



The Audacity of Chapel: Scalable Parallel Programming Done Right

Brad Chamberlain, Chapel Team, Cray Inc.

ACCU 2017, Bristol UK

April 27th, 2017



Safe Harbor Statement



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



Fair Warning Statement



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



COMPUTE

|

STORE

|

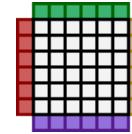
ANALYZE

("Pssst... Who is this turkey?")



Hi, I'm Brad Chamberlain

- graduate of the University of Washington
 - worked on a data-parallel array language, ZPL
- principal engineer at Cray Inc.
 - founding member and technical lead of the Chapel project
- more of a practical parallel computing guy than a PL expert...



COMPUTE

STORE

ANALYZE

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



COMPUTE

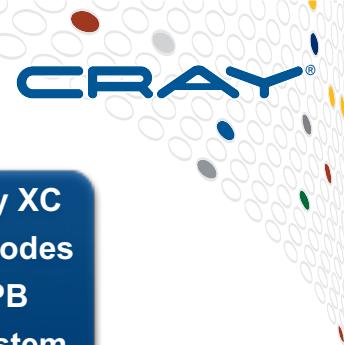
|

STORE

|

ANALYZE

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

COMPUTE

STORE

ANALYZE



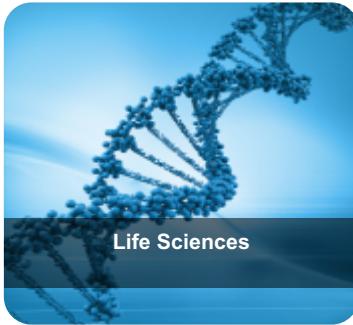
Cray Market Segments



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.

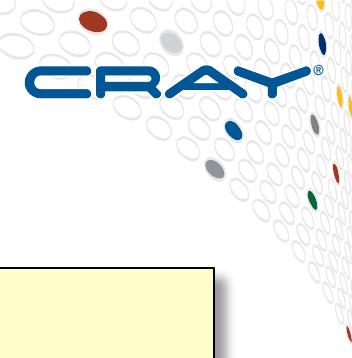


*If you didn't care what happened to me,
And I didn't care for you,
We would zig zag our way
Through the boredom and pain...*

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):

/əd'asɪti/

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.



COMPUTE

|

STORE

|

ANALYZE



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

The Status Quo in HPC Programming

Like Spinning Plates
Radiohead
Amnesiac



COMPUTE

|

STORE

|

ANALYZE

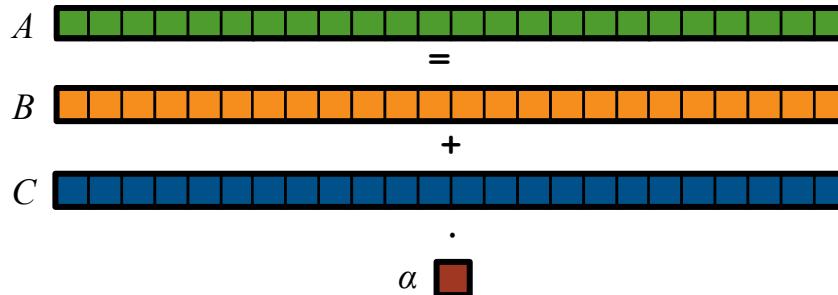
STREAM Triad: a trivial parallel computation



Given: m -element vectors A, B, C

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

In pictures:



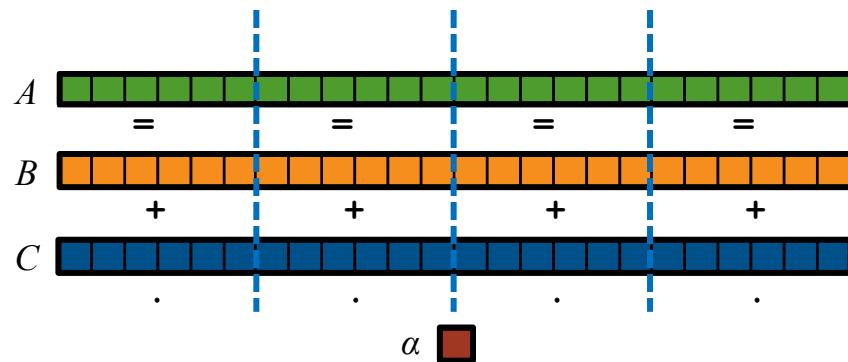
STREAM Triad: a trivial parallel computation



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):



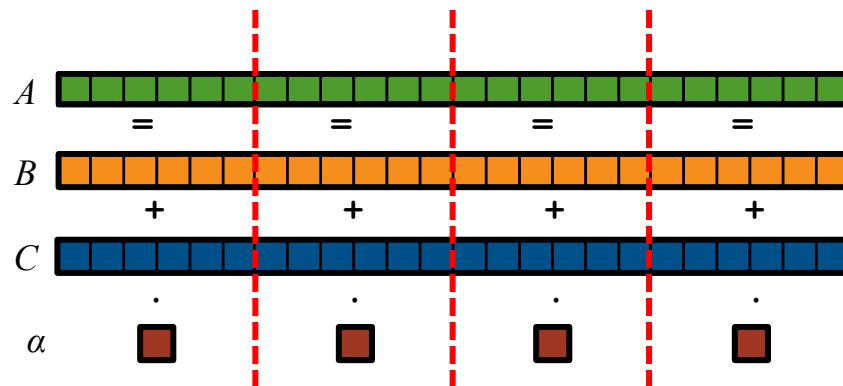
STREAM Triad: a trivial parallel computation



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):



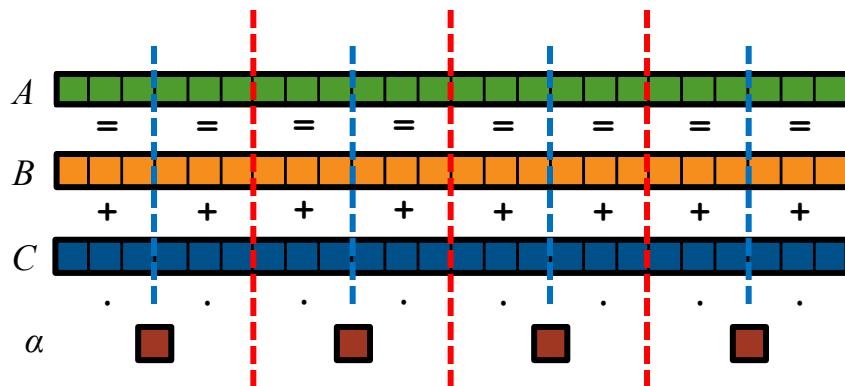
STREAM Triad: a trivial parallel computation



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):



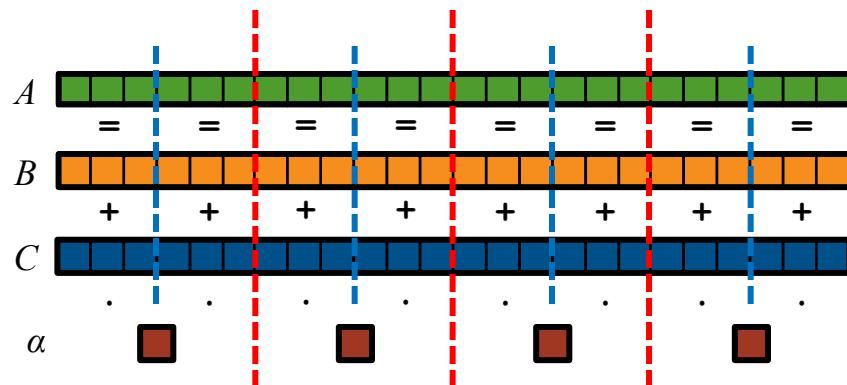
Scalable Parallel Programming Concerns



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>
MPI
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; }
```



COMPUTE

STORE

ANALYZE

STREAM Triad: MPI+OpenMP



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

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

MPI + OpenMP

```
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;
}

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

#ifndef _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; }
```



COMPUTE

STORE

ANALYZE

STREAM Triad: MPI+OpenMP

```
#include <hpcc.h>
#ifndef _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 );

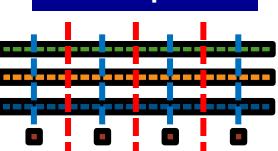
    return errCount;
}

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 );
```

MPI + OpenMP



```
if (!a || !b) #defined N 2000000
    if (c) HPC_free(c);
    if (b) HPC_free(b);
    if (a) HPC_free(a);
    if (doIO) {
        float scalar;
```

```
        fprintf( outFile, "Failed to
allocate cudaMemcpy(void**&d_a, sizeof(float)*N);
cudaMemcpy(void**&d_b, sizeof(float)*N);
VectorSize cudaMemcpy(void**&d_c, sizeof(float)*N);
fclose(outFile);

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

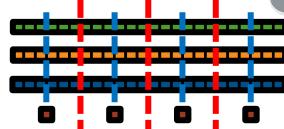
    return 1;
}

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

#endif _OPENMP
#pragma omp parallel for
#endif
_global_ void set_array(float *a, float value, int len) {
    for (j=0; j<VectorSize; j++) {
        if (idx < len) a[idx] = value;
    }
}

