Chapel Background & Overview
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.
Motivation for Chapel

Q: Can a single language be…
   …as productive as Python?
   …as fast as Fortran?
   …as portable as C?
   …as scalable as MPI?
   …as fun as <your favorite language here>?

A: We believe so.
The Challenge

Q: So why don’t we have such languages 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
STREAM Triad: a trivial parallel computation

**Given:** $m$-element vectors $A, B, C$

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

**In pictures:**

\[
\begin{align*}
A & = B + C \\
\alpha & = \alpha
\end{align*}
\]
STREAM Triad: a trivial parallel computation

**Given:** \( m \)-element vectors \( A, B, C \)

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

**In pictures, in parallel:**

\[
\begin{align*}
A & = B + \alpha \cdot C \\
B & = B + \alpha \cdot C \\
C & = B + \alpha \cdot C
\end{align*}
\]
STREAM Triad: a trivial parallel computation

**Given:** $m$-element vectors $A$, $B$, $C$

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

In pictures, in parallel (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):
#include <hpcc.h>

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

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

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

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

  return errCount;
}

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

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

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

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

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

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

  HPCC_free(c);
  HPCC_free(b);
  HPCC_free(a);
#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);
#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);
if( N % dimBlock.x != 0 )
dimGrid.x+=1;

set_array<<<dimGrid,dimBlock>>>(d_b, .5f, N);
set_array<<<dimGrid,dimBlock>>>(d_c, .5f, N);
scalar=3.0f;
STREAM_Triad<<<dimGrid,dimBlock>>>(d_b, d_c, d_a, scalar, N);
cudaThreadSynchronize();
cudaFree(d_a);
cudaFree(d_b);
cudaFree(d_c);
}

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

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

Why so many programming models?

HPC tends to approach programming models bottom-up:
Given a system and its core capabilities…
…provide features that can access the available performance.
  • portability, generality, programmability: not strictly necessary.

<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/m 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
Rewinding a few slides...

**CUDA**

```c
#define N 2000000

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

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

    dim3 dimGrid(128);
    stream_Triad<<<dimGrid,dimBlock>>>(d_b, d_c, d_a, scalar, N);
    cudaMemcpy((void**)&d_a, sizeof(float)*N);
    cudaMemcpy((void**)&d_b, sizeof(float)*N);
    cudaMemcpy((void**)&d_c, sizeof(float)*N);
}

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

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

**HPC suffers from too many distinct notations for expressing parallelism and locality**

```c
#include <hpcc.h>
#ifdef _OPENMP
#include <omp.h>
#endif
static int VectorSize;
static double *a, *b, *c;
int HPCC_StarStream(HPCC_Params *params) {
    int myRank, commSize;
    MPI_Comm comm = MPI_COMM_WORLD;
    MPI_Comm_rank( comm, &myRank );
    MPI_Reduce( &rv, &errCount, 1, MPI_INT, MPI_SUM, 0, comm );
    return errCount;
}

int HPCC_Stream(HPCC_Params *params, int doIO) {
    register int j;
    double scalar;
    VectorSize = HPCC_LocalVectorSize( params, 3, sizeof(double), 0 );
    a = HPCC_XMALLOC( double, VectorSize );
    b = HPCC_XMALLOC( double, VectorSize );
    c = HPCC_XMALLOC( double, VectorSize );
    if (!a || !b || !c) {
        if (c) HPCC_free(c);
        if (b) HPCC_free(b);
        if (a) HPCC_free(a);
        if (doIO) {
            fprintf( outFile, "Failed to allocate memory (%d)\n", VectorSize );
            fclose( outFile );
        }
        return 1;
    }
    #ifdef _OPENMP
    #pragma omp parallel for
    #endif
    for (j=0; j<VectorSize; j++) {
        b[j] = 2.0;
        c[j] = 1.0;
    }
    scalar = 3.0;
    #ifdef _OPENMP
    #pragma omp parallel for
    #endif
    for (j=0; j<VectorSize; j++)
        a[j] = b[j] + scalar*c[j];

