Chapel: Productive Parallel Programming at Scale

Elliot Ronaghan, Chapel Team, Cray Inc.
EAGE 2017
June 16th, 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.
Motivation for Chapel

Q: Why doesn’t HPC programming have an equivalent to Python / Matlab / Scala / Swift / (your favorite programming language here)?

● one that makes it easy to get programs up and running quickly
● one that is portable across system architectures and scales
● one that bridges the HPC, data analysis, and mainstream communities

A: We believe this is due not to any particular technical challenge, but rather a lack of sufficient…

…long-term efforts
…resources
…community will
…patience

Chapel is our attempt to reverse this trend!
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 far more productive
What does “Productivity” mean to you?

**Recent Graduates:**
“something similar to what I used in school: Python, Matlab, Swift, …”

**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 Chapel Team at Cray (May 2017)
Chapel Community R&D Efforts

http://chapel.cray.com/collaborations.html
High Performance Computing (HPC)
Programming Models by Example
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:**

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

```
A = = = =
B + + + +
C . . . .
```

$\alpha$
STREAM Triad: a trivial parallel computation

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

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

**In pictures, in parallel (distributed memory):**

![Diagram showing the STREAM Triad computation](attachment:image.png)
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):

```
A  =  =  =  =  =  =  =  =  =  =
B  +  +  +  +  +  +  +  +  +  +
C  .  .  .  .  .  .  .  .  .  .
α  .  .  .  .  .  .  .  .  .  .
```
m = 1000
alpha = 3.0

A = [0.0] * m
B = [2.0] * m
C = [1.0] * m

for j in range(m):
#include <hpcc.h>

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

int HPCC_StripTraped(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] = 0.0;
    }

    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>
#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] = 0.0;
    }
    scalar = 3.0;

    if (doIO) {
        #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

MPI + OpenMP

```c
#include <hpcc.h>
#ifdef _OPENMP
#include <omp.h>
#endif
static int VectorSize;
static double *a, *b, *c;
int HPCC_StarStream(HPCC_Params *params) {
    int myRank, commSize;
    int rv, errCount;
    MPI_Comm comm = MPI_COMM_WORLD;
    MPI_Comm_size( comm, &commSize );
    MPI_Comm_rank( comm, &myRank );
    rv = HPCC_Stream( params, 0 == myRank );
    MPI_Reduce( &rv, &errCount, 1, MPI_INT, MPI_SUM, 0, comm );
    return errCount;
}
int HPCC_Stream(HPCC_Params *params, int doIO) {
    register int j;
    double scalar;
    VectorSize = HPCC_LocalVectorSize( params, 3, sizeof(double), 0 );
    a = HPCC_XMALLOC( double, VectorSize );
    b = HPCC_XMALLOC( double, VectorSize );
    c = HPCC_XMALLOC( double, VectorSize );
    if (!a || !b || !c) {
        if (c) HPCC_free(c);
        if (b) HPCC_free(b);
        if (a) HPCC_free(a);
        if (doIO) {
           fprintf( outFile, "Failed to allocate memory (%d).\n", VectorSize );
            fclose( outFile );
        }
        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;
}
```

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);

    __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];
    }
}
```

Copyright 2017 Cray Inc.
STREAM Triad: MPI+OpenMP vs. CUDA

MPI + OpenMP

```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);
}
```

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

**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.

```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 mpi_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, mpi_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] = 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;
}
```
Outline

✓ Chapel Motivation and Background

➢ Chapel in a Nutshell
  ● Chapel Project: Status
  ● Chapel Resources
Multiresolution Design: Support multiple tiers of features

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

Chapel language concepts

- build the higher-level concepts in terms of the lower
- permit the user to intermix layers arbitrarily
Lower-Level Features

Chapel language concepts

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

Lower-level Chapel
Base Language Features: Fibonacci Example

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

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

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

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

Copyright 2017 Cray Inc.
Base Language Features: Fibonacci Example

**Iterators**

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

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

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

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>fib #0 is 0</td>
<td>fib #1 is 1</td>
<td>fib #2 is 1</td>
</tr>
<tr>
<td>fib #3 is 2</td>
<td>fib #4 is 3</td>
<td>fib #5 is 5</td>
</tr>
<tr>
<td>fib #6 is 8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

...
Base Language Features: Fibonacci Example

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

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

```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: Fibonacci Example

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

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

```javascript
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: Fibonacci Example

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

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

```haskell
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: Fibonacci Example

Static Type Inference for:
- arguments
- return types
- variables

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

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

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

```plaintext
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: Fibonacci Example

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

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

```javascript
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 |
...

