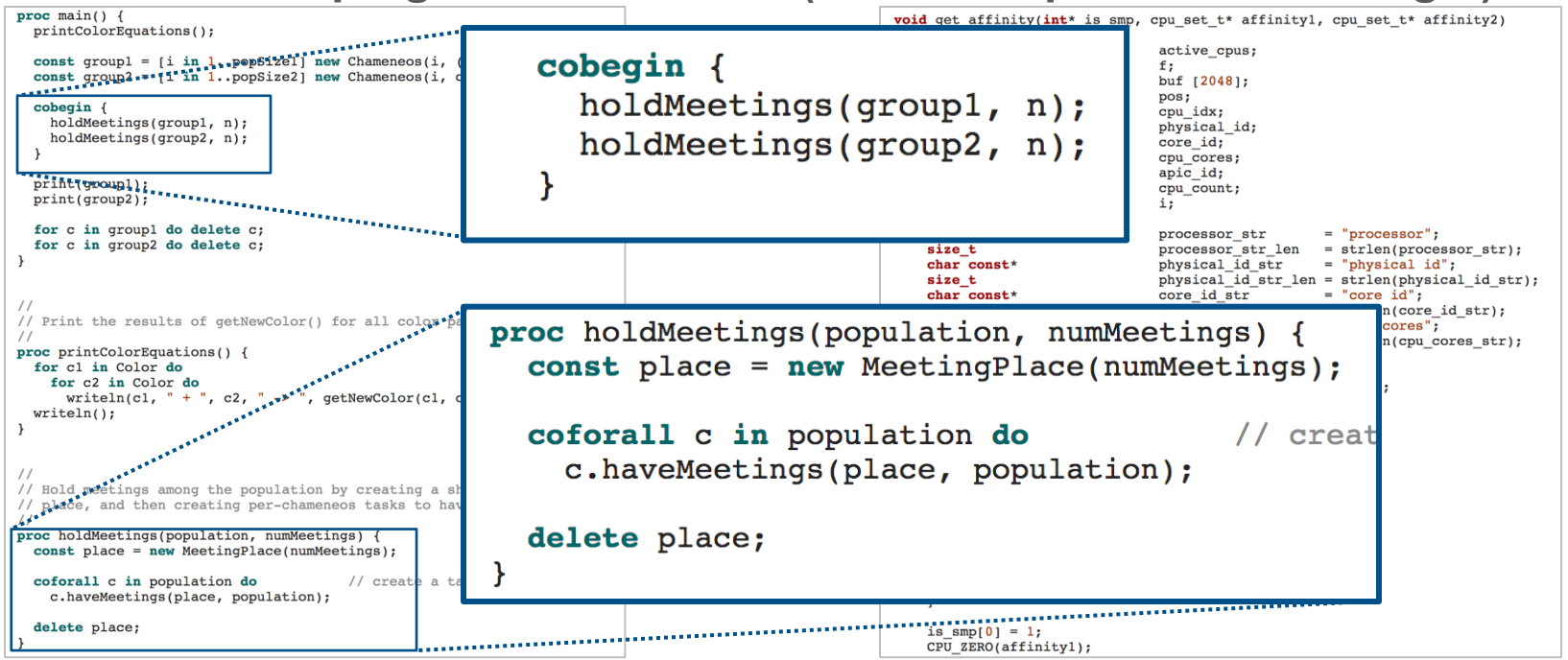
```
void get_affinity(int* is_smp, cpu_set_t* affinity1, cpu_set_t* affinity2)
{
    cpu_set_t                   active_cpus;
    FILE*                       f;
    char                        buf [2048];
    char const*                 pos;
    int                         cpu_idx;
    int                         physical_id;
    int                         core_id;
    int                         cpu_cores;
    int                         apic_id;
    size_t                      cpu_count;
    size_t                      i;

    char const*                 processor_str       = "processor";
    size_t                      processor_str_len   = strlen(processor_str);
    char const*                 physical_id_str     = "physical id";
    size_t                      physical_id_str_len = strlen(physical_id_str);
    char const*                 core_id_str         = "core id";
    size_t                      core_id_str_len     = strlen(core_id_str);
    char const*                 cpu_cores_str       = "cpu cores";
    size_t                      cpu_cores_str_len   = strlen(cpu_cores_str);

    CPU_ZERO(&active_cpus);
    sched_getaffinity(0, sizeof(active_cpus), &active_cpus);
    cpu_count = 0;
    for (i = 0; i != CPU_SETSIZE; i += 1)
    {
        if (CPU_ISSET(i, &active_cpus))
        {
            cpu_count += 1;
        }
    }

    if (cpu_count == 1)
    {
        is_smp[0] = 0;
        return;
    }

    is_smp[0] = 1;
    CPU_ZERO(affinity1);
```

excerpt from 2863 gz C gcc entry

Copyright 2017 Cray Inc.