    HPCC_free(c);
    HPCC_free(b);
    HPCC_free(a);
    return 0;
}
```
**STREAM Triad: Chapel**

```
#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_StarStream( params, 0 == myRank );
MPI_Reduce( &rv, &errCount, 1, MPI_INT, MPI_SUM, 0, comm );
return errCount;
}
int HPCC_StarStream(HPCC_Params *params, int doIO)
{ 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] = 0.0;
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;
}
```
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.”
Chapel is Portable

- Chapel is designed to be hardware-independent

- The current release requires:
  - a C/C++ compiler
  - a *NIX environment (Linux, OS X, BSD, Cygwin, …)
  - POSIX threads
  - UDP, MPI, or RDMA (if distributed memory execution is desired)

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

* = academic work only; not yet supported in the official release
Chapel is Open-Source

- Chapel’s development is hosted at GitHub
  - https://github.com/chapel-lang

- Chapel is licensed as Apache v2.0 software

- Instructions for download + install are online
  - see http://chapel.cray.com/download.html to get started
The Chapel Team at Cray (May 2016)

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

(and several others...)

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

✓ Chapel Motivation and Background
➢ Chapel in a Nutshell
● Chapel Project: Past, Present, Future
Multiresolution Design: Support multiple tiers of features

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

Chapel language concepts

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

- build the higher-level concepts in terms of the lower
- permit the user to intermix layers arbitrarily
Base Language

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

Lower-level Chapel

Target Machine
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 f in fib(n) do
    writeln(f);
```

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>5</th>
<th>8</th>
<th>...</th>
</tr>
</thead>
</table>
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

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>5</th>
<th>8</th>
<th>...</th>
</tr>
</thead>
</table>
Base Language Features, by example

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

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

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

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

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

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

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

```
config const n = 10;

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

```
tuples
  fib #0 is 0
  fib #1 is 1
  fib #2 is 1
  fib #3 is 2
  fib #4 is 3
  fib #5 is 5
  fib #6 is 8
  ...
```
Base Language Features, by example

```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 (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
Task Parallelism, Locality Control, by example

taskParallel.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, Locality Control, by example

Abstraction of System Resources

```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
```
High-Level Task Parallelism

```
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);
  }
```
Task Parallelism, Locality Control, 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, Locality Control, 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
```
Task Parallelism, Locality Control, by example

High-Level Task Parallelism

taskParallel.chpl

```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);
  }
```

```sh
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, Locality Control, by example

```
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
```

Not seen here:
Data-centric task coordination via atomic and full/empty vars
Task Parallelism, Locality Control, by example

taskParallel.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
Parallelism and Locality: Orthogonal in Chapel

- This is a **parallel**, but local program:
  ```chapel
coforall i in 1..msgs do
  writeln("Hello from task ", i);
  ```

- This is a **distributed**, but serial program:
  ```chapel
writeln("Hello from locale 0!");
on Locales[1] do writeln("Hello from locale 1!");
on Locales[2] do writeln("Hello from locale 2!");
  ```

- This is a **distributed parallel** program:
  ```chapel
coforall i in 1..msgs do
  on Locales[i%numLocales] do
  writeln("Hello from task ", i,
  " running on locale ", here.id);
  ```
Higher-Level Features

Chapel language concepts

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

Higher-level Chapel
Data Parallelism, by example

dataParallel.chpl

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

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

prompt> chpl dataParallel.chpl -o dataParallel
prompt> ./dataParallel --n=5

```
1.1 1.3 1.5 1.7 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
```
Data Parallelism, by example

Domains (Index Sets)

dataParallel.chpl

```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

Data-Parallel Forall Loops

```
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);
```

call dataParallel.chpl

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

**Domain Maps**  
(Map Data Parallelism to the System)

