Chapel: A Parallel Language for Productivity at Scale

National Institute of Standards and Technology

October 11, 2019

chapel_info@cray.com

chapel-lang.org

@ChapelLanguage

CRAY®

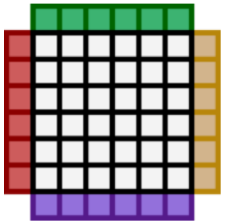
a Hewlett Packard Enterprise company

CRAY CHAPEL

Who am I?

Education:

- Earned Ph.D. from University of Washington CSE in 2001
  - focused on the ZPL data-parallel array language
- Remain associated with UW CSE as an Affiliate Professor



Industry:

a Hewlett Packard Enterprise company

- A Principal Engineer at Cray
- The technical lead / a founding member of the Chapel project

© 2019 Cray, a Hewlett Packard Enterprise company

What is Chapel?

CRAY
a Hewlett Packard Enterprise company

Chapel: A modern parallel programming language

- portable & scalable
- open-source & collaborative

Goals:

- Support general parallel programming
  - “any parallel algorithm on any parallel hardware”
- Make parallel programming at scale far more productive

© 2019 Cray, a Hewlett Packard Enterprise company

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.”

© 2019 Cray, a Hewlett Packard Enterprise company

Why Consider New Languages at all?

Syntax	• High level, elegant syntax • Improve programmer productivity
Semantics	• Static analysis can help with correctness • We need a compiler (front-end)
Performance	• If optimizations are needed to get performance • We need a compiler (back-end)
Algorithms	• Language defines what is easy and hard • Influences algorithmic thinking

[Source: Kathy Yelick, CHIUW 2018 keynote: *Why Languages Matter More Than Ever*]

© 2019 Cray, a Hewlett Packard Enterprise company

Outline

✓ Context and Motivation

➢ Chapel and Productivity

- A Brief Tour of Chapel Features
- Recent Uses of Chapel
- Summary and Resources

© 2019 Cray, a Hewlett Packard Enterprise company

Comparing Chapel to Other Languages

Chapel aims to be as…

- …**programmable** as Python
- …**fast** as Fortran
- …**scalable** as MPI, SHMEM, or UPC
- …**portable** as C
- …**flexible** as C++
- …**fun** as [your favorite programming language]

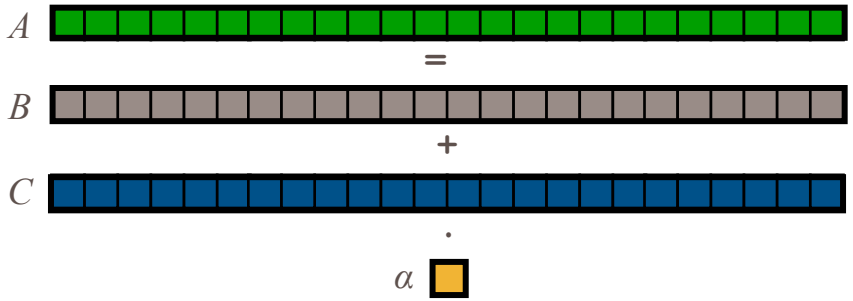
© 2019 Cray, a Hewlett Packard Enterprise company

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:



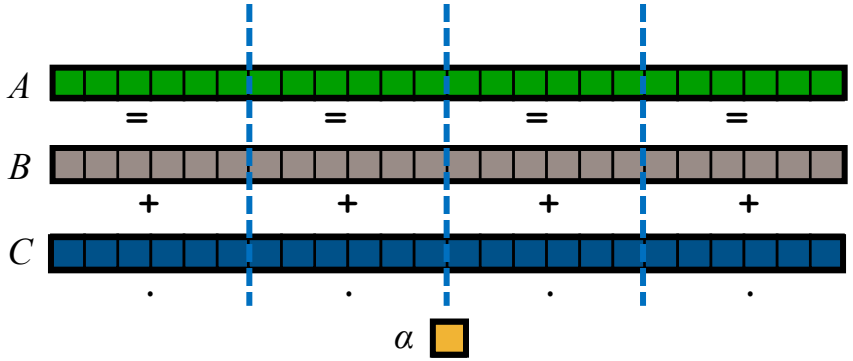
© 2019 Cray, a Hewlett Packard Enterprise company

STREAM Triad: a trivial parallel computation

Given: m-element vectors A, B, C

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

In pictures, in parallel (shared memory / multicore):