HPC_free(c);
HPC_free(b);
HPC_free(a);
_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]; }
```

CUDA



COMPUTE

STORE

ANALYZE

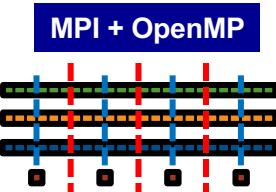
STREAM Triad: MPI+OpenMP



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

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

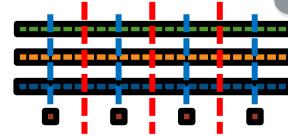
int HPCC_StarStream(HPCC_Params *params) {
    int myRank, commSize;
```



```
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 );
    }
}
```

```
#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);
```



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

```
    HPCC_SrcComm( 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 );
    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<<<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]; }
```



COMPUTE

STORE

ANALYZE

STREAM Triad: Chapel



```
#include <hpcc.h>
#ifndef _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,
        0, comm );
    return errCount;
}

int HPCC_Stream(HPCC_Params *params)
    register int j;
    double scalar;
    VectorSize = HPCC(sizeof(double),
        a = HPCC_XMALLOC(sizeof(double)*VectorSize);
        b = HPCC_XMALLOC(sizeof(double)*VectorSize);
        c = HPCC_XMALLOC(sizeof(double)*VectorSize);
```

```
use ...;

config const m = 1000,
            alpha = 3.0;

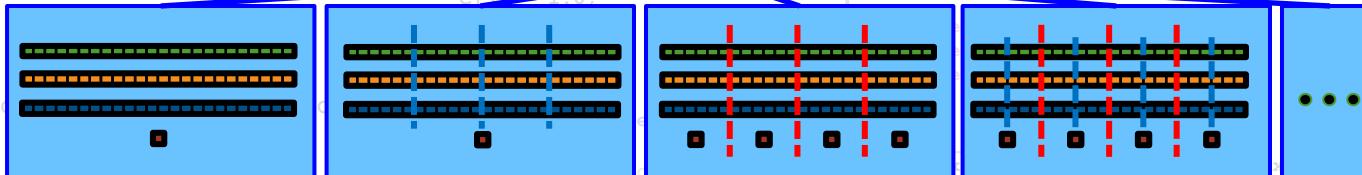
const ProblemSpace = {1..m} dmapped ...;

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

B = 2.0;
C = 1.0;

A = B + alpha * C;
```

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



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

COMPUTE

STORE

ANALYZE

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

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



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



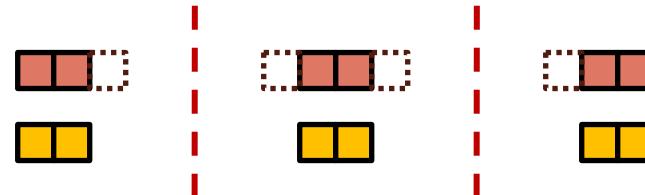
SPMD by Example (in pictures)

“Apply a 3-Point Stencil to a vector”

Conceptual View

$$\begin{array}{c} (\text{--- brown --- pink --- pink ---}) \\ + \text{--- pink --- brown --- brown --- })/2 \\ = \text{--- yellow --- orange --- orange --- yellow ---} \end{array}$$

SPMD View



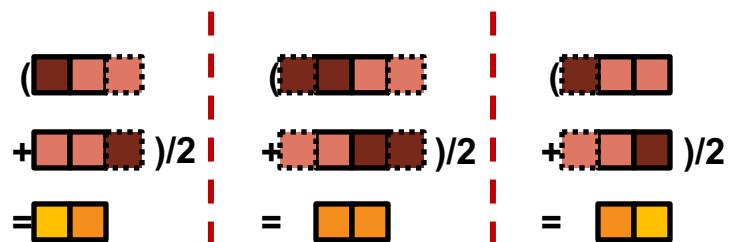
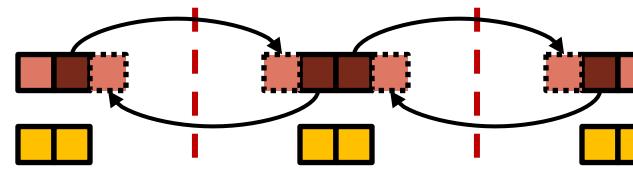
SPMD by Example (in pictures)

“Apply a 3-Point Stencil to a vector”

Conceptual View

$$\begin{aligned}
 & (\text{---} \quad \text{---} \quad \text{---} \quad \text{---} \quad \text{---}) \\
 + & (\text{---} \quad \text{---} \quad \text{---} \quad \text{---} \quad \text{---}) / 2 \\
 = & (\text{---} \quad \text{---} \quad \text{---} \quad \text{---} \quad \text{---})
 \end{aligned}$$

SPMD View



SPMD by Example (in code)



“Apply a 3-Point Stencil to a vector”



SPMD pseudo-code

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

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



COMPUTE

STORE

ANALYZE

SPMD by Example (in code)



“Apply a 3-Point Stencil to a vector”

Global-view code (Chapel)

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

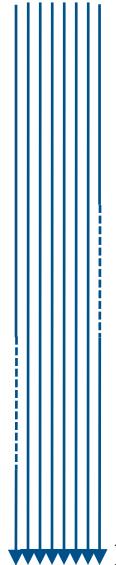
    forall i in 2..n-1 do
        B[i] = (A[i-1] + A[i+1])/2;
}
```



SPMD pseudo-code

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

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



SPMD by Example (in code)



“Apply a 3-Point Stencil to a vector”

Global-view code (Chapel)

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

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

Bug: Refers to uninitialized values at ends of A

SPMD pseudo-code

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

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



COMPUTE

STORE

ANALYZE

SPMD by Example (in code)



“Apply a 3-Point Stencil to a vector”

Global-view code (Chapel)

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

Communication becomes
geometrically more complex
for higher-dimensional arrays

SPMD pseudo-code

```
proc main() {  
    var n = 1000;  
    var p = numProcs(),  
        me = myProc(),  
        myN = n/p,  
        myLo = 1,  
        myHi = myN;  
    var A, B: [0..myN+1] real;  
  
    if (me < p-1) {  
        send(me+1, A[myN]);  
        recv(me+1, A[myN+1]);  
    } else  
        myHi = myN-1;  
    if (me > 0) {  
        send(me-1, A[1]);  
        recv(me-1, A[0]);  
    } else  
        myLo = 2;  
    forall i in myLo..myHi do  
        B[i] = (A[i-1] + A[i+1])/2;
```

Assumes p divides n

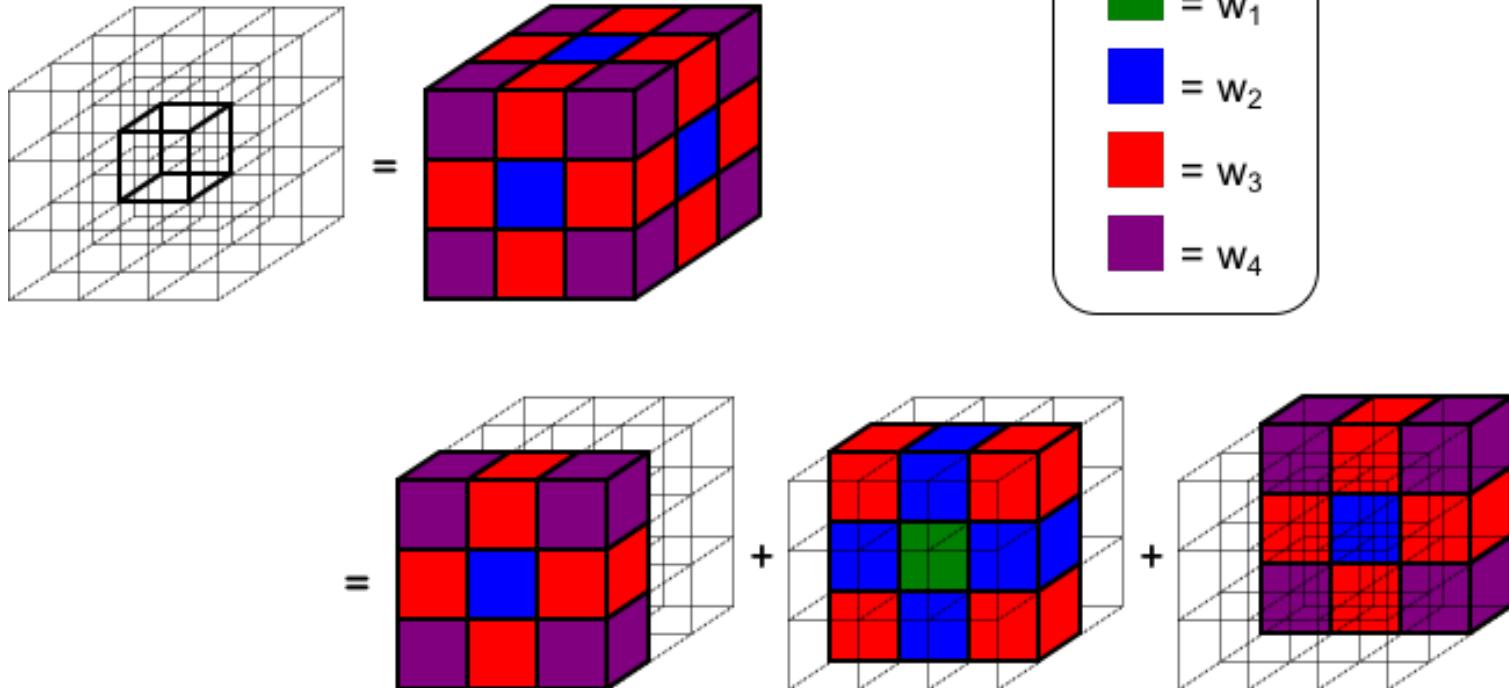


COMPUTE

STORE

ANALYZE

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



COMPUTE

STORE

ANALYZE

rprj3 in Fortran + MPI



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

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

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

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

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

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

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



COMPUTE

STORE

ANALYZE

rprj3 in Fortran + MPI