```
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
```
Outline

✓ Chapel Motivation and Background
✓ Chapel in a Nutshell
➢ Chapel Project: Past, Present, Future
Chapel’s Origins: HPCS

DARPA HPCS: High Productivity Computing Systems

- **Goal:** improve productivity by a factor of 10x
- **Timeframe:** Summer 2002 – Fall 2012
- Cray developed a new system architecture, network, software stack...
  - this became the very successful Cray XC30™ Supercomputer Series

...and a new programming language: Chapel
(at that point, essentially a research prototype)
Chapel’s 5-year push

- Based on positive user response to Chapel under HPCS, Cray undertook a five-year effort to improve it
  - we’re just completing our fourth year

- **Focus Areas:**
  1. Improving **performance** and scaling
  2. **Fixing** immature aspects of the language and implementation
     - e.g., strings, memory management and leaks, OOP, error handling, …
  3. **Porting** to emerging architectures
     - Intel Xeon Phi, accelerators, heterogeneous processors and memories, …
  4. Improving **interoperability**
  5. Growing the Chapel user and developer **community**
     - including non-scientific computing communities
  6. Exploring transition of Chapel **governance** to a neutral, external body
A Year in the Life of Chapel

- **Two major releases per year** (April / October)
  - ~a month later: detailed *release notes*
  - latest release: Chapel 1.15, released April 6th 2017
    - release notes due to be published this week or next

- **CHIUW: Chapel Implementers and Users Workshop** (~June)
  - (4th annual) **CHIUW 2017**, June 1-2 at IPDPS (Orlando, FL)
  - talks from members of the broad community + a Chapel code camp

- **SC** (Nov)
  - tutorials, panels, BoFs, posters, educator sessions, exhibits, …
  - annual **CHUG (Chapel Users Group) happy hour**

- **Talks, tutorials, research visits, blog posts, …** (year-round)
Chapel is a Work-in-Progress

- Currently being picked up by early adopters
  - ~3000+ downloads per year across two releases

- Users who try it generally like what they see
A notable early adopter

Chapel in the (Cosmological) Wild

Nikhil Padmanabhan, Yale University Professor, Physics & Astronomy

1:00 – 2:00

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 underlying physics. 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.

Chapel Parallel Programming Language

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

Chapel Parallel Programming Language

1 month ago • 86 views

This is Nikhil Padmanabhan's keynote talk from CHIUW 2016: the 3rd Annual Chapel Implementers and Users workshop. The slides are availabl...
3. Core Language Feature Improvements

- Historical problems that are now much better:
  - strings, memory leaks, memory management
- Areas that have improved, but are still in-progress:
  - initializers (constructor replacement), error-handling
Memory Leak Improvements

- Effort in recent years has dramatically reduced leaks
  - most remaining cases are due to user-level leaks in tests themselves

Memory Leaks for all Tests

- Y-axis: bytes (8.00e+9, 6.00e+9, 4.00e+9, 2.00e+9, 0)
Chapel: Top 3 Traditional Barriers to Use

3. Core Language Feature Improvements
   - Historical problems that are now much better:
     - strings, memory leaks, memory management, interoperability, generics
   - Areas that have improved, but are still in-progress:
     - initializers (constructor replacement), error-handling

2. Access to Standard Libraries
   - Situation has improved significantly over past few years
     - Several core libraries added:
       - BigInteger, BitOps, DateTime, FileSystem, Random, Reflection, Spawn, …
     - As well as access to many standard libraries / technologies:
       - BLAS, Curl, FFTW, Futures, HDFS, LAPACK, LinearAlgebra, MPI, ZMQ, …

1. Performance
   - Particularly for old-school HPC users, performance is crucial
   - That said, as of this month’s release, we’re reaching parity more often
Single-Locale Improvements in Execution Time

- Single-locale is increasingly on par with C / C++ / OpenMP
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?

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

Website that supports cross-language game / comparisons

- 13 toy benchmark programs
- exercises key features like:
  - memory management
  - tasking and synchronization
  - vectorization
  - big integers
  - strings and regular expressions