CLBG: Qualitative Comparisons

Can also browse program source code (but this requires actual thought):



excerpt from 1210 gz Chapel entry

excerpt from 2863 gz C gcc entry

Copyright 2017 Cray Inc.

CLBG: Qualitative Comparisons

Can also browse program source code (but this requires actual thought):

```
proc main() {
char const*                     core_id_str          = "core id"
size_t                          core_id_str_len      = strlen(co
char const*                     cpu_cores_str        = "cpu core
size_t                          cpu_cores_str_len    = strlen(cp

CPU_ZERO(&active_cpus);
sched_getaffinity(0, sizeof(active_cpus), &active_cpus);
cpu_count = 0;
for (i = 0; i != CPU_SETSIZE; i += 1)
{
    if (CPU_ISSET(i, &active_cpus))
    {
        cpu_count += 1;
    }
}

if (cpu_count == 1)
{
    is_smp[0] = 0;
    return;
}
```

excerpt from 1210 gz Chapel entry

```
void get_affinity(int* is_smp, cpu_set_t* affinity1, cpu_set_t* affinity2)
{
    cpu_set_t                   active_cpus;
    FILE*                       f;
    char                        buf [2048];
    char const*                 pos;
    int                         cpu_idx;
    int                         physical_id;
    int                         core_id;
    int                         cpu_cores;
    int                         apic_id;
    size_t                      cpu_count;
    size_t                      i;

    char const*                 processor_str       = "processor";
    size_t                      processor_str_len   = strlen(processor_str);
    char const*                 physical_id_str     = "physical id";
    size_t                      physical_id_str_len = strlen(physical_id_str);
    char const*                 core_id_str         = "core id";
    size_t                      core_id_str_len     = strlen(core_id_str);
    char const*                 cpu_cores_str       = "cpu cores";
    size_t                      cpu_cores_str_len   = strlen(cpu_cores_str);

    CPU_ZERO(&active_cpus);
    sched_getaffinity(0, sizeof(active_cpus), &active_cpus);
    cpu_count = 0;
    for (i = 0; i != CPU_SETSIZE; i += 1)
    {
        if (CPU_ISSET(i, &active_cpus))
        {
            cpu_count += 1;
        }
    }

    if (cpu_count == 1)
    {
        is_smp[0] = 0;
        return;
    }

    is_smp[0] = 1;
    CPU_ZERO(affinity1);
```

excerpt from 2863 gz C gcc entry

Copyright 2017 Cray Inc.