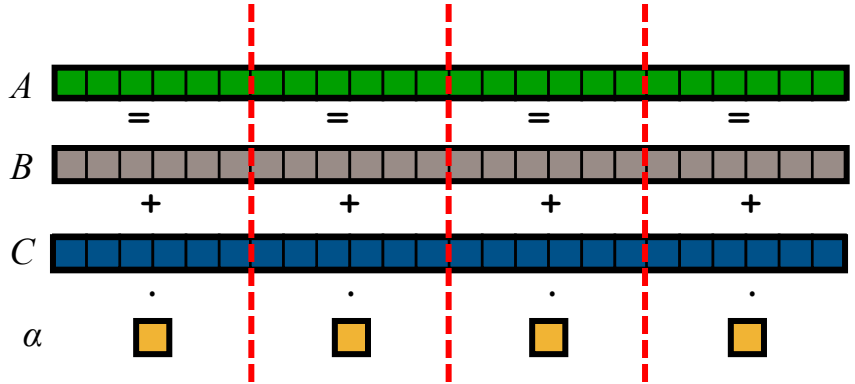
© 2019 Cray, a Hewlett Packard Enterprise company

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):



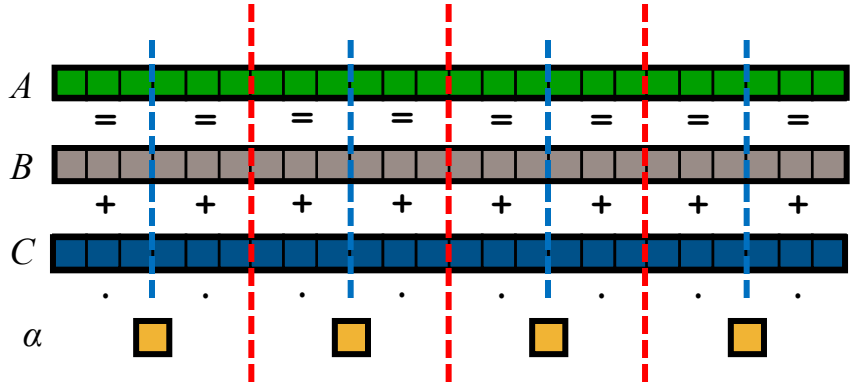
© 2019 Cray, a Hewlett Packard Enterprise company

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):



© 2019 Cray, a Hewlett Packard Enterprise company

STREAM Triad: C + MPI

```
#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);

  return 0;
}
```

© 2019 Cray, a Hewlett Packard Enterprise company

STREAM Triad: C + MPI + OpenMP

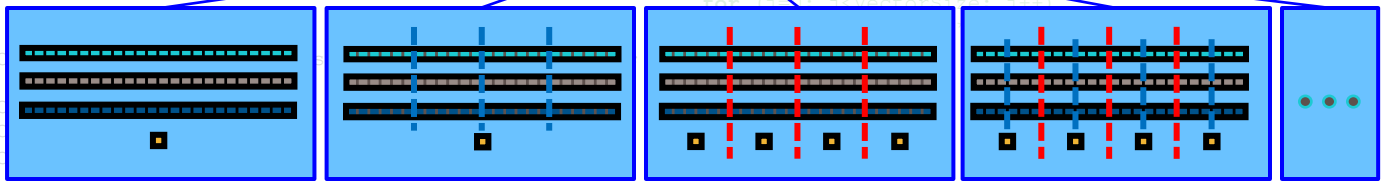
```c
#include <hpcc.h>
#ifdef _OPENMP
#include <omp.h>
#endif

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

  return errCount;
}

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

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

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

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

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

#ifdef _OPENMP
#pragma omp parallel for
#endif
  for (j=0; j<VectorSize; j++)
    a[j] = b[j]+scalar*c[j];

  HPCC_free(c);
  HPCC_free(b);
  HPCC_free(a);

  return 0;
}
```

