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
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…
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

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

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:**

```
A   =   
B +   
C \cdot 
\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 (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
α
    . . . .
```
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):

\[
\begin{align*}
A & \quad = \quad = \quad = \quad = \quad = \quad = \quad = \\
B & \quad + \quad + \quad + \quad + \quad + \quad + \quad + \\
C & \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \\
\alpha & \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot 
\end{align*}
\]
Scalable Parallel Programming Concerns

Q: What should scalable parallel programmers focus on?
A: **Parallelism**: What should execute simultaneously?
**Locality**: Where should those tasks execute?
#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).
            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;
}
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;
}
#include <hpcc.h>
#ifdef _OPENMP
#include <omp.h>
#endif

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);
        float scalar;
        printf( outFile, "Failed to allocate memory (%d).\n", VectorSize);
        free(outFile);
        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];
    HPCC_free(c);
    HPCC_free(b);
    HPCC_free(a);
    return 0;
    }
    #endif
    #endif
    return 0;
}

__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];
}
```c
#include <hpcc.h>
#ifdef _OPENMP
#include <omp.h>
#endif

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

int HPCC_StartStream(HPCC_Params *params) {
    int myRank, commSize;

    MPI_Comm comm = MPI_COMM_WORLD;
    MPI_Comm_size(comm, &commSize);
    MPI_Comm_rank(comm, &myRank);

    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;
}

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);

    if( N % dimBlock.x != 0 )
        dimGrid.x+=1;

    __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];
    }
```
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.

<table>
<thead>
<tr>
<th>Type of HW Parallelism</th>
<th>Programming Model</th>
<th>Unit of Parallelism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inter-node</td>
<td>MPI</td>
<td>executable</td>
</tr>
<tr>
<td>Intra-node/multicore</td>
<td>OpenMP / pthreads</td>
<td>iteration/task</td>
</tr>
<tr>
<td>Instruction-level vectors/threads</td>
<td>pragmas</td>
<td>iteration</td>
</tr>
<tr>
<td>GPU/accelerator</td>
<td>CUDA / Open[MP</td>
<td>CL</td>
</tr>
</tbody>
</table>

**benefits:** lots of control; decent generality; easy to implement

**downsides:** lots of user-managed detail; brittle to changes
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.
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
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

\[
\frac{(\text{red} + \text{green} \, \text{blue})}{2}
\]

SPMD View
SPMD by Example (in pictures)

“Apply a 3-Point Stencil to a vector”

Conceptual View

\[
\frac{+}{2}
\]

SPMD View

\[
\frac{+}{2} = \frac{+}{2} = \frac{+}{2}
\]
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**

```java
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;
}
```
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;
}
```

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

\[
\begin{align*}
= & \quad \text{w}_1 \\
= & \quad \text{w}_2 \\
= & \quad \text{w}_3 \\
= & \quad \text{w}_4
\end{align*}
\]
principal 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
         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
   enddo
enddo

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

x2 = r(i1, i2-1,i3  ) + r(i1, i2-1,i3+1)
>              + r(i1, i2+1,i3-1) + r(i1, i2+1,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
```

end
enddo

j = k-1

please call comm3(s,m1j,m2j,m3j,j)

return

end
**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
         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
   enddo
   do j1=2,m1j
      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
enddo