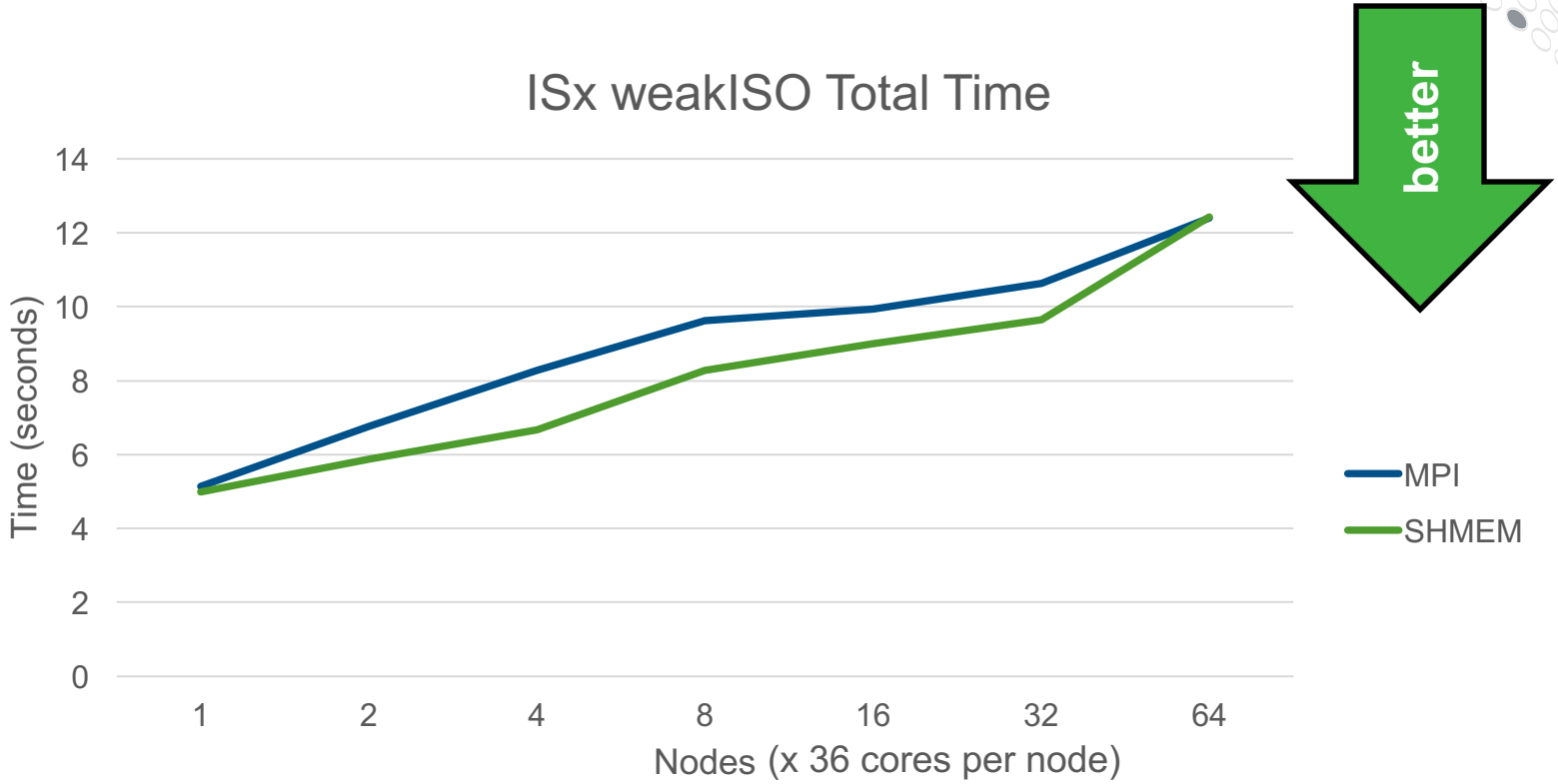
I have run
I have crawled
I have scaled…

