Chapel Overview
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.
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:**

- $A$
- $B$
- $C$

$= \quad + \quad \cdot \quad \alpha$
STREAM Triad: a trivial parallel computation

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

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

**In pictures, in parallel:**

![Diagram showing parallel computation of A, B, and C vectors.](image)
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):**
STREAM Triad: MPI

```c
#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 );

    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];

    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;
    }
}
```
#include <hpcc.h>
#else_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);
}
STREAM Triad: MPI+OpenMP vs. CUDA

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

CUDA

```c
#define N  2000000

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

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

    dim3 dimBlock(128);
    dim3 dimGrid((N+dimBlock.x-1)/dimBlock.x ,dimBlock.x);
    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);
}
```

MPI + OpenMP

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

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

int HPCC_StarStream(HPCC_Params *params) {
    int myRank, commSize;
    int rv, errCount;
    MPI_Comm comm = MPI_COMM_WORLD;
    MPI_Comm_size( comm, &commSize );
    MPI_Comm_rank( comm, &myRank );
    rv = HPCC_Stream( params, myRank);
    MPI_Reduce( &rv, &errCount, 1, MPI_INT, MPI_SUM, 0, comm );
    return errCount;
}

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

__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/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;

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

    dim3 dimBlock(128);
    dim3 dimGrid(N/dimBlock.x);
    if( N % dimBlock.x != 0 )
        dimGrid.x+=1;

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

HPC suffers from too many distinct notations for expressing parallelism and locality
STREAM Triad: Chapel

**Chapel**

```chapel
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;
```

**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.
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](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](http://chapel.cray.com/download.html) to get started
14 full-time employees + 2 summer interns
(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
**Chapel’s Multiresolution Philosophy**

**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

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

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

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

```rust
config const n = 10;

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

0
1
1
2
3
5
8
...

Copyright 2016 Cray Inc.
Base Language Features, by example

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

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

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

  for i in 1..n {
    yield current;
    current += next;
    current <=> next;
  }
}
```
Base Language Features, by example

```go
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

Copyright 2016 Cray Inc.
Base Language Features, by example

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

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

```python
class config {
    const n = 10;
}
```

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

Zippered iteration

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

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

```
config const n = 10;
for (i,f) in zip(0..#n, fib(n)) do
  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;
  }
}
```

```
cfg const n = 10;
for (i,f) in zip(0..#n, fib(n)) do
  writeln("fib #", i, " is ", f);
```

tuples

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

Base Language Features, by example

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

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

```
config const n = 10;

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

```
fib #0 is 0
fib #1 is 1
fib #2 is 1
fib #3 is 2
fib #4 is 3
fib #5 is 5
fib #6 is 8
...```
Task Parallelism
Task Parallelism, Locality Control, by example

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

```
prompt> chpl taskParallel.chpl -o taskParallel
prompt> ./taskParallel --numLocales=2
Hello from task 1 of 2 running on n1033
Hello from task 2 of 2 running on n1032
Hello from task 2 of 2 running on n1033
Hello from task 1 of 2 running on n1032
```
Task Parallelism, Locality Control, by example

Abstraction of System Resources

taskParallel.chpl

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

```chpl
coforall loc in Locales do
  on loc {
    const numTasks = here.maxTaskPar;
    coforall tid in 1..numTasks do
      wprintf("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

```plaintext
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
", tid, numTasks, here.name);
  }
```
High-Level Task Parallelism

taskParallel.chpl

```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
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);
    }
```

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

```
prompt> chpl taskParallel.chpl -o taskParallel
prompt> ./taskParallel --numLocales=2
Hello from task 1 of 2 running on n1033
Hello from task 2 of 2 running on n1032
Hello from task 2 of 2 running on n1033
Hello from task 1 of 2 running on n1032
```
Task Parallelism, Locality Control, by example

taskParallel.chpl

```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
./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)

```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

```
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

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

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

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

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

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

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

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

● Based on positive user response to Chapel under HPCS, Cray undertook a five-year effort to improve it
  ● we’ve recently completed our third year

● Focus Areas:
  1. Improving performance and scaling
  2. Fixing immature aspects of the language and implementation
     ● e.g., strings, memory management, 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
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

Abstract: This talk aims to present my personal experiences using Chapel in my research. My research interests are in observational cosmology; more specifically, I use large surveys of galaxies to constrain the evolution of the Universe and to probe the physics underlying that evolution. Operationally, this involves measuring a number of spatial statistics of the distribution of galaxies, both on actual observations, but also on large numbers of simulated universes.

I’ll start by presenting a whirlwind introduction to cosmology, the problems that keep me up at night and our approaches to solving these. I’ll then discuss what attracted me to Chapel—the ability to prototype algorithms quickly and the promised ease and flexibility of writing parallel programs. I’ll then present a worked example of Chapel being used in a real-world application, discussing some of these aspects as well highlighting its interoperability with existing libraries, as well as some of the challenges. I’ll conclude with what it would take for me to switch over to using Chapel all of the time.
Chapel is a Work-in-Progress

- Currently being picked up by early adopters
  - Last two releases got ~3500 downloads total in a year
  - Users who try it generally like what they see

- Most core features are functional and working well
  - some areas need improvements, e.g., error-handling, constructors

- Performance varies, but is continually improving
  - shared memory performance is typically competitive with C+OpenMP
  - distributed memory performance tends to be more hit-and-miss
    - PAW workshop talk tomorrow: LLNL got 87% of reference version for CoMD

- We are actively working on addressing these lacks
Chapel-related Events at SC16

**Today:** This tutorial

**Today:** Women in HPC Workshop (all day)
- "Array initialization improvements in Chapel": Lydia Duncan (Cray)

**This evening:** CHUG (Chapel Users Group) happy hour
- 7th annual meet-up, everyone’s welcome to attend
- 5:30pm Settebello Pizzeria Napoletana

**Monday afternoon:** PGAS Applications Workshop
- "CoMD study in Chapel": Dave Richards and Riyaz Haque (LLNL)
- "ISx study in SHMEM and Chapel": Jake Hemstad (U Minn / Sandia), Ulf Hanebutte (Intel), Ben Harshbarger and Brad Chamberlain (Cray)
- "PGAS Applications panel": chaired by Brad Chamberlain (Cray)

**Wednesday:** PGAS BoF, 12:15pm

**Thursday:** Talk to a Chapel developer, PGAS booth, 10am-noon

**all week:** PGAS Booth Poster on Chapel CoMD study, Meet by Request

additional details at [http://chapel.cray.com/events.html](http://chapel.cray.com/events.html)
High-level Questions about Chapel?
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.