© 2019 Cray, a Hewlett Packard Enterprise company

STREAM Triad: Chapel

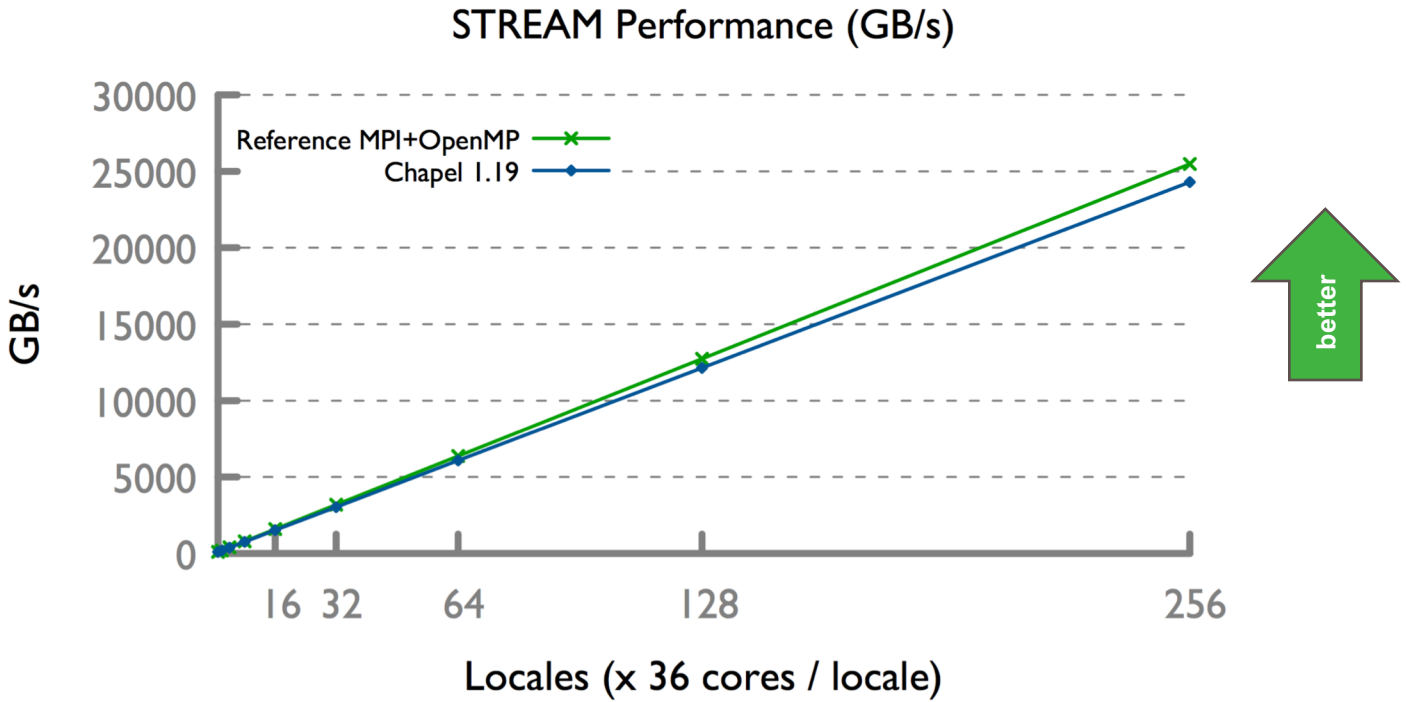
```
use …;

config const m = 1000,
             alpha = 3.0;

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

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

B = 2.0;
C = 1.0;

A = B + alpha * C;
```

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



© 2019 Cray, a Hewlett Packard Enterprise company

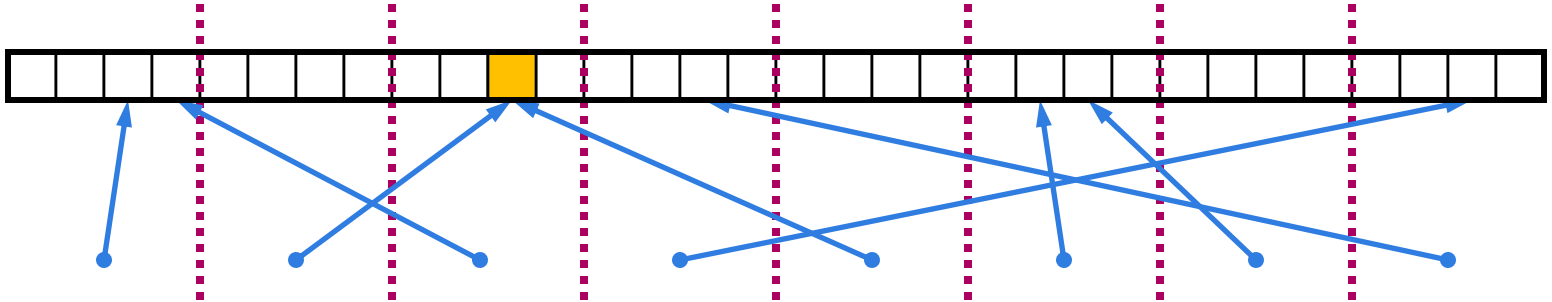
HPCC STREAM Triad: Chapel vs. C+MPI+OpenMP



© 2019 Cray, a Hewlett Packard Enterprise company

HPCC Random Access (RA)