Copyright 2017 Cray Inc.
Lower-Level Features

Chapel language concepts

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

Lower-level Chapel
Task Parallelism

```chpl
begin writeln("Hello!");
writeln("Goodbye...");
```

```
prompt> chpl beginTask.chpl -o beginTask
prompt> ./beginTask
Hello!
Goodbye...
prompt> ./beginTask
Goodbye...
Hello!
```
Task Parallelism

beginTask.chpl

begin writeln("Hello!");
writeln("Goodbye...");

prompt> chpl beginTask.chpl -o beginTask
prompt> ./beginTask
Hello!
Goodbye...
prompt> ./beginTask
Goodbye...
Hello!

Creates a new task
Lower-Level Features

Chapel language concepts

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

Lower-level Chapel
Task Parallelism & Locality Control

taskParallel.chpl

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

High-Level Task Parallelism

```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

```chapel
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

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

Control of Locality/Affinity

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

Abstraction of System Resources

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);
  }
```

```
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

High-Level Task Parallelism

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

dataParallel.chpl

```chapl
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

#### Domains (Index Sets)

```
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);
```

#### 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
```
Data Parallelism

```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);
```

Arrays

```
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

Data-Parallel Forall Loops

```chpl
cfg 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

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

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

Distributions
- BlockCycDist
- BlockDist
- CyclicDist
- DimensionalDist2D
- PrivateDist
- ReplicatedDist
- SparseBlockDist
- StencilDist

DataParallel.chpl

```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);
```

Layouts
- CSR

```
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
```
Chapel can target KNL’s MCDRAM via on-clauses

```chapel
on here.highBandwidthMemory() {
    x = new myClass();   // placed in MCDRAM
    ...
    on here.defaultMemory() {
        y = new myClass();   // placed in DDR
        ...
    }
}

on y.locale.highBandwidthMemory() {
    z = new myClass();   // same locale as y, but using MCDRAM
    ...
}
```
Intel Xeon Phi ("KNL") locale model

- Working towards distributions that can target MCDRAM

```chapel
config const m = 1000,
        alpha = 3.0;

const ProblemSpace = {1..m}
        dmapped KNLDist(...);

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

B = 2.0;
C = 1.0;
A = B + alpha * C;
```
Outline

✓ Chapel Motivation and Background
✓ Chapel in a Nutshell
➢ Chapel Project: Status
● Chapel Resources
Chapel is a Work-in-Progress

- **Currently being picked up by early adopters**
  - Users who try it like what they see

- **Most features are functional and working well**
  - some areas are under active development, particularly:
    - Initializers
    - Error handling

- **Performance is improving**
  - shared memory performance is competitive with C+OpenMP
  - some distributed memory applications now competitive with C+MPI
    - many cases continue to need more work
Single-locale Performance
Overall, single-locale performance improved dramatically
LCALS: Serial Timings, Chapel 1.13.0

Long problem size
(Similar results for medium and short problem sizes)

Normalized time – serial reference is 1.0

Serial Chapel vs g++

g++ serial
Chapel serial

chpl --fast
--no-ieee-float

better

g++ -Ofast -fopenmp
LCALS: Serial Timings, Chapel 1.14.0

Long problem size
(Similar results for medium and short problem sizes)

Serial Chapel vs g++

Normalized time – serial reference is 1.0

chpl --fast
--no-ieee-float

g++ -Ofast -fopenmp
LCALS: Parallel Timings, Chapel 1.14.0

- Parallel variants still lagged behind the reference in 1.14
  - between 1.5X and 8X slower for long problem size

Parallel Chapel vs OMP

- Normalized time – parallel reference is 1.0
- g++ OMP vs Chapel parallel
LCALS: Parallel Timings, Chapel 1.15.0

- Chapel 1.15 closed the gap significantly
  - ~3-4x speedup: on par or very close to reference for most kernels

Parallel Chapel vs g++/OMP

Normalized time – parallel reference is 1.0

Chapel 1.15 closed the gap significantly
- ~3-4x speedup: on par or very close to reference for most kernels
Multi-Locale Performance
Multi-locale Performance

