The Audacity of Chapel: Scalable Parallel Programming Done Right

Brad Chamberlain, Chapel Team, Cray Inc.
ACCU 2017, Bristol UK
April 27th, 2017
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.
This keynote contains many slides and no guitar solos 😞

BUT... it does contain thematically relevant

UK rock lyrics trivia!!
I am the egg man

Introductions

I am the Walrus
The Beatles
Magical Mystery Tour
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…

(“Pssst… Who is this turkey?”)
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
Recent Highlighted Cray Systems

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

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

Petroleum Geo-Services
Cray Market Segments

- Manufacturing
- Machine Learning & Deep Learning
- Life Sciences
- Higher Education
- Cybersecurity
- Earth Sciences
- Energy
- Financial Services
- Government and Defense
“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.*
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
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)
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
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.”
“The Audacity of Chapel”?

**audacity** (according to Google):

/ɔːˈdæsɪti/

**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 | … ]!”
This Talk’s Thesis

Programming language designers have, to date, largely failed the large-scale parallel computing community.
This just feels like spinning plates
I’m living in cloud-cuckoo land

The Status Quo in HPC Programming
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:**

\[
\begin{align*}
A &= B + \alpha \cdot C \\
    &= \alpha
\end{align*}
\]
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):
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):

- $A$
- $B$
- $C$
- $\alpha$
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):
Scalable Parallel Programming Concerns

Q: What should scalable parallel programmers focus on?
A: **Parallelism:** What should execute simultaneously?
**Locality:** Where should those tasks execute?
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;
}

// Example usage:
HPCC_Params params = { /* some parameters */ };
HPCC_StarStream(&params, 1);
`#include <hpcc.h>`
`#ifdef _OPENMP`  
`#include <omp.h>`  
`#endif`

```c
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;
}
```
STREAM Triad: MPI+OpenMP

```c
#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 );
        } else
            return 1;
    }
    #ifdef _OPENMP
    #pragma omp parallel for
    for (j=0; j<VectorSize; j++) {
        b[j] = 2.0;
        c[j] = 1.0;
    }
    scalar = 3.0;
    #ifdef _OPENMP
    #pragma omp parallel for
    for (j=0; j<VectorSize; j++)
        a[j] = b[j] + scalar * c[j];
    #endif
    HPCC_free(c);
    HPCC_free(b);
    HPCC_free(a);
    return 0;
    }
```

CUDA

```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];
}
```
STREAM Triad: MPI+OpenMP

```c
#include <hpcc.h>
#include <omp.h>

#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.x+=1;
    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 N) {
    int idx = threadIdx.x + blockIdx.x * blockDim.x;
    if (idx < N) 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.
STREAM Triad: Chapel

```
#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.x+=1;
    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];
}
```

```
#include <hpcc.h>
#include <omp.h>
static int VectorSize;
static double *a, *b, *c;
int HPCC_StarStream(HPCC_Params *params) {
    int myRank, commSize;
    MPI_Comm comm = MPI_COMM_WORLD;
    MPI_Comm_size(comm, &commSize);
    MPI_Comm_rank(comm, &myRank);
    int rv, errCount;
    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;
}
```

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.
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
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
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
SPMD by Example (in pictures)

“Apply a 3-Point Stencil to a vector”

Conceptual View

\[
\frac{(\text{black} + \text{red})}{2}
\]

SPMD View

```
“Apply a 3-Point Stencil to a vector”

**Conceptual View**

\[(\text{red} + \text{red} + \text{red})/2\] = \[\text{red} + \text{red} + \text{red}\]

**SPMD View**

\[(\text{red} + \text{red} + \text{red})/2\] = \[\text{red} + \text{red} + \text{red}\]

\[(\text{red} + \text{red} + \text{red})/2\] = \[\text{red} + \text{red} + \text{red}\]

\[(\text{red} + \text{red} + \text{red})/2\] = \[\text{red} + \text{red} + \text{red}\]
SPMD by Example (in code)

“Apply a 3-Point Stencil to a vector”

```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;
}
```
SPMD by Example (in code)

“Apply a 3-Point Stencil to a vector”

Global-view code (Chapel)

```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

```chapel
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[0]);
        recv(me-1, A[0]);
    }
    forall i in 1..myN do
        B[i] = (A[i-1] + A[i+1])/2;
}
```
“Apply a 3-Point Stencil to a vector”

Global-view code (Chapel)

```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;
}
```

Bug: Refers to uninitialized values at ends of A

SPMD pseudo-code

```plaintext
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[0]);
        recv(me-1, A[1]);
    }
    forall i in 1..myN do
        B[i] = (A[i-1] + A[i+1])/2;
}
```
Assumes p divides n

GL

Global-view code (Chapel)

```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;
}
```

Comm

Communication becomes geometrically more complex for higher-dimensional arrays

SPMD pseudo-code