d = k-1

cALL comm3(s,m1j,m2j,m3j,j)
return
end
```

Copyright 2017 Cray Inc.
BUFF_LEN

DOUBLE PRECISION U(N1, N2, N3)

INTEGER N1, N2, N3,

INCLUDE 'CAF_INTRINSCES'

USE SUBROUTINE COMM3(U,N1,N2,N3,KK)

COMM3:

DO  AXIS = 1, 3

ENDDO

CAF_INTRINSCES

BUFF(1:BUFF_LEN,BUFF_ID+1)

DO I3=2,N3

ELSE IF( AXIS EQ. 3 ) THEN

BUFF_ID = U(I1, I2, I3)

ENDIF

ENDIF

ENDIF

RETURN

ENDSUBROUTINE COMM3

SUBROUTINE COMM3(A, I1, I2, I3)

USE CAF_INTRINSCES

INTEGER I1, I2, I3,

DOUBLE PRECISION U(I1, I2, I3)

INTEGER N1, N2, N3,

INCLUDE 'CAF_INTRINSCES'

USE SUBROUTINE COMM3(U,N1,N2,N3,KK)

COMM3:

DO  AXIS = 1, 3

ENDDO

CAF_INTRINSCES

BUFF(1:BUFF_LEN,BUFF_ID+1)(A)

ENDIF

RETURN

ENDSUBROUTINE COMM3

COMM3: THE COMMUNICATION FOR RPRJ3

BUFF_LEN

DOUBLE PRECISION U(N1, N2, N3)

INTEGER N1, N2, N3,

INCLUDE 'CAF_INTRINSCES'

INTEGER N1, N2, N3,

DOUBLE PRECISION U(I1, I2, I3)

INTEGER N1, N2, N3,

INCLUDE 'CAF_INTRINSCES'

LICENSE

BUFF(1:BUFF_LEN,BUFF_ID)

ELSE IF( DIR EQ. 1 ) THEN

DO 124, N3

1

BUFF_ID <= U(I1, I2, I3) - 12, 13)

ENDIF

ENDIF

ENDIF

RETURN

ENDSUBROUTINE COMM3

SUBROUTINE COMM3(A, I1, I2, I3)

USE CAF_INTRINSCES

INTEGER I1, I2, I3,

DOUBLE PRECISION U(I1, I2, I3)

INTEGER N1, N2, N3,

INCLUDE 'CAF_INTRINSCES'

USE SUBROUTINE COMM3(U,N1,N2,N3,KK)

COMM3:

DO  AXIS = 1, 3

ENDDO

CAF_INTRINSCES

BUFF(1:BUFF_LEN,BUFF_ID+1)(A)

ENDIF

RETURN

ENDSUBROUTINE COMM3

COMM3: THE COMMUNICATION FOR RPRJ3

BUFF_LEN

DOUBLE PRECISION U(N1, N2, N3)

INTEGER N1, N2, N3,

INCLUDE 'CAF_INTRINSCES'

INTEGER N1, N2, N3,

DOUBLE PRECISION U(I1, I2, I3)

INTEGER N1, N2, N3,

INCLUDE 'CAF_INTRINSCES'

LICENSE

BUFF(1:BUFF_LEN,BUFF_ID)

ELSE IF( DIR EQ. 1 ) THEN

DO 124, N3

1

BUFF_ID <= U(I1, I2, I3) - 12, 13)

ENDIF

ENDIF

ENDIF

RETURN

ENDSUBROUTINE COMM3

SUBROUTINE COMM3(A, I1, I2, I3)

USE CAF_INTRINSCES

INTEGER I1, I2, I3,

DOUBLE PRECISION U(I1, I2, I3)

INTEGER N1, N2, N3,
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
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;
NAS MG Speedup: Cray T3E

![Graph showing speedup over best 16-processor time with linear speedup, A-ZPL, ZPL, and F+MPI.](image)
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
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)
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
And now for something completely different.
[Instrumental]

Chapel Characteristics
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]
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
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
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

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

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

```javascript
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

| 0 | 1 | 1 | 2 | 3 | 5 | 8 | ... |
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

```pascal
config const n = 10;

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

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

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

Static type inference for:
- arguments
- return types
- variables
Iterative Fibonacci function:

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

Configuration:

```haskell
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
- ...

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
...
**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);
```

**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

```
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
...
```
Task Parallelism and Locality Control

- Domain Maps
- Data Parallelism
- Task Parallelism
- Base Language
- Locality Control
- Target
Task Parallelism and Locality, by example

c forall 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

```
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

High-Level Task Parallelism

```
taskParallel.chpl

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
```
Control of Locality/Affinity

```chpl
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

Abstraction of System Resources

```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

Task Parallelism

```chpl
coforall loc in Locales do
  on loc {
    const numTasks = here.maxTaskPar;
    coforall tid in 1..numTasks do
      printff("Hello from task %d of %d 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

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

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

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

    i = 0 2 4 6 8
    j =    

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.

    i = 0 2 4 6 8
    j = 2 4 6 8 0
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”)
Chapel: Scoping and Locality