Chapel Scalability Results

U2
I Still Haven't Found What I'm Looking For
The Joshua Tree

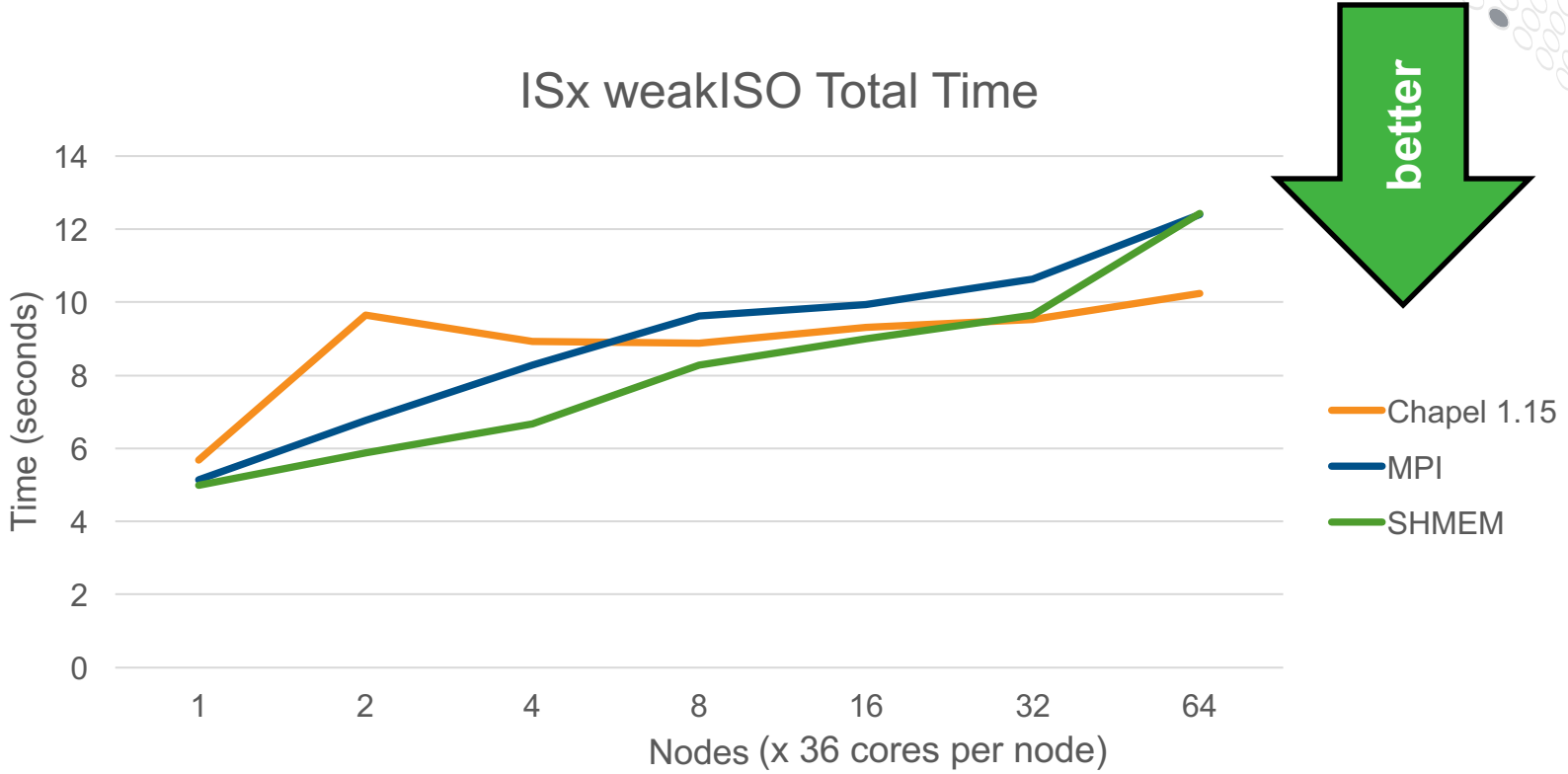
Copyright 2017 Cray Inc.