```chapel
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;
}
```

Communication becomes geometrically more complex for higher-dimensional arrays

```
```
27-point stencils \((rprj3 \text{ from NAS MG})\)
**rprj3 in Fortran + MPI**

```fortran
subroutine rprj3(r,m1k,m2k,m3k,s,m1j,m2j,m3j,k)
imPLICIT none
inclusive 'cafnpb.h'
inclusive '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-1
         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
   endo
   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) =
   > 0.50D0 * 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

j = k-1

Call comm3(s,m1j,m2j,m3j,j)

return
endo
```

Copyright 2017 Cray Inc.
**rprj3 in Fortran + MPI**

```fortran
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-1
        i1 = 2*j1-d1
        x1(i1) = r(i1,i2,i3) + r(i1,i2+1,i3) + r(i1,i2, i3-1) + r(i1,i2, i3+1)
        y1(i1) = r(i1,i2-1,i3) + r(i1,i2-1,i3-1) + r(i1,i2-1,i3+1) + r(i1,i2+1,i3-1) + r(i1,i2+1,i3+1)
      enddo
    enddo
  enddo
  do j1=2,m1j-1
    i1 = 2*j1-d1
    y2 = r(i1, i2-1,i3-1) + r(i1, i2-1,i3) + r(i1, i2+1,i3-1) + r(i1, i2+1,i3)
    x2 = r(i1, i2-1,i3) + r(i1, i2,i3 ) + r(i1, i2+1,i3 )
    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
  call comm3(s,m1j,m2j,m3j,j)
  return
end
```

Copyright 2017 Cray Inc.
comm3: the communication for rprj3
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.
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 (see Chapel)
And now for something completely different.
[Instrumental]

Chapel Characteristics

The Liberty Bell March
John Philip Sousa
(unreleased?)
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]
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
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
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
The Chapel Team at Cray (May 2016)

14 full-time employees + 2 summer interns + occasional visiting academics
(one of each started after photo taken)
Chapel Community R&D Efforts

(and several others…)

http://chapel.cray.com/collaborations.html
You say you got a real solution
Well, you know
We’d all love to see the plan

Chapel in a Nutshell
Chapel language feature areas

Chapel language concepts

- Domain Maps
- Data Parallelism
- Task Parallelism
- Base Language
- Locality Control
- Target
Base Language

- Domain Maps
- Data Parallelism
- Task Parallelism
- Base Language
- Locality Control
- Target

Lower-level Chapel
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...
```
Base Language Features, by example

```plaintext
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);
```

- Modern iterators
-CLU-style iterators
-CLU-style iterators
-Modern iterators

```

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
...
```
Base Language Features, by example

```go
config const n = 10;
for f in fib(n) do writeln(f);
```

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

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

Static type inference for:
- arguments
- return types
- variables
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);
```

Zippered iteration

```
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
...```
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
data writeln("fib ", i, " is ", f);
```

<table>
<thead>
<tr>
<th>fib</th>
<th>is</th>
</tr>
</thead>
<tbody>
<tr>
<td>fib #0</td>
<td>0</td>
</tr>
<tr>
<td>fib #1</td>
<td>1</td>
</tr>
<tr>
<td>fib #2</td>
<td>1</td>
</tr>
<tr>
<td>fib #3</td>
<td>2</td>
</tr>
<tr>
<td>fib #4</td>
<td>3</td>
</tr>
<tr>
<td>fib #5</td>
<td>5</td>
</tr>
<tr>
<td>fib #6</td>
<td>8</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
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);
```

```
tuples
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
...
**Base Language Features, by example**

```plaintext
iter fib(n) {
    var current = 0,
        next = 1;
    for i in 1..n {
        yield current;
        current += next;
        current <=> next;
    }
}
```

```plaintext
config const n = 10;

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

```
<table>
<thead>
<tr>
<th>i</th>
<th>fib #i is</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
```
Task Parallelism and Locality Control
Task Parallelism and Locality, by example

```chpl
coforall loc in Locales do
    on loc {
        const numTasks = here.maxTaskPar;
        coforall tid in 1..numTasks do
            printf("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
```
Task Parallelism and Locality, by example

```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);
  }
```

Abstraction of System Resources

```
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
```
Task Parallelism and Locality, by example

```chapel
coforall loc in Locales do
  on loc {
    const numTasks = here.maxTaskPar;
    coforall tid in 1..numTasks do
      writeln("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
```
Task Parallelism and Locality, by example