  - specific approach prescribed

Take results w/ grain of salt
- other programs may be different
- not to mention other programmers
- specific to this platform / OS / …

That said, it’s one of the only games in town…
Chapel’s approach to CLBG:
- want to know how we compare
- strive for entries that are elegant rather than heroic
  - e.g., “Want to learn how program x works? Check out the Chapel version.”

The Computer Language Benchmarks Game

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

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

<table>
<thead>
<tr>
<th>Ada</th>
<th>C</th>
<th>Chapel</th>
<th>Clojure</th>
<th>C#</th>
<th>C++</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dart</td>
<td>Erlang</td>
<td>F#</td>
<td>Fortran</td>
<td>Go</td>
<td>Hack</td>
</tr>
<tr>
<td>Haskell</td>
<td>Java</td>
<td>JavaScript</td>
<td>Lisp</td>
<td>Lua</td>
<td></td>
</tr>
<tr>
<td>OCaml</td>
<td>Pascal</td>
<td>Perl</td>
<td>PHP</td>
<td>Python</td>
<td></td>
</tr>
<tr>
<td>Racket</td>
<td>Ruby</td>
<td>JRuby</td>
<td>Rust</td>
<td>Scala</td>
<td></td>
</tr>
<tr>
<td>Smalltalk</td>
<td>Swift</td>
<td>TypeScript</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Can sort results by execution time, code size, memory or CPU use:

**The Computer Language Benchmarks Game**

---

### Chameneos-Redux

**Description**

- Sort results by execution time, code size, memory or CPU use.

<table>
<thead>
<tr>
<th>Source</th>
<th>Secs</th>
<th>Mem</th>
<th>Gz</th>
<th>CPU</th>
<th>CPU Load</th>
</tr>
</thead>
<tbody>
<tr>
<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>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>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>Chapel #2</td>
<td>1.39</td>
<td>76,564</td>
<td>1210</td>
<td>5.43</td>
<td>99% 99% 98% 99%</td>
</tr>
<tr>
<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>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>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>Java #4</td>
<td>4.82</td>
<td>37,132</td>
<td>1607</td>
<td>16.73</td>
<td>98% 98% 54% 99%</td>
</tr>
<tr>
<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>Java</td>
<td>6.13</td>
<td>53,760</td>
<td>1770</td>
<td>8.76</td>
<td>42% 45% 41% 16%</td>
</tr>
<tr>
<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>C# .NET Core</td>
<td>6.59</td>
<td>86,076</td>
<td>1400</td>
<td>22.96</td>
<td>99% 82% 78% 91%</td>
</tr>
<tr>
<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>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>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>
CLBG: Chapel Standings as of Apr 20th

- **8 / 13 programs in top-20 smallest:**
  - two #1 smallest:
    - n-body
    - thread-ring
  - 2 others in the top-5 smallest:
    - pidigits
    - spectral-norm
  - 1 other in the top-10 smallest:
    - regex-redux
  - 3 others in the top-20 smallest:
    - chameneos-redux
    - mandelbrot
    - meteor-contest

- **12 /13 programs in top-20 fastest:**
  - one #1 fastest:
    - pidigits
  - 3 others in the top-5 fastest:
    - chameneos-redux
    - meteor-contest
    - thread-ring
  - 3 others in the top-10 fastest:
    - fannkuch-redux
    - fasta
    - mandelbrot
  - 5 others in the top-20 fastest:
    - binary-trees
    - k-nucleotide
    - n-body
    - regex-redux
    - spectral-norm
Can also compare languages pair-wise:

### The Computer Language Benchmarks Game

#### Chapel programs versus Go
all other Chapel programs & measurements

by benchmark task performance

<table>
<thead>
<tr>
<th>benchmark</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>regex-redux</td>
<td>Chapel</td>
<td>10.02</td>
<td>1,022,052</td>
<td>477</td>
<td>19.68</td>
<td>99% 72% 14% 12%</td>
</tr>
<tr>
<td></td>
<td>Go</td>
<td>29.51</td>
<td>352,804</td>
<td>798</td>
<td>61.51</td>
<td>77% 49% 43% 40%</td>
</tr>
<tr>
<td>binary-trees</td>
<td>Chapel</td>
<td>14.32</td>
<td>324,660</td>
<td>484</td>
<td>44.15</td>
<td>100% 58% 78% 75%</td>
</tr>
<tr>
<td></td>
<td>Go</td>
<td>34.77</td>
<td>269,068</td>
<td>654</td>
<td>132.04</td>
<td>95% 97% 95% 95%</td>
</tr>
<tr>
<td>fannkuch-redux</td>
<td>Chapel</td>
<td>11.38</td>
<td>46,056</td>
<td>728</td>
<td>45.18</td>
<td>100% 99% 99% 100%</td>
</tr>
<tr>
<td></td>
<td>Go</td>
<td>15.81</td>
<td>1,372</td>
<td>900</td>
<td>62.92</td>
<td>100% 100% 99% 99%</td>
</tr>
</tbody>
</table>
Can also browse program source code (but this requires actual thought):