ISx Execution Time: MPI, SHMEM



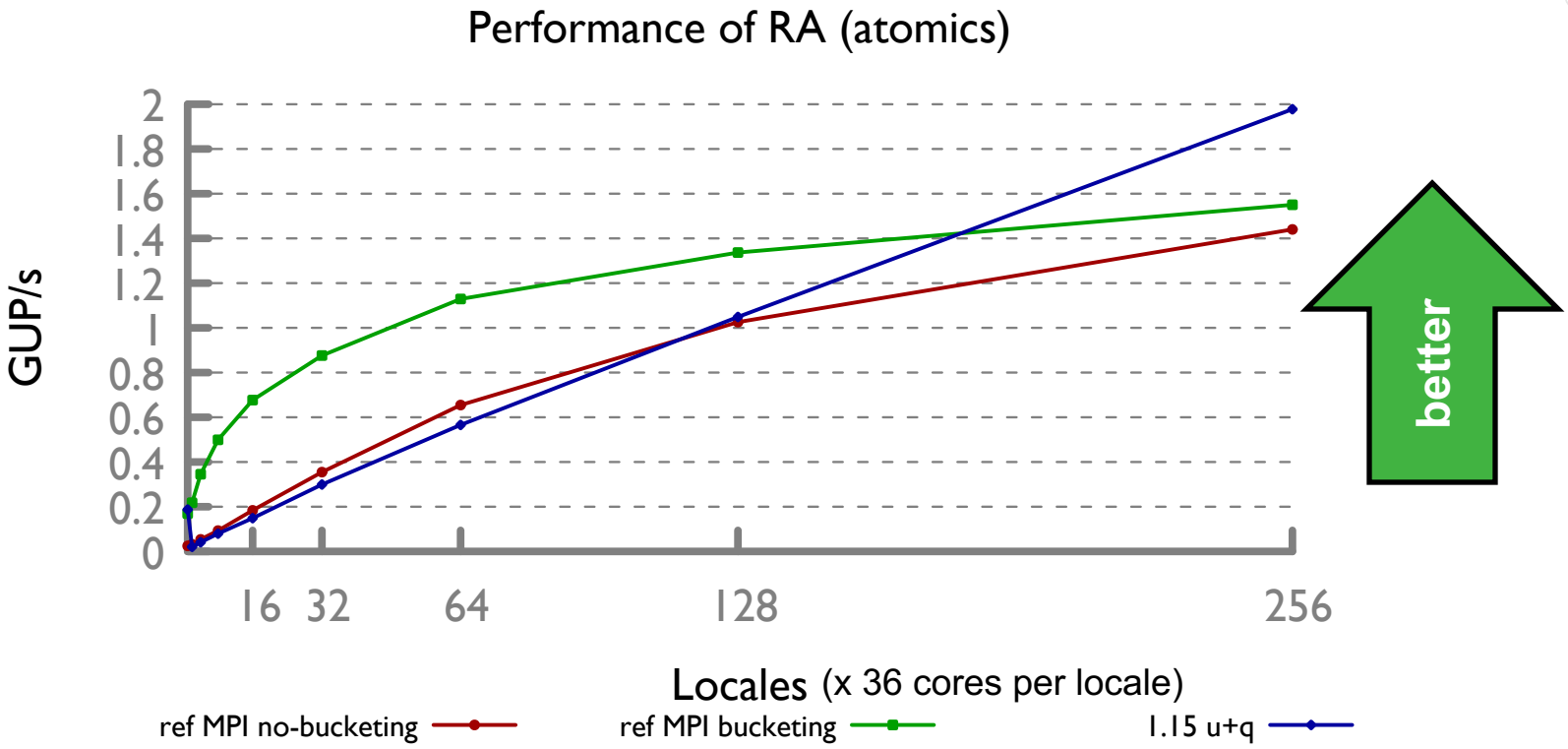
Copyright 2017 Cray Inc.

ISx Execution Time: MPI, SHMEM, Chapel



Copyright 2017 Cray Inc.

RA Performance: Chapel vs. MPI



Copyright 2017 Cray Inc.

[At? And? Ah'm?] fourteen, and you know
That I've learned the easy way
Some stupid decisions
And with a...a broken heart

Summary

That Summer, at Home I Had Become The Invisible Boy
The Twilight Sad
Fourteen Autumns and Fifteen Winters

Copyright 2017 Cray Inc.

This Talk's Thesis

Programming language designers have, to date, largely failed the large-scale parallel computing community.

- parallel features have historically been an afterthought
  - tacked on through libraries, pragmas, and extensions
  - rarely first-class
- even when languages are designed for parallelism…

  …most fail to consider scalable computing (distributed memory)

  …others tend to be domain-specific and not very general

We can do better!

Copyright 2017 Cray Inc.

We think Chapel is an attractive candidate

- **Attractive language for end-users**
  - modern design
  - separates algorithmic specification from mapping to architecture
- **Able to achieve competitive performance**
  - though a good deal of tuning and optimization work remain…

Copyright 2017 Cray Inc.

Recap of This Talk's Takeaways

If you design a parallel programming language…

…don't tie yourself to architecture-specific mechanisms

…don't base your model for parallelism and locality on SPMD

…create attractive ways of expressing parallelism and locality

…support first-class index sets

…support multiresolution features

Copyright 2017 Cray Inc.