```

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

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

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

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

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

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

```

```

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

```



comm3: the communication for rprj3



```

subroutine comm3(u,n1,n2,n3,kk)
use caf_intrinsics
implicit none
include 'cafnpb.h'
include 'globals.h'
integer n1, n2, n3
double precision u(n1,n2,n3)
integer axis
integer axis3(u,n1,n2,n3)
if(.not. dead(kk)) then
do axis = 1, 3
  if( axis .eq. 1 ) then
    call sync_all()
    call give3( axis, +1, u, n1, n2, n3 )
    call take3( axis, -1, u, n1, n2, n3 )
    call sync_all()
    call take3( axis, -1, u, n1, n2, n3 )
    else
      call comm3p( axis, u, n1, n2, n3 )
    endif
  else
    do axis = 1, 3
      call sync_all()
      call sync_all()
      call zero3(u,n1,n2,n3)
    endif
  end
end

subroutine give3( axis, dir, u, n1, n2, n3, k )
use caf_intrinsics
implicit none
include 'cafnpb.h'
include 'globals.h'
integer axis, dir, n1, n2, n3, k, ierr
double precision u(n1,n2,n3)
integer i3, i2, i1, buff_len,buff_id
buff_id = 2 + dir
buff_len = 0
if( axis .eq. 1 ) then
  if( dir .eq.-1 ) then
    do i3=2,n3-1
      do i2=2,n2-1
        buff_len = buff_len + 1
        buff(buff_len,buff_id) = u( i2, i3 )
      enddo
    enddo
  else
    buff(1:buff_len,buff_id+1)[nbr](axis,direction) = u( 2, 1, i3 )
  endif
endif

subroutine take3( axis, dir, u, n1, n2, n3, k )
use caf_intrinsics
implicit none
include 'cafnpb.h'
include 'globals.h'
integer axis, dir, n1, n2, n3, k, ierr
double precision u(n1,n2,n3)
integer i3, i2, i1, buff_len,buff_id
buff_id = 2 + dir
buff_len = 0
if( axis .eq. 1 ) then
  if( dir .eq. +1 ) then
    do i3=2,n3-1
      do i2=2,n2-1
        do i1=1,n1
          buff_len = buff_len + 1
          buff(buff_len,buff_id) = u( i1, i2, i3 )
        enddo
      enddo
    enddo
  else
    buff(1:buff_len,buff_id+1)[nbr](axis,direction) = u( i1, i2, i3 )
  endif
endif

subroutine comm3p( axis, u, n1, n2, n3, kk )
use caf_intrinsics
implicit none
include 'cafnpb.h'
include 'globals.h'
integer axis, dir, n1, n2, n3, k, ierr
double precision u(n1,n2,n3)
integer i3, i2, i1, buff_len,buff_id
buff_id = 2 + dir
buff_len = 0
if( axis .eq. 1 ) then
  if( dir .eq. +1 ) then
    do i3=2,n3-1
      do i2=2,n2-1
        do i1=1,n1
          buff_len = buff_len + 1
          buff(buff_len,buff_id) = u( i1, i2, i3 )
        enddo
      enddo
    enddo
  else
    buff(1:buff_len,buff_id+1)[nbr](axis,direction) = u( i1, i2, i3 )
  endif
endif

subroutine subroutine take3( axis, dir, u, n1, n2, n3, k )
use caf_intrinsics
implicit none
include 'cafnpb.h'
include 'globals.h'
integer axis, dir, n1, n2, n3, k, ierr
double precision u(n1,n2,n3)
integer buff_id, indx
integer i3, i2, i1
buff_id = 3 + dir
indx = 0
if( axis .eq. 1 ) then
  if( dir .eq. +1 ) then
    do i3=2,n3-1
      do i2=2,n2-1
        do i1=1,n1
          indx = indx + 1
          u(i1,i2,i3) = buff(indx)
        enddo
      enddo
    enddo
  else
    buff(1:buff_len,buff_id+1)[nbr](axis,direction) = u( i1, i2, i3 )
  endif
endif

subroutine subroutine comm3p( axis, u, n1, n2, n3, kk )
use caf_intrinsics
implicit none
include 'cafnpb.h'
include 'globals.h'
integer axis, dir, n1, n2, n3, k, ierr
double precision u(n1,n2,n3)
integer i3, i2, i1, buff_len,buff_id
buff_id = 3 + dir
buff_len = 0
if( axis .eq. 1 ) then
  if( dir .eq. +1 ) then
    do i3=2,n3-1
      do i2=2,n2-1
        do i1=1,n1
          buff_len = buff_len + 1
          buff(buff_len,buff_id) = u( i1, i2, i3 )
        enddo
      enddo
    enddo
  else
    buff(1:buff_len,buff_id+1)[nbr](axis,direction) = u( i1, i2, i3 )
  endif
endif

subroutine subroutine give3( axis, dir, u, n1, n2, n3, k )
use caf_intrinsics
implicit none
include 'cafnpb.h'
include 'globals.h'
integer axis, dir, n1, n2, n3, k, ierr
double precision u(n1,n2,n3)
integer i3, i2, i1, buff_len,buff_id
buff_id = 3 + dir
buff_len = 0
if( axis .eq. 1 ) then
  if( dir .eq. +1 ) then
    do i3=2,n3-1
      do i2=2,n2-1
        do i1=1,n1
          indx = indx + 1
          u(i1,i2,i3) = buff(indx)
        enddo
      enddo
    enddo
  else
    buff(1:buff_len,buff_id+1)[nbr](axis,direction) = u( i1, i2, i3 )
  endif
endif

subroutine subroutine take3( axis, dir, u, n1, n2, n3, k )
use caf_intrinsics
implicit none
include 'cafnpb.h'
include 'globals.h'
integer axis, dir, n1, n2, n3, k, ierr
double precision u(n1,n2,n3)
integer buff_id, indx
integer i3, i2, i1
buff_id = 3 + dir
indx = 0
if( axis .eq. 1 ) then
  if( dir .eq. +1 ) then
    do i3=2,n3-1
      do i2=2,n2-1
        do i1=1,n1
          indx = indx + 1
          u(i1,i2,i3) = buff(indx)
        enddo
      enddo
    enddo
  else
    buff(1:buff_len,buff_id+1)[nbr](axis,direction) = u( i1, i2, i3 )
  endif
endif

subroutine subroutine comm3p( axis, u, n1, n2, n3, kk )
use caf_intrinsics
implicit none
include 'cafnpb.h'
include 'globals.h'
integer axis, dir, n1, n2, n3, k, ierr
double precision u(n1,n2,n3)
integer i3, i2, i1, buff_len,buff_id
buff_id = 3 + dir
buff_len = 0
if( axis .eq. 1 ) then
  if( dir .eq. +1 ) then
    do i3=2,n3-1
      do i2=2,n2-1
        do i1=1,n1
          buff_len = buff_len + 1
          buff(buff_len,buff_id) = u( i1, i2, i3 )
        enddo
      enddo
    enddo
  else
    buff(1:buff_len,buff_id+1)[nbr](axis,direction) = u( i1, i2, i3 )
  endif
endif

```



COMPUTE

STORE

ANALYZE

Copyright 2017 Cray Inc.

Being Gutsy, Gentle, and Kind to MPI



- It's enabled the vast majority of HPC results for the past ~20 years
- It's very analogous to assembly programming
 - explicitly move data from memory to registers
vs.
explicitly move data between compute nodes' memories
- Just like assembly, it's an important technology
 - for programming at low levels
 - for enabling higher-level technologies
- Yet, as with assembly, we should develop higher-level alternatives



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)



A color photograph of Rowan Atkinson as Mr. Bean. He is standing behind a dark wooden podium, wearing a black tuxedo and white shirt. He has his hands clasped in front of him and is looking slightly downwards. On the podium to his left is a silver teapot and a small glass. To his right is a white telephone. The background is a dark, textured wall.

And now for something
completely different.

[Instrumental]

Chapel Characteristics

The Liberty Bell March
John Philip Sousa
(unreleased?)



COMPUTE

|

STORE

|

ANALYZE

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



COMPUTE

|

STORE

|

ANALYZE

The Chapel Team at Cray (May 2016)



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



COMPUTE

STORE

ANALYZE

Chapel Community R&D Efforts



Lawrence Berkeley
National Laboratory



Yale

(and several others...)

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



COMPUTE

STORE

ANALYZE

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

Chapel in a Nutshell

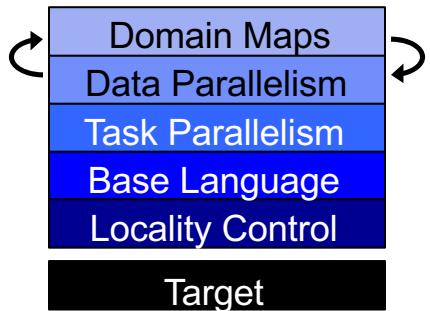
Revolution
The Beatles
The Beatles



Chapel language feature areas



Chapel language concepts



COMPUTE

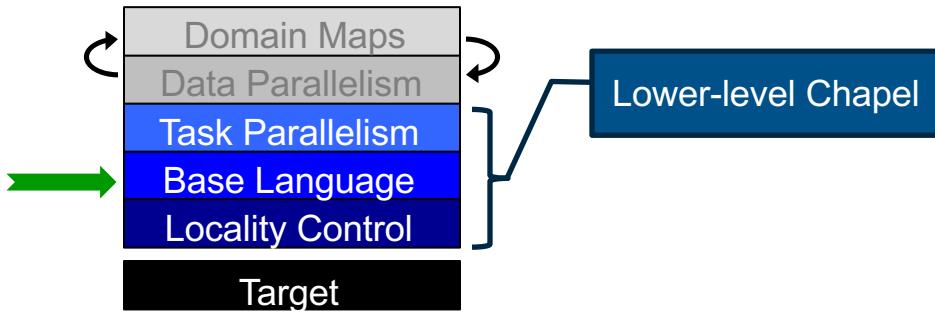
|

STORE

|

ANALYZE

Base Language



COMPUTE

STORE

ANALYZE

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



Modern iterators

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

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