Data Structure: distributed table



Computation: update random table locations in parallel

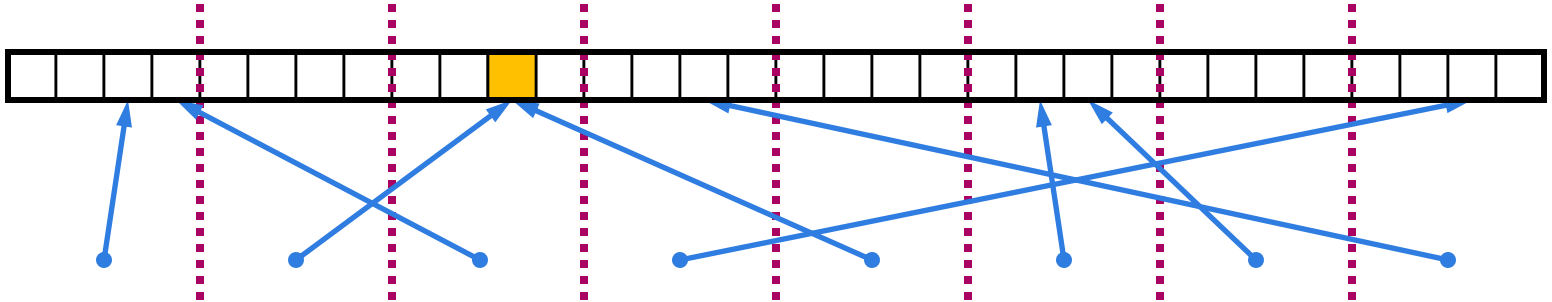
Two variations:

- **lossless:** don't allow any updates to be lost
- **lossy:** permit some fraction of updates to be lost

© 2019 Cray, a Hewlett Packard Enterprise company

HPCC Random Access (RA)

Data Structure: distributed table



Computation: update random table locations in parallel

Two variations:

- **lossless:** don't allow any updates to be lost
- **lossy:** permit some fraction of updates to be lost