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

Control of Locality/Affinity

```
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
```
Task Parallelism and Locality, by example

```
taskParallel.chpl

c forall 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
```
Task Parallelism and Locality, by example

```
coforall loc in Locales do
  on loc {
    const numTasks = here.maxTaskPar;
    coforall tid in 1..numTasks do
      printf("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
```
Task Parallelism and Locality, by example

```chpl
coforall loc in Locales do
  on loc {
    const numTasks = here.maxTaskPar;
    coforall tid in 1..numTasks do
      printf("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
```
Task Parallelism and Locality, by example

```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
```
Parallelism and Locality: Distinct in Chapel

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

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

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

  ```chapel
 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:

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

### Chapel language concepts

<table>
<thead>
<tr>
<th>Domain Maps</th>
<th>Data Parallelism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task Parallelism</td>
<td>Base Language</td>
</tr>
<tr>
<td>Locality Control</td>
<td>Target</td>
</tr>
</tbody>
</table>

Higher-level Chapel
Data Parallelism, by example

```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
```
Data Parallelism, by example

Domains (Index Sets)

```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

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>1.3</td>
<td>1.5</td>
<td>1.7</td>
<td>1.9</td>
</tr>
<tr>
<td>2.1</td>
<td>2.3</td>
<td>2.5</td>
<td>2.7</td>
<td>2.9</td>
</tr>
<tr>
<td>3.1</td>
<td>3.3</td>
<td>3.5</td>
<td>3.7</td>
<td>3.9</td>
</tr>
<tr>
<td>4.1</td>
<td>4.3</td>
<td>4.5</td>
<td>4.7</td>
<td>4.9</td>
</tr>
<tr>
<td>5.1</td>
<td>5.3</td>
<td>5.5</td>
<td>5.7</td>
<td>5.9</td>
</tr>
</tbody>
</table>
Data Parallelism, by example

```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
```
Data Parallelism, by example

```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
```
Distributed Data Parallelism, by example

```
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
```
Distributed Data Parallelism, by example

```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
```
Domain Maps

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

\[ A = B + \alpha \times C; \]

...to the target locales’ memory and processors:
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
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
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
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
Puttin’ me down for thinking of someone new
Always the same
Playin’ your game
Drive me insane…

Computer Language Benchmarks Game Results
The Computer Language Benchmarks Game

64-bit quad core data set
Will your toy benchmark program be faster if you write it in a different programming language? It depends how you write it!

Which programs are fast?
Which are succinct? Which are efficient?

<table>
<thead>
<tr>
<th>Ada</th>
<th>C</th>
<th>Chapel</th>
<th>Clojure</th>
<th>C#</th>
<th>C++</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dart</td>
<td>Erlang</td>
<td>F#</td>
<td>Fortran</td>
<td>Go</td>
<td>Hack</td>
</tr>
<tr>
<td>Haskell</td>
<td>Java</td>
<td>JavaScript</td>
<td>Lisp</td>
<td>Lua</td>
<td></td>
</tr>
<tr>
<td>OCaml</td>
<td>Pascal</td>
<td>Perl</td>
<td>PHP</td>
<td>Python</td>
<td></td>
</tr>
<tr>
<td>Racket</td>
<td>Ruby</td>
<td>JRuby</td>
<td>Rust</td>
<td>Scala</td>
<td></td>
</tr>
<tr>
<td>Smalltalk</td>
<td>Swift</td>
<td>TypeScript</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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…
Chapel’s approach to the CLBG:

- striving for elegance over heroism
- ideally: “Want to learn how program xyz works? Read the Chapel version.”
How many times slower?

benchmarks game 20 Apr 2017 u64q
Can sort results by execution time, code size, memory or CPU use:

**chameleon-redux**

description

<table>
<thead>
<tr>
<th>program</th>
<th>source code, command-line and measurements</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x</td>
</tr>
<tr>
<td>1.0 C</td>
<td>60</td>
</tr>
<tr>
<td>1.2 C++ g++ #5</td>
<td>0.70</td>
</tr>
<tr>
<td>1.7 Lisp SBCL #3</td>
<td>1.01</td>
</tr>
<tr>
<td>2.3 Chapel #2</td>
<td>1.39</td>
</tr>
<tr>
<td>3.3 Rust #2</td>
<td>2.01</td>
</tr>
<tr>
<td>5.6 C++ g++ #2</td>
<td>3.40</td>
</tr>
<tr>
<td>6.8 Chapel</td>
<td>4.09</td>
</tr>
<tr>
<td>8.0 Java #4</td>
<td>4.82</td>
</tr>
<tr>
<td>8.5 Haskell GHC</td>
<td>5.15</td>
</tr>
<tr>
<td>10 Java</td>
<td>6.13</td>
</tr>
<tr>
<td>10 Haskell GHC #4</td>
<td>6.34</td>
</tr>
<tr>
<td>11 C# .NET Core</td>
<td>6.59</td>
</tr>
<tr>
<td>11 Go</td>
<td>6.90</td>
</tr>
<tr>
<td>13 Go #2</td>
<td>7.59</td>
</tr>
<tr>
<td>13 Java #3</td>
<td>7.94</td>
</tr>
</tbody>
</table>