- Significant multi-locale performance improvements

---

**ISx (weakIO) --n=5592400**

**miniMD --size 20 (sec)**

**SSCA2 Size 22, 2^4 Vertices, Time**

**DOE: Lulesh Dense Time (sec) sedov15oct**
Multi-locale Performance

- Significant multi-locale performance improvements

**HPCC FFT Perf (Gflop/s) n=2^{20}**

- Gflop/s
- Release
- ugni-threads
- ugni-mixed
- gn-aries
- gn-mpi

**HPCC HPL Release Perf (Gflop/s) n=255, nb=32**

- Gflop/s
- Release
- ugni-threads
- ugni-mixed
- gn-aries
- gn-mpi

**HPCC: PTRANS Perf (GB/sec) n=2,000, nb=100**

- GB/s
- Release
- ugni-threads
- ugni-mixed
- gn-aries
- gn-mpi

**HPCC: RA-atomics Perf (GUPS) n=2^{33}, N_U=10M**

- GUPS
- Release
- ugni-threads
- ugni-mixed
- gn-aries
- gn-mpi
64 nodes on Cray XC

ISx Execution Time: MPI, SHMEM, Chapel 1.15

- ISx weakISO Total Time

- Better performance with Chapel 1.15 compared to MPI and SHMEM

- Time (seconds) vs. Nodes (x 36 cores per node)
RA Performance: Chapel vs. MPI

Performance of RA (atomics)

GUP/s

Locales (x 36 cores per locale)

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

better
Performance: Summary

Summary:
- performance has been dramatically improving
- shared memory performance is competitive with C+OpenMP
- some distributed memory applications now competitive with C+MPI
  - many applications need more work

Next steps:
- Multi-locale:
  - benchmark-driven performance and scalability improvements
    - particularly for DOE proxy apps, stencils codes, PRK benchmarks, and others
- Single-locale:
  - vectorization
Our #1 Challenge

- How to grow the user and developer communities?
- How to encourage people to look at Chapel again?
  - overcome impressions made in our young, awkward years…

‘Scientific computing communities are very wary of new technologies (it took 10+ years for Python to start getting any traction), with the usual, self-fulfilling, fear being “what if it goes away?”’
- Jonathan Dursi, from Should I Use Chapel or Julia for my next project?
This Talk’s Takeaways

● Chapel is a modern and productive parallel language
  ● Well suited for current and emerging HPC and commodity systems
  ● Ready for early adopters now
  ● Performance and stability are increasing with each release

‘There are other research projects in this area - productive, performant, parallel computing languages for distributed-memory scientific computing.

But Chapel, especially now with 1.15, is a mature product.

It is crossing the barrier of Fast Enough'

- Jonathan Dursi, from Chapel’s Home in the New Landscape of Scientific Frameworks
Outline

✓ Chapel Motivation and Background
✓ Chapel in a Nutshell
✓ Chapel Project: Status
➢ Chapel Resources
Chapel Websites

Project page: http://chapel.cray.com
● overview, papers, presentations, language spec, …

GitHub: https://github.com/chapel-lang
● download Chapel; browse source repository; contribute code

Facebook: https://www.facebook.com/ChapelLanguage

Twitter: https://twitter.com/ChapelLanguage
Suggested Reading

Chapel chapter from *Programming Models for Parallel Computing*
- a detailed overview of Chapel’s history, motivating themes, features
- chapter is now also available online

Other Chapel papers/publications available at [http://chapel.cray.com/papers.html](http://chapel.cray.com/papers.html)
Chapel Blog Articles

- a short-and-sweet introduction to Chapel

**Chapel Springs into a Summer of Code**, Cray Blog, April 2016.
- a run-down of some current events

**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

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

low-traffic (read-only):
chapel-announce@lists.sourceforge.net: announcements about Chapel

community lists:
chapel-users@lists.sourceforge.net: user-oriented discussion list
chapel-developers@lists.sourceforge.net: developer discussions
chapel-education@lists.sourceforge.net: educator discussions

(subscribe at SourceForge: http://sourceforge.net/p/chapel/mailman/)}

To contact the Cray team:
chapel_info@cray.com: contact the team at Cray
chapel_bugs@cray.com: for reporting non-public bugs
Other Community Resources

IRC channels (freenode.net):
- #chapel: user-oriented discussions
- #chapel-developers: developer discussions

Stack Overflow
- stackoverflow.com: [chapel] tag monitored by core team

GitHub Issues:
- github.com/chapel-lang/chapel/issues: bug reports & feature requests
Chapel: Productive Parallel Programming at Scale

Questions?
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.