© 2019 Cray, a Hewlett Packard Enterprise company

HPCC RA: MPI kernel

```
/* Perform updates to main table.  The scalar equivalent is:
 *
 *    for (i=0; i<NUPDATE; i++) {
 *      Ran = (Ran << 1) ^ (((s64Int) Ran < 0) ? POLY : 0);
 *      Table[Ran & (TABSIZE-1)] ^= Ran;
 *    }
 */

MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
          MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
 while (i < SendCnt) {
   /* receive messages */
   do {
     MPI_Test(&inreq, &have_done, &status);
     if (have_done) {
       if (status.MPI_TAG == UPDATE_TAG) {
         MPI_Get_count(&status, tparams.dtype64, &recvUpdates);
         bufferBase = 0;
         for (j=0; j < recvUpdates; j ++) {
           inmsg = LocalRecvBuffer[bufferBase+j];
           LocalOffset = (inmsg & (tparams.TableSize - 1)) -
                         tparams.GlobalStartMyProc;
           HPCC_Table[LocalOffset] ^= inmsg;
         }
       } else if (status.MPI_TAG == FINISHED_TAG) {
         NumberReceiving--;
       } else
         MPI_Abort( MPI_COMM_WORLD, -1 );
       MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
                 MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
     }
   } while (have_done && NumberReceiving > 0);
   if (pendingUpdates < maxPendingUpdates) {
     Ran = (Ran << 1) ^ ((s64Int) Ran < ZERO64B ? POLY : ZERO64B);
     GlobalOffset = Ran & (tparams.TableSize-1);
     if ( GlobalOffset < tparams.Top)
       WhichPe = ( GlobalOffset / (tparams.MinLocalTableSize + 1) );
     else
       WhichPe = ( (GlobalOffset - tparams.Remainder) /
                 tparams.MinLocalTableSize );
     if (WhichPe == tparams.MyProc) {
       LocalOffset = (Ran & (tparams.TableSize - 1)) -
                     tparams.GlobalStartMyProc;
       HPCC_Table[LocalOffset] ^= Ran;
     } else {
       HPCC_InsertUpdate(Ran, WhichPe, Buckets);
       pendingUpdates++;
     }
     i++;
   }
   else {
     MPI_Test(&outreq, &have_done, MPI_STATUS_IGNORE);
     if (have_done) {
       outreq = MPI_REQUEST_NULL;
       pe = HPCC_GetUpdates(Buckets, LocalSendBuffer, localBufferSize,
                            &peUpdates);
       MPI_Isend(&LocalSendBuffer, peUpdates, tparams.dtype64, (int)pe,
                 UPDATE_TAG, MPI_COMM_WORLD, &outreq);
       pendingUpdates -= peUpdates;
     }
   }
 }
 /* send remaining updates in buckets */
 while (pendingUpdates > 0) {
   /* receive messages */
   do {
     MPI_Test(&inreq, &have_done, &status);
     if (have_done) {
       if (status.MPI_TAG == UPDATE_TAG) {
         MPI_Get_count(&status, tparams.dtype64, &recvUpdates);
         bufferBase = 0;
         for (j=0; j < recvUpdates; j ++) {
           inmsg = LocalRecvBuffer[bufferBase+j];
           LocalOffset = (inmsg & (tparams.TableSize - 1)) -
                         tparams.GlobalStartMyProc;
           HPCC_Table[LocalOffset] ^= inmsg;
         }
       } else if (status.MPI_TAG == FINISHED_TAG) {
         /* we got a done message.  Thanks for playing... */
         NumberReceiving--;
       } else {
         MPI_Abort( MPI_COMM_WORLD, -1 );
       }
       MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
                 MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
     }
   } while (have_done && NumberReceiving > 0);
   MPI_Test(&outreq, &have_done, MPI_STATUS_IGNORE);
   if (have_done) {
     outreq = MPI_REQUEST_NULL;
     pe = HPCC_GetUpdates(Buckets, LocalSendBuffer, localBufferSize,
                          &peUpdates);
     MPI_Isend(&LocalSendBuffer, peUpdates, tparams.dtype64, (int)pe,
               UPDATE_TAG, MPI_COMM_WORLD, &outreq);
     pendingUpdates -= peUpdates;
   }
 }
 /* send our done messages */
 for (proc_count = 0 ; proc_count < tparams.NumProcs ; ++proc_count) {
   if (proc_count == tparams.MyProc) { tparams.finish_req[tparams.MyProc] =
                                       MPI_REQUEST_NULL; continue; }
   /* send garbage - who cares, no one will look at it */
   MPI_Isend(&Ran, 0, tparams.dtype64, proc_count, FINISHED_TAG,
             MPI_COMM_WORLD, tparams.finish_req + proc_count);
 }
 /* Finish everyone else up... */
 while (NumberReceiving > 0) {
   MPI_Wait(&inreq, &status);
   if (status.MPI_TAG == UPDATE_TAG) {
     MPI_Get_count(&status, tparams.dtype64, &recvUpdates);
     bufferBase = 0;
     for (j=0; j < recvUpdates; j ++) {
       inmsg = LocalRecvBuffer[bufferBase+j];
       LocalOffset = (inmsg & (tparams.TableSize - 1)) -
                     tparams.GlobalStartMyProc;
       HPCC_Table[LocalOffset] ^= inmsg;
     }
   } else if (status.MPI_TAG == FINISHED_TAG) {
     /* we got a done message.  Thanks for playing... */
     NumberReceiving--;
   } else {
     MPI_Abort( MPI_COMM_WORLD, -1 );
   }
   MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
             MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
 }

 MPI_Waitall( tparams.NumProcs, tparams.finish_req, tparams.finish_statuses);
```