```
config const n = 10;

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

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



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;
    }
}
```

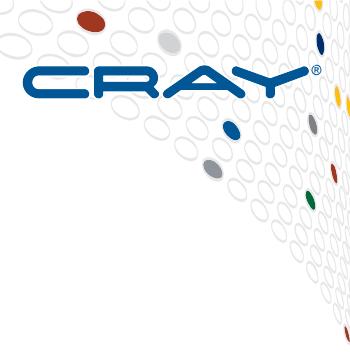
```
config const n = 10;

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

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



Base Language Features, by example



Static type inference for:

- arguments
- return types
- variables

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

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

```
config const n = 10;

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

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

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



Range types and operators

```
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
...
```



COMPUTE

STORE

ANALYZE

Base Language Features, by example



tuples

```
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
```

...



COMPUTE

STORE

ANALYZE

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
```

...

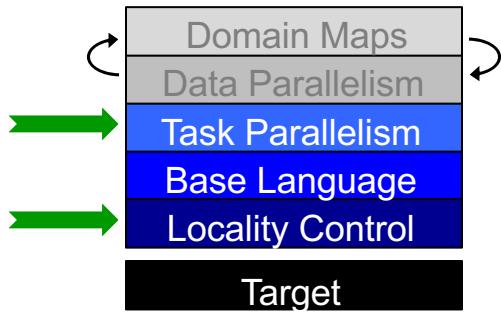


COMPUTE

STORE

ANALYZE

Task Parallelism and Locality Control



COMPUTE

|

STORE

|

ANALYZE

Task Parallelism and Locality, by example



taskParallel.chpl

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

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

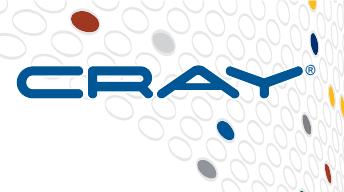


COMPUTE

STORE

ANALYZE

Task Parallelism and Locality, by example



Abstraction of
System Resources

taskParallel.chpl

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

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



COMPUTE

STORE

ANALYZE

Task Parallelism and Locality, by example



High-Level
Task Parallelism

taskParallel.chpl

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

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



COMPUTE

STORE

ANALYZE

Task Parallelism and Locality, by example



Control of Locality/Affinity

taskParallel.chpl

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

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



COMPUTE

STORE

ANALYZE

Task Parallelism and Locality, by example



Abstraction of
System Resources

taskParallel.chpl

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

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



COMPUTE

STORE

ANALYZE

Task Parallelism and Locality, by example



High-Level
Task Parallelism

taskParallel.chpl

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

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



COMPUTE

STORE

ANALYZE

Task Parallelism and Locality, by example



Not seen here:
Data-centric task coordination
via atomic and full/empty vars

taskParallel.chpl

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

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



COMPUTE

STORE

ANALYZE

Task Parallelism and Locality, by example



taskParallel.chpl

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

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



COMPUTE

STORE

ANALYZE

Parallelism and Locality: Distinct in Chapel



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

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

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

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

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

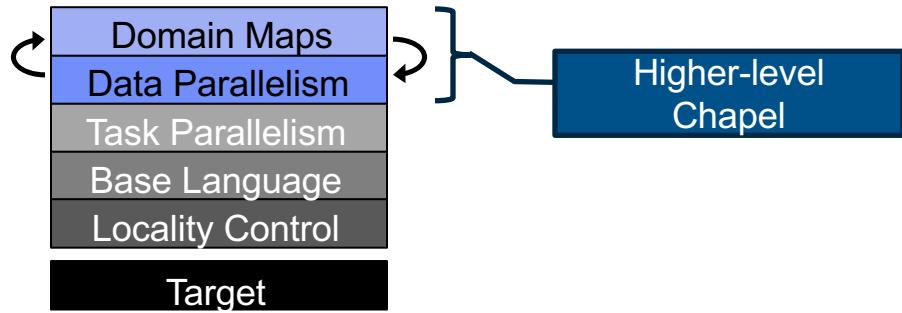
```
coforall i in 1..msgs do  
    on Locales[i%numLocales] do  
        writeln("Hello from task ", i,  
               " running on locale ", here.id);
```



Higher-Level Features



Chapel language concepts



COMPUTE

|

STORE

|

ANALYZE

Data Parallelism, by example



dataParallel.chpl

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

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

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



COMPUTE

STORE

ANALYZE

Data Parallelism, by example



Domains (Index Sets)

dataParallel.chpl

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

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

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

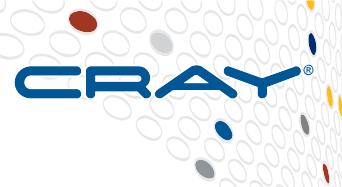


COMPUTE

STORE

ANALYZE

Data Parallelism, by example



Arrays

dataParallel.chpl

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

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



COMPUTE

STORE

ANALYZE

Data Parallelism, by example



Data-Parallel Forall Loops

dataParallel.chpl

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

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

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



COMPUTE

STORE

ANALYZE

Distributed Data Parallelism, by example



Domain Maps
(Map Data Parallelism to the System)

dataParallel.chpl

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

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



COMPUTE

STORE

ANALYZE

Distributed Data Parallelism, by example



dataParallel.chpl

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

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



COMPUTE

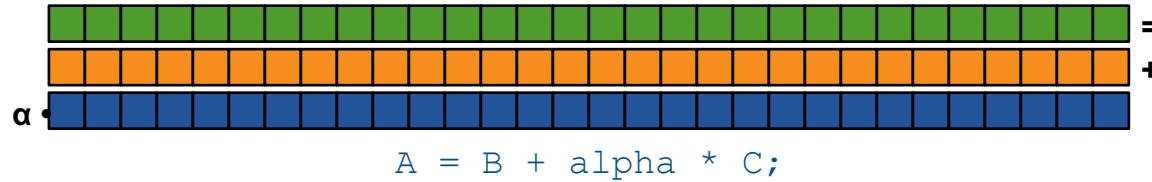
STORE

ANALYZE

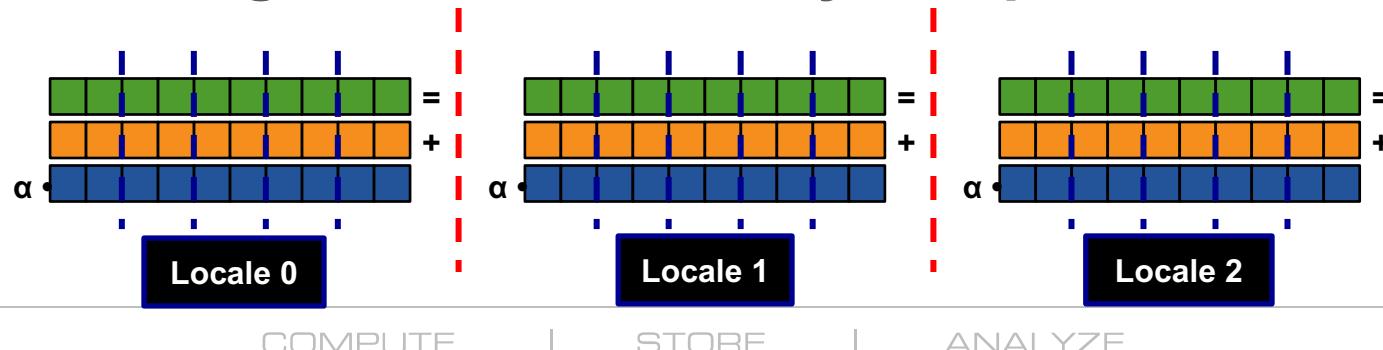
Domain Maps



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



...to the target locales' memory and processors:



Chapel's Domain Map Philosophy

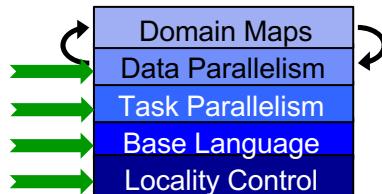


1. Chapel provides a library of standard domain maps

- 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

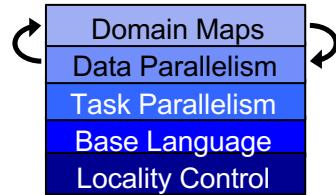


Chapel's Multiresolution Philosophy



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



COMPUTE

STORE

ANALYZE

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



Computer Language Benchmarks Game (CLBG)



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?