```c
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 = cpu_idx;
    int physical_id;
    cpu_id;
    int cpu_cores;
    int apic_id;
    size_t cpu_count;
    size_t i;
    char const* processor_str = "processor";
    size_t processor_str_len = strlen(processor_str);
    char const* physical_id_str = "physical id";
    size_t physical_id_str_len = strlen(physical_id_str);
    char const* core_id_str = "core id";
    size_t core_id_str_len = strlen(core_id_str);
    char const* cpu_cores_str = "cpu cores";
    size_t cpu_cores_str_len = strlen(cpu_cores_str);
    CPU_ZERO(&active_cpus);
sched_getaffinity(0, sizeof(active_cpus), &active_cpus);
    cpu_count = 0;
    for (i = 0; i != CPU_SETSIZE; i += 1) {
        if (CPU_ISSET(i, &active_cpus)) {
            cpu_count += 1;
        }
    }
    if (cpu_count == 1) {
        is_smp[0] = 0;
        return;
    }
    is_smp[0] = 1;
    CPU_ZERO(affinity1);
}
```

```c
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);
//    for all c in population do // create a task for chameneos
//        c.haveMeetings(place, population);
//        delete place;
//}
```

excerpt from 1210 gz Chapel #2 entry

excerpt from 2863 gz C gcc #5 entry
Can also browse program source code (but this requires actual thought):

```chapel
proc main() {
    proc printColorEquations() {
        for c1 in group1 do delete c1;
        for c2 in group2 do delete c2;
    }
}

void get_affinity(int* is_omp, cpu_set_t* affinity1, cpu_set_t* affinity2) {
    cobegin {
        proc holdMeetings(population, numMeetings) {
            const place = new MeetingPlace(numMeetings);
            coforall c in population do c.haveMeetings(place, population);
        }