© 2019 Cray, a Hewlett Packard Enterprise company

HPCC RA: MPI kernel comment vs. Chapel

Chapel Kernel

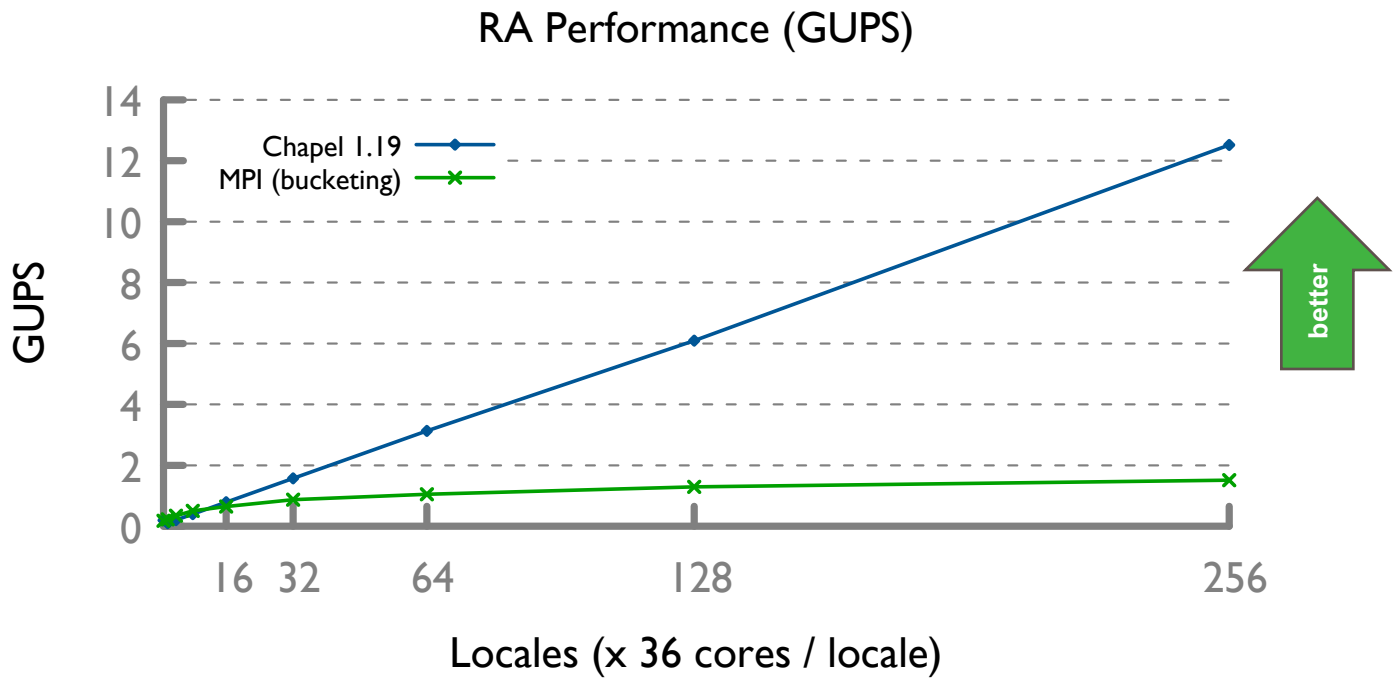
```
forall (_, r) in zip(Updates, RAStream()) do
  T[r & indexMask].xor(r);
```

MPI Comment

```
/* Perform updates to main table. The scalar equivalent is:
 *
 *      for (i=0; i<NUPDATE; i++) {
 *        Ran = (Ran << 1) ^ (((s64Int) Ran < 0) ? POLY : 0);
 *        Table[Ran & (TABSIZE-1)] ^= Ran;
 *      }
 */
```

© 2019 Cray, a Hewlett Packard Enterprise company

HPCC RA: Chapel vs. C+MPI



© 2019 Cray, a Hewlett Packard Enterprise company

HPCC RA: MPI vs. Chapel

Chapel Kernel

```
forall (_, r) in zip(Updates, RAStream()) do
  T[r & indexMask].xor(r);
```