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

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...



COMPUTE

STORE

ANALYZE

Computer Language Benchmarks Game (CLBG)

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?

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

Chapel's approach to the CLBG:

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

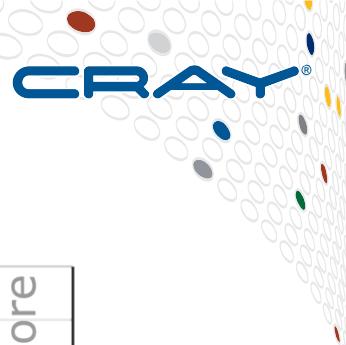


COMPUTE

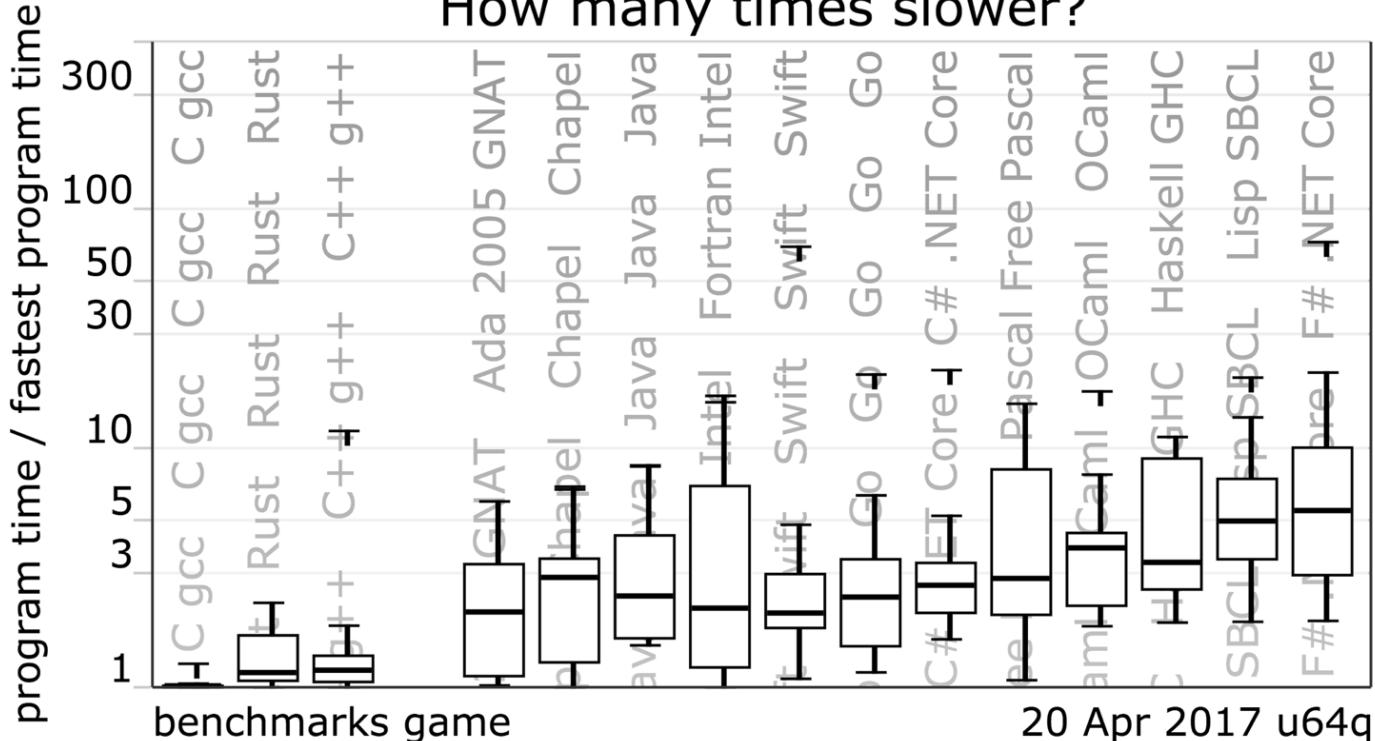
STORE

ANALYZE

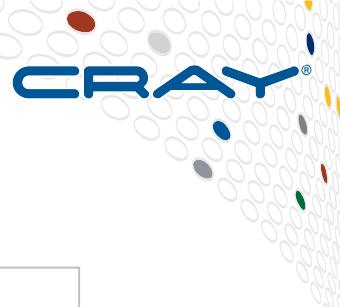
CLBG: Relative Performance Summary



How many times slower?