```chapel
var i: int;
```

**Locales** (think: “compute nodes”)
Chapel: Scoping and Locality

\[
\text{var } i : \text{ int; }
\]
\[
\text{on Locales[1] } \\
\]

Locales (think: “compute nodes”)
var i: int;
on Locales[1] {
  var j: int;
}
var i: int;
on Locales[1] {
    var j: int;
    coforall loc in Locales {
        on loc {
        }
    }
}

Locales (think: “compute nodes”)
Chapel: Scoping and Locality

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

Locales (think: “compute nodes”)
Chapel: Scoping and Locality

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

OK to access `i`, `j`, and `k` wherever they live

Locales (think: “compute nodes”)
Chapel: Scoping and Locality

```chapel
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”)
Chapel: Locality queries

```chapel
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”)
Higher-Level Features

Chapel language concepts

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

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

```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
```
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

```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
```
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
```
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
LULESH in Chapel

<table>
<thead>
<tr>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
<th>Column 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data 1</td>
<td>Data 2</td>
<td>Data 3</td>
<td>Data 4</td>
</tr>
<tr>
<td>Data 5</td>
<td>Data 6</td>
<td>Data 7</td>
<td>Data 8</td>
</tr>
<tr>
<td>Data 9</td>
<td>Data 10</td>
<td>Data 11</td>
<td>Data 12</td>
</tr>
<tr>
<td>Data 13</td>
<td>Data 14</td>
<td>Data 15</td>
<td>Data 16</td>
</tr>
<tr>
<td>Data 17</td>
<td>Data 18</td>
<td>Data 19</td>
<td>Data 20</td>
</tr>
<tr>
<td>Data 21</td>
<td>Data 22</td>
<td>Data 23</td>
<td>Data 24</td>
</tr>
<tr>
<td>Data 25</td>
<td>Data 26</td>
<td>Data 27</td>
<td>Data 28</td>
</tr>
<tr>
<td>Data 29</td>
<td>Data 30</td>
<td>Data 31</td>
<td>Data 32</td>
</tr>
<tr>
<td>Data 33</td>
<td>Data 34</td>
<td>Data 35</td>
<td>Data 36</td>
</tr>
<tr>
<td>Data 37</td>
<td>Data 38</td>
<td>Data 39</td>
<td>Data 40</td>
</tr>
</tbody>
</table>

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
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
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
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

Your Time is Gonna Come
Led Zeppelin
Led Zeppelin
The Computer Language Benchmarks Game (CLBG)

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></td>
<td>Erlang</td>
<td>F#</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.”

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>

The Computer Language Benchmarks Game

Copyright 2017 Cray Inc.
How many times slower?