© 2019 Cray, a Hewlett Packard Enterprise company

HPCC RA: MPI vs. Chapel

Chapel Kernel

```
forall (_, r) in zip(Updates, RAStream()) do
  T[r & indexMask].xor(r);
```

```
/* Perform updates to main table.  The scalar equivalent is:
 *
 *   for (i=0; i<NUPDATE; i++) {
 *     Ran = (Ran << 1) ^ (((s64Int) Ran < 0) ? POLY : 0);
 *     Table[Ran & (TABSIZE-1)] ^= Ran;
 *   }
 */

MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dty
          MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD,
 while (i < SendCnt) {
   /* receive messages */
   do {
     MPI_Test(&inreq, &have_done, &status);
     if (have_done) {
       if (status.MPI_TAG == UPDATE_TAG) {
         MPI_Get_count(&status, tparams.dtype64, &recvUpdates);
         bufferBase = 0;
         for (j=0; j < recvUpdates; j ++) {
           inmsg = LocalRecvBuffer[bufferBase+j];
           LocalOffset = (inmsg & (tparams.TableSize - 1)) -
                         tparams.GlobalStartMyProc;
           HPCC_Table[LocalOffset] ^= inmsg;
         }
       } else if (status.MPI_TAG == FINISHED_TAG) {
         NumberReceiving--;
       } else
         MPI_Abort( MPI_COMM_WORLD, -1 );
       MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
                 MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
     }
   } while (have_done && NumberReceiving > 0);
   if (pendingUpdates < maxPendingUpdates) {
     Ran = (Ran << 1) ^ ((s64Int) Ran < ZERO64B ? POLY : ZERO64B);
     GlobalOffset = Ran & (tparams.TableSize-1);
     if ( GlobalOffset < tparams.Top)
       WhichPe = ( GlobalOffset / (tparams.MinLocalTableSize + 1) );
     else
       WhichPe = ( (GlobalOffset - tparams.Remainder) /
                   tparams.MinLocalTableSize );
     if (WhichPe == tparams.MyProc) {
       LocalOffset = (Ran & (tparams.TableSize - 1)) -
                     tparams.GlobalStartMyProc;
       HPCC_Table[LocalOffset] ^= Ran;
```

```
   } else {
     HPCC_InsertUpdate(Ran, WhichPe, Buckets);
                                        peUpdates);
     MPI_Isend(&LocalSendBuffer, peUpdates, tparams.dtype64, (int)pe,
               UPDATE_TAG, MPI_COMM_WORLD, &outreq);
     pendingUpdates -= peUpdates;
   }
 }
}
/* send remaining updates in buckets */
while (pendingUpdates > 0) {
  /* receive messages */
  do {
    MPI_Test(&inreq, &have_done, &status);
    if (have_done) {
      if (status.MPI_TAG == UPDATE_TAG) {
        MPI_Get_count(&status, tparams.dtype64, &recvUpdates);
        bufferBase = 0;
        for (j=0; j < recvUpdates; j ++) {
          inmsg = LocalRecvBuffer[bufferBase+j];
          LocalOffset = (inmsg & (tparams.TableSize - 1)) -
                        tparams.GlobalStartMyProc;
          HPCC_Table[LocalOffset] ^= inmsg;
        }
      } else if (status.MPI_TAG == FINISHED_TAG) {
        /* we got a done message.  Thanks for playing... */
        NumberReceiving--;
      } else {
        MPI_Abort( MPI_COMM_WORLD, -1 );
      }
      MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
                MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
    }
  } while (have_done && NumberReceiving > 0);
```

```
                                                          e,
                                                          pe,
}
/* send our done messages */
for (proc_count = 0 ; proc_count < tparams.NumProcs ; ++proc_count) {
  if (proc_count == tparams.MyProc) { tparams.finish_req[tparams.MyProc] =
                                      MPI_REQUEST_NULL; continue; }
  /* send garbage - who cares, no one will look at it */
  MPI_Isend(&Ran, 0, tparams.dtype64, proc_count, FINISHED_TAG,
            MPI_COMM_WORLD, tparams.finish_req + proc_count);
}
/* Finish everyone else up... */
while (NumberReceiving > 0) {
  MPI_Wait(&inreq, &status);
  if (status.MPI_TAG == UPDATE_TAG) {
    MPI_Get_count(&status, tparams.dtype64, &recvUpdates);
    bufferBase = 0;
    for (j=0; j < recvUpdates; j ++) {
      inmsg = LocalRecvBuffer[bufferBase+j];
      LocalOffset = (inmsg & (tparams.TableSize - 1)) -
                    tparams.GlobalStartMyProc;
      HPCC_Table[LocalOffset] ^= inmsg;
    }
  } else if (status.MPI_TAG == FINISHED_TAG) {
    /* we got a done message.  Thanks for playing... */
    NumberReceiving--;
  } else {
    MPI_Abort( MPI_COMM_WORLD, -1 );
  }
  MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
            MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
}

MPI_Waitall( tparams.NumProcs, tparams.finish_req, tparams.finish_statuses);
```

© 2019 Cray, a Hewlett Packard Enterprise company

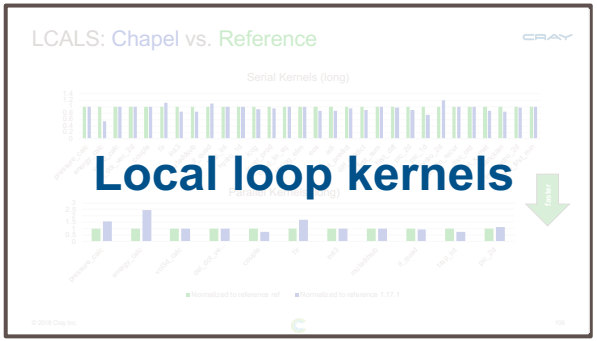
Why Consider New Languages at all?