CLBG: Website



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

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

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

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

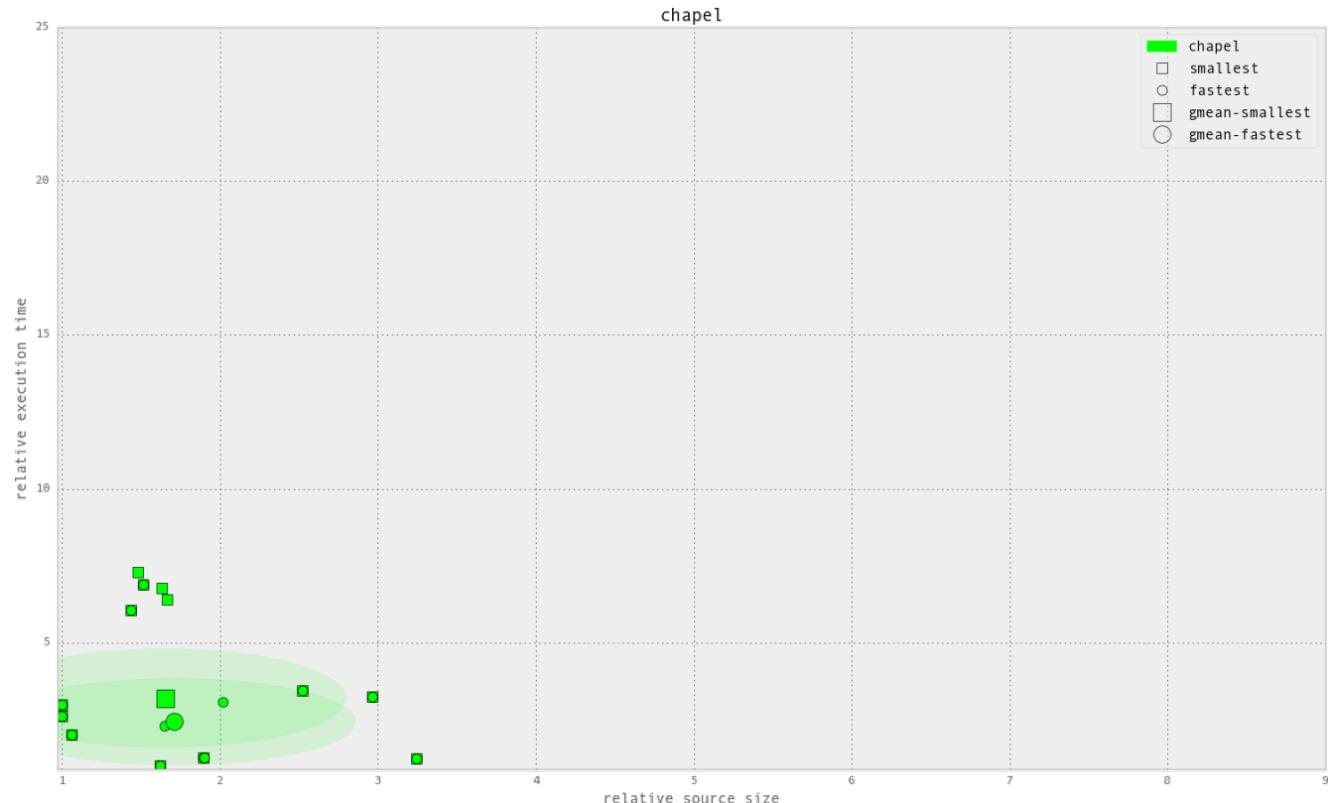


COMPUTE

STORE

ANALYZE

CLBG: Chapel entries

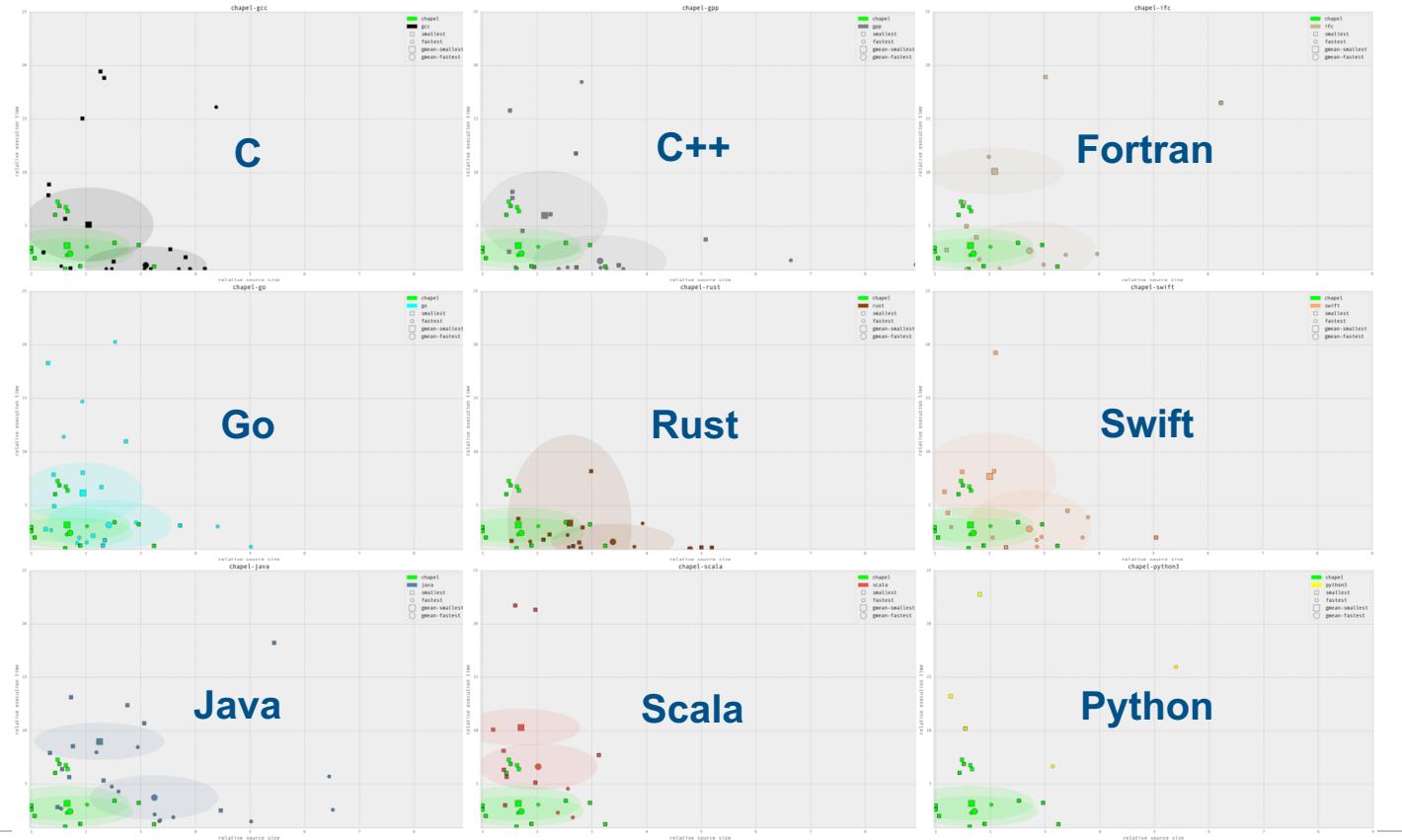
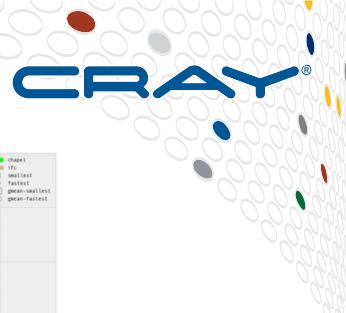


COMPUTE

STORE

ANALYZE

CLBG: Chapel vs. 9 key languages



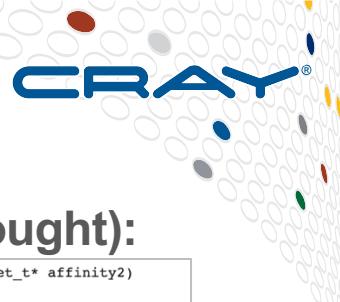
COMPUTE

STORE

ANALYZE



CLBG: Qualitative Comparisons



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

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

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

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

    print(group1);
    print(group2);

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

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

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

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

    delete place;
}
```

excerpt from 1210.gz Chapel entry

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

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

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

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

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

excerpt from 2863.gz C gcc entry



COMPUTE

STORE

ANALYZE

CLBG: Qualitative Comparisons



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

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

    const group1 = [i in 1..popSize1] new Chameneos(i, c);
    const group2 = [i in 1..popSize2] new Chameneos(i, c);

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

    print(group1);
    print(group2);

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

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

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

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

    delete place;
}
```

excerpt from 1210.gz Chapel entry

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

```
void get_affinity(int* is_smp, cpu_set_t* affinity1, cpu_set_t* affinity2)
```

```
size_t
char const*
size_t
char const*
```

```
proc holdMeetings(population, numMeetings) {
    const place = new MeetingPlace(numMeetings);

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

    delete place;
}
```

excerpt from 2863.gz C gcc entry

```
active_cpus;
f;
buf [2048];
pos;
cpu_idx;
physical_id;
core_id;
cpu_cores;
apic_id;
cpu_count;
i;

processor_str      = "processor";
processor_str_len = strlen(processor_str);
physical_id_str   = "physical id";
physical_id_str_len = strlen(physical_id_str);
core_id_str        = "core id";
cores              = n(core_id_str);
cores              = n(cpu_cores_str);
```

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



COMPUTE

STORE

ANALYZE

CLBG: Qualitative Comparisons



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

```
proc main() {  
  
    char const* core_id_str = "core id";  
    size_t core_id_str_len = strlen(core_id_str);  
    char const* cpu_cores_str = "cpu cores";  
    size_t cpu_cores_str_len = strlen(cpu_cores_str);  
  
    CPU_ZERO(&active_cpus);  
    sched_getaffinity(0, sizeof(active_cpus), &active_cpus);  
    cpu_count = 0;  
    for (i = 0; i != CPU_SETSIZE; i += 1)  
    {  
        if (CPU_ISSET(i, &active_cpus))  
        {  
            cpu_count += 1;  
        }  
    }  
  
    if (cpu_count == 1)  
    {  
        is_smp[0] = 0;  
        return;  
    }  
}
```

excerpt from 1210.gz Chapel entry

```
void get_affinity(int* is_smp, cpu_set_t* affinity1, cpu_set_t* affinity2)  
{  
    cpu_set_t active_cpus;  
    FILE* f;  
    char buf [2048];  
    pos;  
    cpu_idx;  
    physical_id;  
    core_id;  
    cpu_cores;  
    apic_id;  
    cpu_count;  
    i;  
  
    char const* processor_str = "processor";  
    size_t processor_str_len = strlen(processor_str);  
    physical_id_str = "physical id";  
    physical_id_str_len = strlen(physical_id_str);  
    core_id_str = "core id";  
    core_id_str_len = strlen(core_id_str);  
    cpu_cores_str = "cpu cores";  
    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] = 1;  
        CPU_ZERO(affinity1);  
    }  
}
```