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

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

excerpt from 1210 gz Chapel #2 entry
excerpt from 2863 gz C gcc #5 entry
```
Site summary: relative performance (sorted by geometric mean)
● site has a sound philosophy about too-easy answers

We want easy answers, but easy answers are often incomplete or wrong. You and I know, there's more we should understand:

| stories | details | fast? | conclusions |

● yet, most readers probably still jump to conclusions
  ● execution time dominates default (or only) views of results
  ● it's simply human nature

● we're interested in elegance as well as performance
  ● elegance is obviously in the eye of the beholder
    ● we compare source codes manually
    ● but then use CLBG's code size metric as a quantitative stand-in
  ● want to be able to compare both axes simultaneously
  ● to that end, we used scatter plots to compare implementations
Chapel entries
Chapel vs. 9 other languages
Chapel vs. 9 other languages (zoomed out)
Multi-locale Improvements in Execution Time

- Multi-locale performance is improving significantly as well

![Graphs showing improvements in execution time for different benchmarks and applications](image-url)
3 Key Multi-Locale Communication Benchmarks

STREAM Triad:
- measures embarrassingly / pleasingly parallel computation

RA:
- measures random updates to a large distributed array

ISx:
- measures bucket-exchange idiom
STREAM Triad: Chapel vs. MPI Scalability

Performance of STREAM
(GASNet/mpi+qthreads)

GB/s

Locales
Reference 1.11 EP 1.12 EP 1.12 Global
1.11 Global

Copyright 2017 Cray Inc.
RA: Chapel vs. MPI Scalability

Performance of RA (atomics)

GUP/s

Locales

16 32 64 128 256

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

Copyright 2017 Cray Inc.
ISx: Performance Summary

- Gathered on Cray XC with default problem size
  - reference versions

![Graph showing ISx weakISO Total Time vs Nodes]

**Graph Details:**
- **Y-axis:** Time (seconds)
- **X-axis:** Nodes (1 to 64)
- Two lines:
  - Blue: MPI
  - Green: SHMEM
ISx: Performance Summary

- Gathered on Cray XC with default problem size
  - adding Chapel, six months ago:

**ISx weakISO Total Time**

- Time (seconds)
- Nodes

**Graph**

- Chapel 1.14
- MPI
- SHMEM

Copyright 2017 Cray Inc.
Gathered on Cray XC with default problem size
- adding Chapel, today:

ISx: Performance Summary

![Graph showing ISx weakISO Total Time vs. Nodes for different versions of Chapel and MPI.]
**ISx: Performance Summary**

- Gathered on Cray XC with default problem size
  - dropping the old Chapel timings, and zooming in:
  - ISx weakISO Total Time

![Graph showing performance comparison between Chapel 1.15, MPI, and SHMEM across different nodes.](image-url)
Overview Summary

- Chapel has nice features for parallelism and locality
- Traditional reasons for not using Chapel are falling away
  - performance specifically is becoming less of a concern with time
- Aiming for a “version 2.0 release” over the near year or so
  - intent: no further breaking changes after that point
High-level Questions about Chapel?
Full-size CLBG Scatter Plots
Chapel vs. C
Chapel vs. C (zoomed out)
Chapel vs. C++
Chapel vs. C++ (zoomed out)
Chapel vs. Fortran

![Graph showing relative execution time vs. relative source size for Chapel and Fortran code. The graph includes data points for Chapel and Fortran, with Chapel represented in green and Fortran in brown. The x-axis represents relative source size, while the y-axis represents relative execution time. The graph also includes different markers for different performance metrics: smallest, fastest, gmean-smallest, and gmean-fastest.]
Chapel vs. Fortran (zoomed out)
Chapel vs. Go

A scatter plot comparing Chapel and Go based on relative execution time and relative source size. The plot shows the performance of different code sizes and execution durations, with Chapel and Go represented by different colors and symbols.
Chapel vs. Go (zoomed out)
Chapel vs. Rust
Chapel vs. Rust (zoomed out)
Chapel vs. Swift
Chapel vs. Swift (zoomed out)
Chapel vs. Java
Chapel vs. Java (zoomed out)
Chapel vs. Scala
Chapel vs. Scala (zoomed out)
Chapel vs. Python

The diagram represents a comparison of Chapel and Python based on the relative execution time and relative source size. Each point on the graph corresponds to a different scenario or experiment, with Chapel and Python represented by different colors. The x-axis shows the relative source size, while the y-axis shows the relative execution time. The different markers and colors help identify trends and performance differences between the two languages.
Chapel vs. Python (zoomed out)
Legal Disclaimer

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

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

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

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

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

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

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