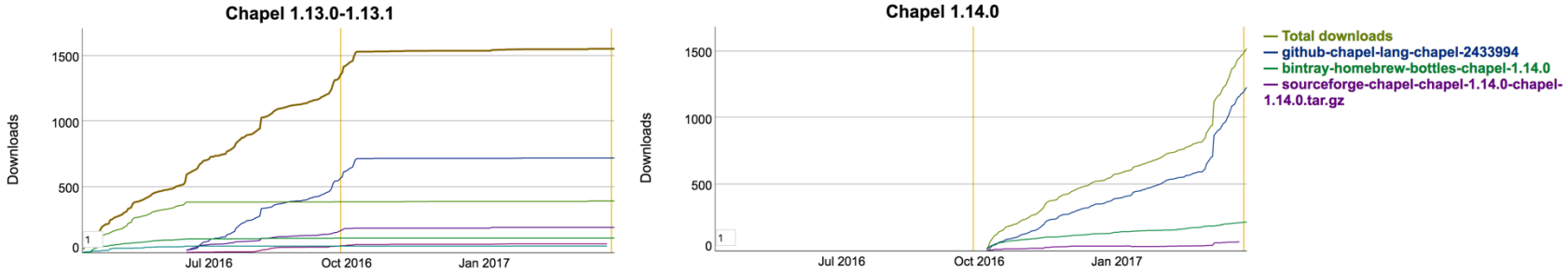
Chapel Challenges: Technical

- **design and optimization of multiresolution language features:**
  - domain maps
  - parallel iterators
  - locale models
- **compiler architecture: outgrowing the initial research prototype**
- **tools: classic chicken-and-egg problem**
  - IDE
  - debuggers
  - performance analysis
  - package manager
  - interactive development (interpreter, REPL, iPython, …)

Copyright 2017 Cray Inc.

Chapel Challenges: Social

- **building up momentum in the user community**
  - lots of interest, but also lots of fear of being first / only adopter
  - 3000+ downloads / year for 2 releases



- **combatting impatience / numbness to our message and progress**
- **developing an aggressive language in a conservative field (HPC)**
- **engaging with peers in mainstream computing**

Copyright 2017 Cray Inc.

Scratch my name on your arm with a fountain pen
This means you really love me

Further Resources

Rusholme Ruffians
The Smiths
Meat is Murder

Copyright 2017 Cray Inc.

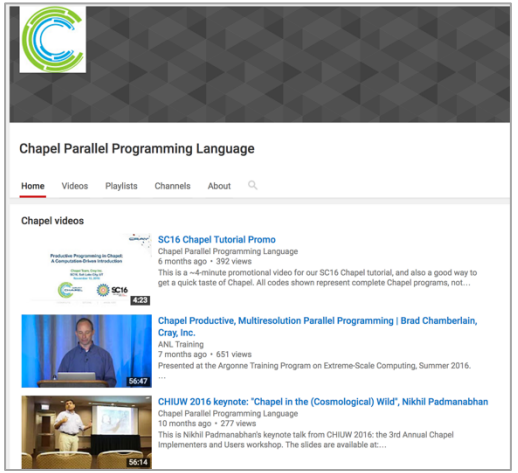
How to Stalk Chapel

http://facebook.com/ChapelLanguage

http://twitter.com/ChapelLanguage

https://www.youtube.com/channel/UCHmm27bYjhknK5mU7ZzPGsQ/

chapel-announce@lists.sourceforge.net



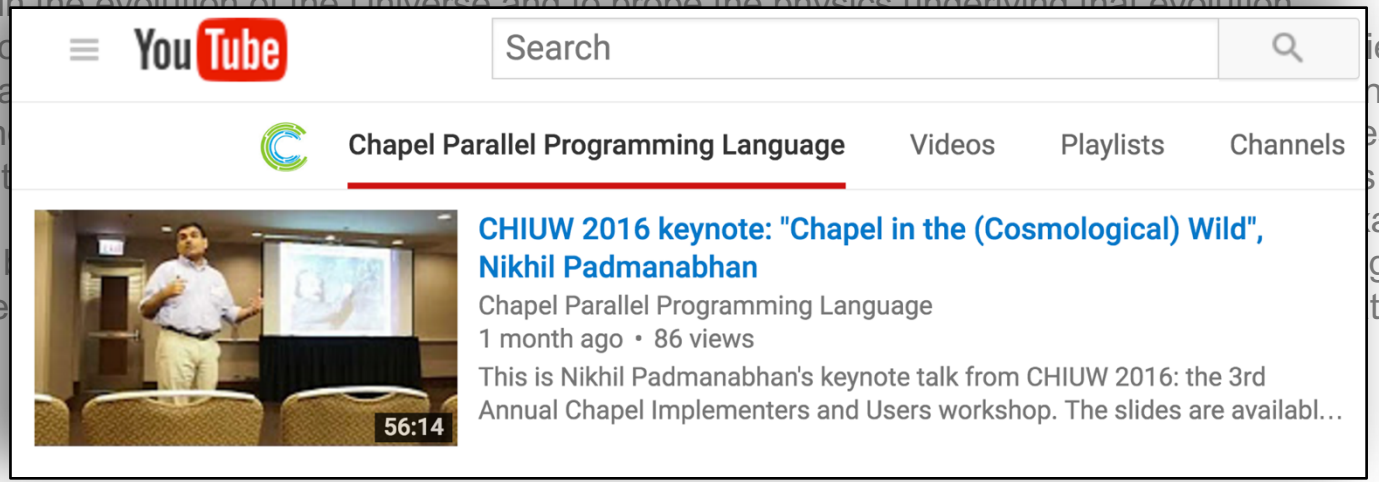
Copyright 2017 Cray Inc.