benchmarks game

20 Apr 2017 u64q
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

<table>
<thead>
<tr>
<th>x</th>
<th>source</th>
<th>secs</th>
<th>mem</th>
<th>gz</th>
<th>cpu</th>
<th>cpu load</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>C gcc #5</td>
<td>0.60</td>
<td>820</td>
<td>2863</td>
<td>2.37</td>
<td>100% 100% 98% 100%</td>
</tr>
<tr>
<td>1.2</td>
<td>C++ g++ #5</td>
<td>0.70</td>
<td>3,356</td>
<td>1994</td>
<td>2.65</td>
<td>100% 100% 91% 92%</td>
</tr>
<tr>
<td>1.7</td>
<td>Lisp SBCL #3</td>
<td>1.01</td>
<td>55,604</td>
<td>2907</td>
<td>3.93</td>
<td>97% 96% 99% 99%</td>
</tr>
<tr>
<td>2.3</td>
<td>Chapel #2</td>
<td>1.39</td>
<td>76,564</td>
<td>1210</td>
<td>5.43</td>
<td>99% 99% 99% 99%</td>
</tr>
<tr>
<td>3.3</td>
<td>Rust #2</td>
<td>2.01</td>
<td>56,936</td>
<td>2882</td>
<td>7.81</td>
<td>97% 98% 98% 98%</td>
</tr>
<tr>
<td>5.6</td>
<td>C++ g++ #2</td>
<td>3.40</td>
<td>1,880</td>
<td>2016</td>
<td>11.88</td>
<td>100% 51% 100% 100%</td>
</tr>
<tr>
<td>6.8</td>
<td>Chapel</td>
<td>4.09</td>
<td>66,584</td>
<td>1199</td>
<td>16.25</td>
<td>100% 100% 100% 100%</td>
</tr>
<tr>
<td>8.0</td>
<td>Java #4</td>
<td>4.82</td>
<td>37,132</td>
<td>1607</td>
<td>16.73</td>
<td>98% 99% 54% 99%</td>
</tr>
<tr>
<td>8.5</td>
<td>Haskell GHC</td>
<td>5.15</td>
<td>8,596</td>
<td>989</td>
<td>9.26</td>
<td>79% 100% 2% 2%</td>
</tr>
<tr>
<td>10</td>
<td>Java</td>
<td>6.13</td>
<td>53,760</td>
<td>1770</td>
<td>8.78</td>
<td>42% 45% 41% 16%</td>
</tr>
<tr>
<td>10</td>
<td>Haskell GHC #4</td>
<td>6.34</td>
<td>6,908</td>
<td>989</td>
<td>12.67</td>
<td>99% 100% 2% 1%</td>
</tr>
<tr>
<td>11</td>
<td>C# .NET Core</td>
<td>6.59</td>
<td>86,076</td>
<td>1407</td>
<td>22.96</td>
<td>99% 82% 78% 91%</td>
</tr>
<tr>
<td>11</td>
<td>Go</td>
<td>6.90</td>
<td>832</td>
<td>1167</td>
<td>24.19</td>
<td>100% 96% 56% 100%</td>
</tr>
<tr>
<td>13</td>
<td>Go #2</td>
<td>7.59</td>
<td>1,384</td>
<td>1408</td>
<td>27.65</td>
<td>91% 99% 99% 78%</td>
</tr>
<tr>
<td>13</td>
<td>Java #3</td>
<td>7.94</td>
<td>53,232</td>
<td>1267</td>
<td>26.86</td>
<td>54% 96% 98% 94%</td>
</tr>
</tbody>
</table>

**gz** == code size metric

strip comments and extra whitespace, then gzip
CLBG: Chapel vs. 9 key languages
Chapel vs. C++
Chapel vs. C++ (zoomed out)
Chapel vs. Python
Chapel vs. Python (zoomed out)
Can also browse program source code (but this requires actual thought):

excerpts from:
- 1210 gz Chapel entry
- 2863 gz C gcc entry
CLBG: Qualitative Comparisons

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

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

const group1 = [i in 1..5] new Chameneos[i,];
const group2 = [i in 1..5] new Chameneos[i,];

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

proc holdMeetings(population, numMeetings)
{
    const place = new MeetingPlace(numMeetings);
    for c in population do
        c.haveMeetings(place, population);
    delete place;
}

proc holdMeetings(population, numMeetings)
{
    const place = new MeetingPlace(numMeetings);
    for c in population do
        c.haveMeetings(place, population);
    delete place;
}
```

```
proc holdMeetings(population, numMeetings)
{
    const place = new MeetingPlace(numMeetings);
    for c in population do
        c.haveMeetings(place, population);
    delete place;
}
```

excerpt from 1210 gz Chapel entry

excerpt from 2863 gz C gcc entry
Can also browse program source code (but this requires actual thought):

```cpp
#include <stdio.h>

int main() {
    printf("Hello, world!");
    return 0;
}
```

---

**CLBG: Qualitative Comparisons**

excerpt from 1210 gz Chapel entry

```chapel
proc main() {
    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 2863 gz C gcc entry

```c
#include <stdio.h>

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 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_str"
    size_t core_str_len = strlen(core_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));
```
I have run
I have crawled
I have scaled...

Chapel Scalability Results
ISx Execution Time: MPI, SHMEM

ISx weakISO Total Time

Time (seconds)

Nodes (x 36 cores per node)

Time (seconds)

better

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
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…
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
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
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, but also on large numbers of simulated universes to solving the problems that keep me up at night and our approaches to these. I'll start by presenting a whirlwind introduction to cosmology and the problems that keep me up at night and then discuss what attracted me to Chapel—the ability to quickly prototype algorithms and the promised ease and flexibility of writing parallel programs. I'll then present a worked example of Chapel being used in a real-world application, discussing some of these aspects as well as highlighting its interoperability with existing libraries, as well as some of the challenges. 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](http://chapel.cray.com/papers.html)

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

(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
sandwich bars, ....

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