**gz == code size metric**

strip comments and extra whitespace, then gzip
CLBG: Chapel entries
CLBG: Chapel vs. 9 key languages

- C
- C++
- Fortran
- Go
- Rust
- Swift
- Java
- Scala
- Python
Can also browse program source code (but this requires actual thought):

```c
#include <stdio.h>

int main() {
    printf("Hello, World!\n");
    return 0;
}
```

**CLBG: Qualitative Comparisons**
Can also browse program source code (but this requires actual thought):

excerpt from 1210 gz Chapel entry

```chapel
proc main()
{
    printColorEquations();
    cobegin{
        holdMeetings(group1, n);
        holdMeetings(group2, n);
    }
    p=$(guru)
    print(guru); // Print the results of getNewColor() for all colors
    proc printColorEquations()
    {
        for cl in Color do
        {writeln(cl, " + ", c2, " " );
        }
    }
    proc holdMeetings(population, numMeetings)
    {
        const place = new MeetingPlace(numMeetings);
        for c in population do
        {c.haveMeetings(place, population);
        }
        delete place;
    }
}
```

excerpt from 2863 gz C gcc entry

```c
void set_affinity(int* mnp)
{
    cpu_set_t* affinity1, cpu_set_t* affinity2;
    active_cpus;
    buf[2048];
    pos;
    cpu_idx;
    physical_id;
    core_id;
    cpu_count;
    i;
    processor_str = "processor";
    processor_str_len = strlen(processor_str);
    physical_id = "physical_id";
    physical_id_str_len = strlen(physical_id);
    core_id_str = "core_id";
    core_id_str_len = strlen(core_id);
    // create
    for (c = core; c <= cores; c++)
    {
        printf("%s\n", cpu_cores_str);
        printf("%s\n", cpu_core);
        process_create();
        // create
```
Can also browse program source code (but this requires actual thought):

excerpt from 1210 gz Chapel entry

```c
char const* core_id_str = "core id";
size_t core_id_str_len = strlen(core_id_str);
char const* cpu_cores_str = "cpu core";
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;
}
```

excerpt from 2863 gz C gcc entry

```c
void get_affinity(int* is_smp, cpu_set_t* affinity, cpu_set_t* affinity2) {
    cpu_set_t active_cpus;
    FILE* f;
    char buf [2048];
    char const* core_id;
    int cpu_idx;
    int physical_id;
    int core_id;
    int cpu_cores;
    int apic_id;
    size_t size_t;
    cpu_count;
    size_t
    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_str = "core";
    size_t core_str_len = strlen(core_str);
    char const* cpu_cores_str = "cpu core";
    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;
}
```
I have run
I have crawled
I have scaled...

Chapel Scalability Results
ISx Execution Time: MPI, SHMEM

![Graph showing ISx weakISO Total Time comparison between MPI and SHMEM across different nodes]

- **Time (seconds)**
  - 0
  - 2
  - 4
  - 6
  - 8
  - 10
  - 12
  - 14
- **Nodes (x 36 cores per node)**
  - 1
  - 2
  - 4
  - 8
  - 16
  - 32
  - 64

**Better**

**ISx weakISO Total Time**

**MPI**

**SHMEM**
ISx Execution Time: MPI, SHMEM, Chapel

ISx weakISO Total Time

Time (seconds)

Nodes (x 36 cores per node)

Chapel 1.15
MPI
SHMEM

better
RA Performance: Chapel vs. MPI

Performance of RA (atomics)

GUP/s

Locales (x 36 cores per locale)

ref MPI no-bucketing
ref MPI bucketing
1.15 u+q

better
[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
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!*
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…
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
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, …)
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
Scratch my name on your arm with a fountain pen
This means you really love me

Further Resources
How to Stalk Chapel

http://facebook.com/ChapelLanguage
http://twitter.com/ChapelLanguage
https://www.youtube.com/channel/UCHmm27bYjhknK5mU7ZzPGsQ/
chapel-announce@lists.sourceforge.net
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. Operationally, this involves measuring a number of spatial statistics of the distribution of galaxies, both on actual observations and on large numbers of simulated universes to solving the problems that keep me up at night and our approaches to solving these. The key motivation for using Chapel was the ability to prototype algorithms quickly and the promise of ease and flexibility when writing parallel programs. An example of Chapel's interoperability is demonstrated by highlighting its ability to interface with existing libraries. I'll conclude with what it would take for me to switch over to using Chapel all of the time.
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](http://chapel.cray.com/papers.html)
Suggested Reading (short attention spans)

- 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
All we ever wanted was everything…
(I have Chapel swag to give you)

Get up
Eat jelly,
sandwiches, …. 

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

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