excerpt from 2863.gz C gcc entry



COMPUTE

STORE

ANALYZE

*I have run
I have crawled
I have scaled...*

Chapel Scalability Results

U2

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

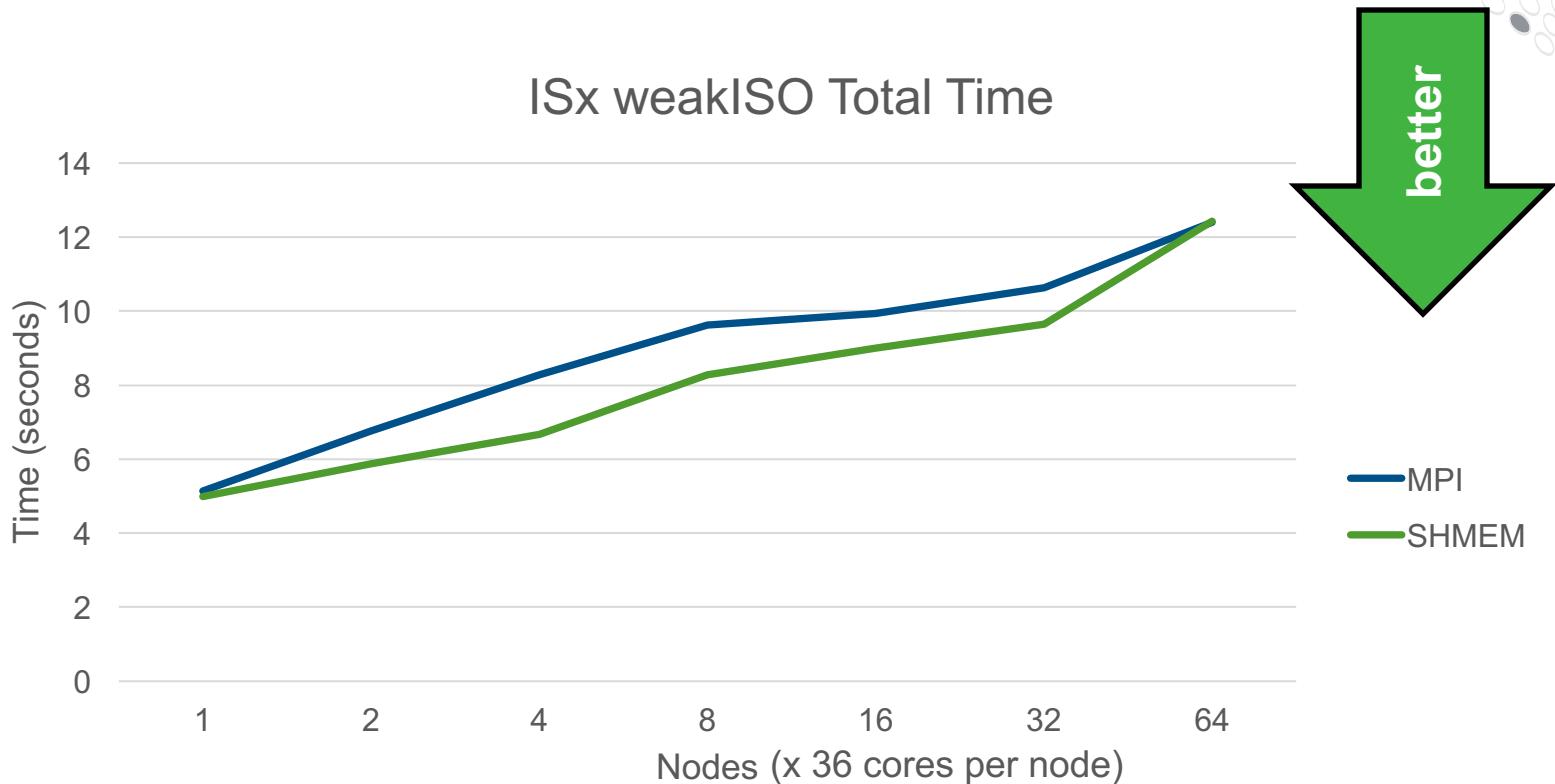


COMPUTE

STORE

ANALYZE

ISx Execution Time: MPI, SHMEM

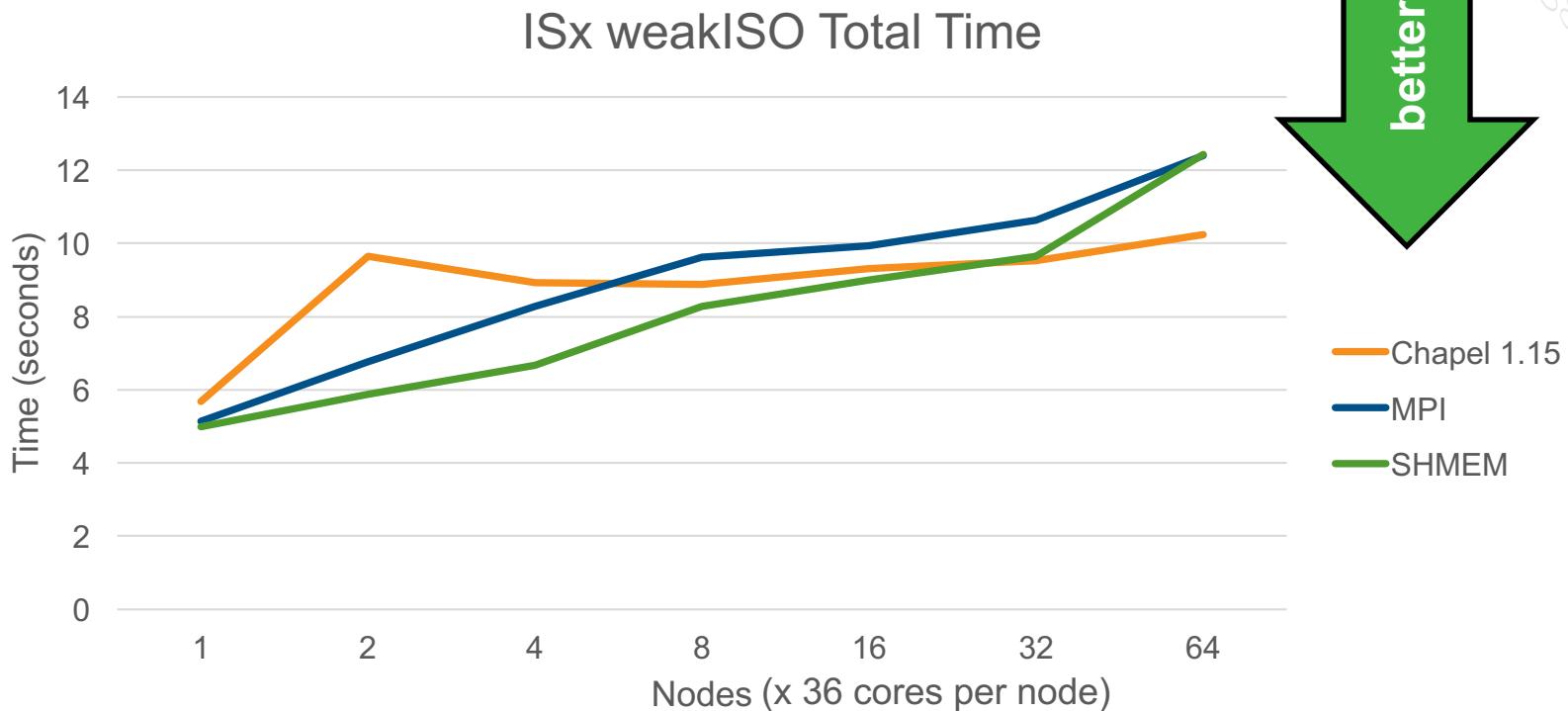


COMPUTE

STORE

ANALYZE

ISx Execution Time: MPI, SHMEM, Chapel

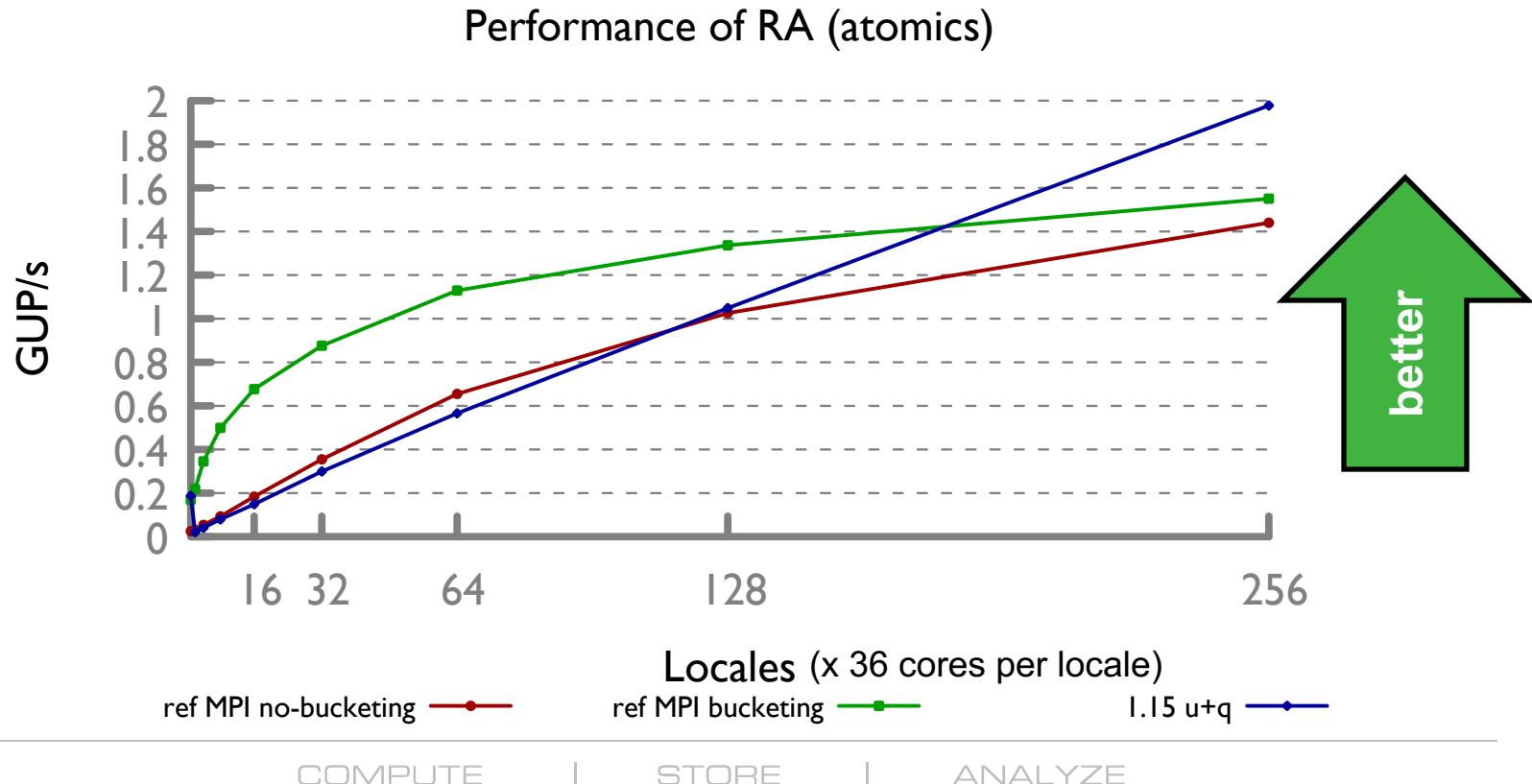


COMPUTE

STORE

ANALYZE

RA Performance: Chapel vs. MPI



*[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...