A Chapel talk to watch next (user perspective)

Chapel in the (Cosmological) Wild (CHIUW 2016 keynote)

Nikhil Padmanabhan, *Yale University Professor, Physics & Astronomy*

Abstract: This talk aims to present my personal experiences using Chapel in my research. My research interests are in observational cosmology; more specifically, I use large surveys of galaxies to constrain the evolution of the Universe and to probe the physics underlying that evolution. Operatic ies, both on actua ng a whirlwin es to solving t s quickly and the ample of Chapel l ghting its interope t it would take for

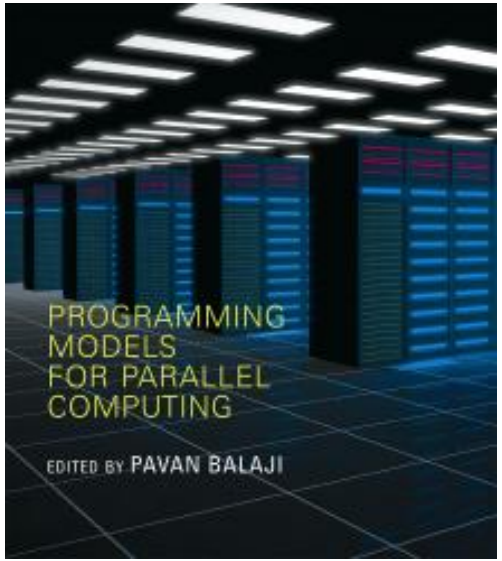


Copyright 2017 Cray Inc.

Suggested Reading (healthy attention spans)

Chapel chapter from ***Programming Models for Parallel Computing***

- a detailed overview of Chapel's history, motivating themes, features
- published by MIT Press, November 2015
- edited by Pavan Balaji (Argonne)
- chapter is now also available online



Other Chapel papers/publications available at **http://chapel.cray.com/papers.html**

Copyright 2017 Cray Inc.

Suggested Reading (short attention spans)

Chapel: Productive Parallel Programming, Cray Blog, May 2013.

- *a short-and-sweet introduction to Chapel*

Six Ways to Say "Hello" in Chapel **(parts 1, 2, 3)**, Cray Blog, Sep-Oct 2015.

- *a series of articles illustrating the basics of parallelism and locality in Chapel*

Why Chapel? **(parts 1, 2, 3)**, Cray Blog, Jun-Oct 2014.

- *a series of articles answering common questions about why we are pursuing Chapel in spite of the inherent challenges*

[Ten] Myths About Scalable Programming Languages, IEEE TCSC Blog (index available on chapel.cray.com "blog articles" page), Apr-Nov 2012.

- *a series of technical opinion pieces designed to argue against standard reasons given for not developing high-level parallel languages*

Copyright 2017 Cray Inc.

All we ever wanted was everything…

(I have Chapel swag to give you)

Get up
Eat jelly
sandwich bars, ….

(Have a good break and get some yummy snacks!)

ACCU 2017

All We Ever Wanted Was Everything
Bauhaus
The Sky's Gone Out

Copyright 2017 Cray Inc.

Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publically announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, and URIKA. The following are trademarks of Cray Inc.: ACE, APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, THREADSTORM. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT, and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used in this document are the property of their respective owners.

Copyright 2017 Cray Inc.

CRAY

THE SUPERCOMPUTER COMPANY

accu
2017