COMPUTE

|

STORE

|

ANALYZE

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



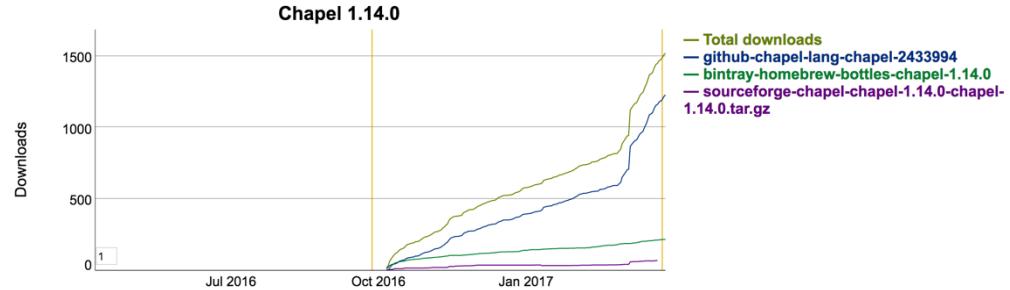
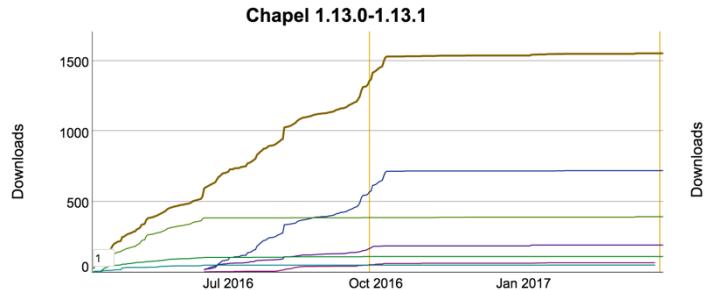
- **design and optimization of multiresolution language features:**
 - domain maps
 - parallel iterators
 - locale models
- **compiler architecture: outgrowing the initial research prototype**
- **tools: classic chicken-and-egg problem**
 - IDE
 - debuggers
 - performance analysis
 - package manager
 - 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



COMPUTE

STORE

ANALYZE



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

Further Resources

Rusholme Ruffians
The Smiths
Meat is Murder



COMPUTE

STORE

ANALYZE

How to Stalk Chapel



<http://facebook.com/ChapelLanguage>

<http://twitter.com/ChapelLanguage>

<https://www.youtube.com/channel/UCHmm27bYjhknK5mU7ZzPGsQ/>
chapel-announce@lists.sourceforge.net

The screenshot shows a Facebook page for 'Chapel Programming Language'. The page has 270 people reached and 1 post. A chart titled 'How many times slower?' compares the performance of various languages. The y-axis represents 'program time / fastest program time' from 1 to 300. The x-axis lists languages: C, C++, Fortran, Java, Julia, MATLAB, Octave, Python, R, and several .NET/C/C++/Fortran variants. The chart shows that most languages perform similarly to the fastest ones (around 1.0), while some like MATLAB and Octave show significantly higher times. A note at the bottom states: 'Vladimír Fukáč it measures how many programmers of given language can about that pace anyway. Fortran times are a joke and always be lowest.'



Chapel Parallel Programming Language

Home Videos Playlists Channels About 

Chapel videos



SC16 Chapel Tutorial Promo

Chapel Parallel Programming Language
6 months ago • 392 views

This is a ~4-minute promotional video for our SC16 Chapel tutorial, and also a good way to get a quick taste of Chapel. All codes shown represent complete Chapel programs, not...



Chapel Productive, Multiresolution Parallel Programming | Brad Chamberlain, Cray, Inc.

AM, Training
7 months ago • 651 views

Presented at the Argonne Training Program on Extreme-Scale Computing, Summer 2016.



CHI16 2016 keynote: "Chapel in the (Cosmological) Wild", Nikhil Padmanabhan

Chapel Parallel Programming Language
10 months ago • 277 views

This is Nikhil Padmanabhan's keynote talk from CHI16W 2016: the 3rd Annual Chapel Implementers and Users workshop. The slides are available at....



COMPUTE

STORE

ANALYZE

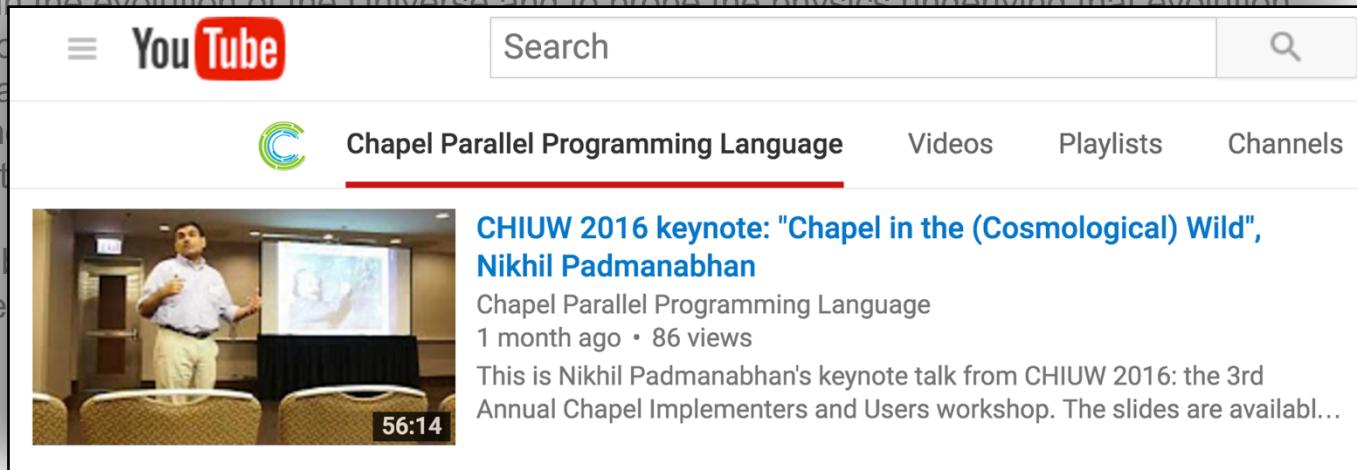
Copyright 2017 Cray Inc.

A Chapel talk to watch next (user perspective)

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

Nikhil Padmanabhan, Yale University Professor, Physics & Astronomy

Abstract: This talk aims to present my personal experiences using Chapel in my research. My research interests are in observational cosmology; more specifically, I use large surveys of galaxies to constrain the evolution of the Universe and to probe the physics underlying that evolution.



COMPUTE

STORE

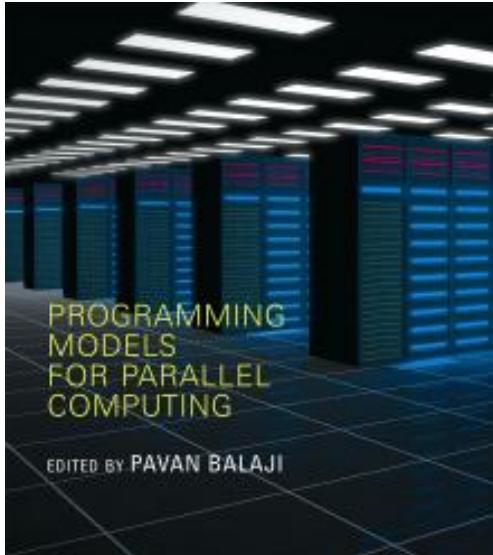
ANALYZE

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>

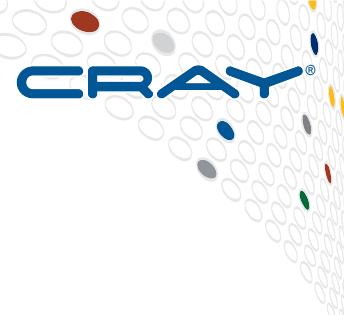


COMPUTE

STORE

ANALYZE

Suggested Reading (short attention spans)



[Chapel: Productive Parallel Programming](#), [Cray Blog](#), May 2013.

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

[Six Ways to Say “Hello” in Chapel](#) (parts [1](#), [2](#), [3](#)), [Cray Blog](#), Sep-Oct 2015.

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

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

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

[\[Ten\] Myths About Scalable Programming Languages](#), [IEEE TCSC Blog](#)

([index available on chapel.cray.com “blog articles” page](#)), Apr-Nov 2012.

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





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!)

ACCU
2017

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



COMPUTE

STORE

ANALYZE

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.





CRAY
THE SUPERCOMPUTER COMPANY

accu
2017