Syntax	• High level, elegant syntax • Improve programmer productivity
Semantics	• Static analysis can help with correctness • We need a compiler (front-end)
Performance	• If optimizations are needed to get performance • We need a compiler (back-end)
Algorithms	• Language defines what is easy and hard • Influences algorithmic thinking

[Source: Kathy Yelick, CHIUW 2018 keynote: *Why Languages Matter More Than Ever*]

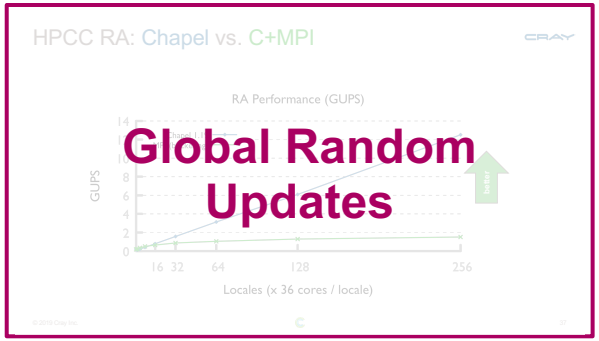
© 2019 Cray, a Hewlett Packard Enterprise company

HPC Patterns: Chapel vs. Reference



LCALS

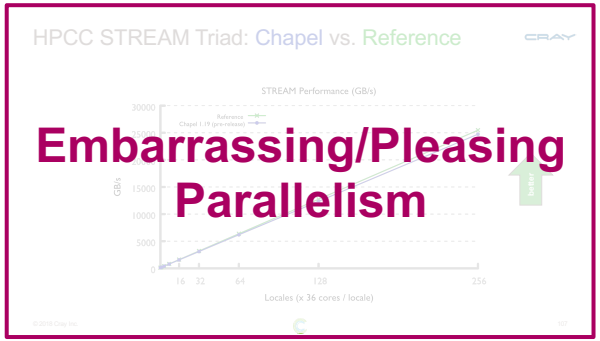
HPCC RA

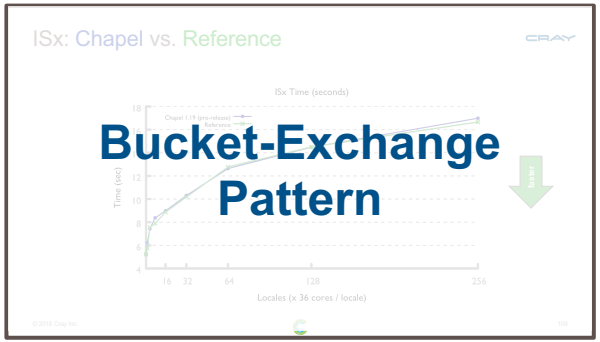


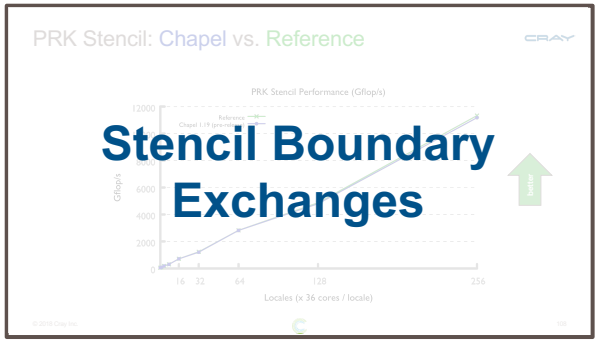
STREAM Triad

ISx

PRK Stencil



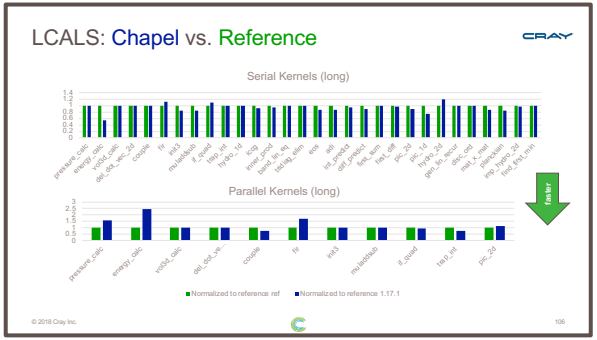




Nightly performance tickers online at:
https://chapel-lang.org/perf-nightly.html

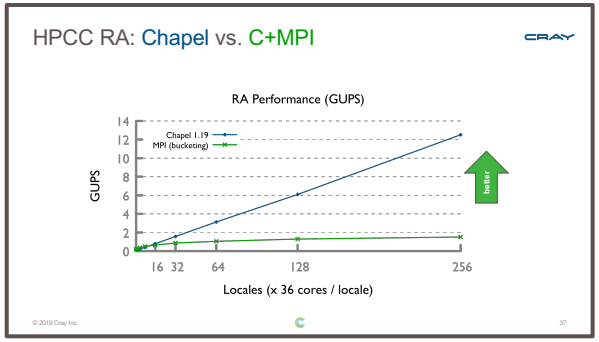
© 2019 Cray, a Hewlett Packard Enterprise company

HPC Patterns: Chapel vs. Reference



LCALS

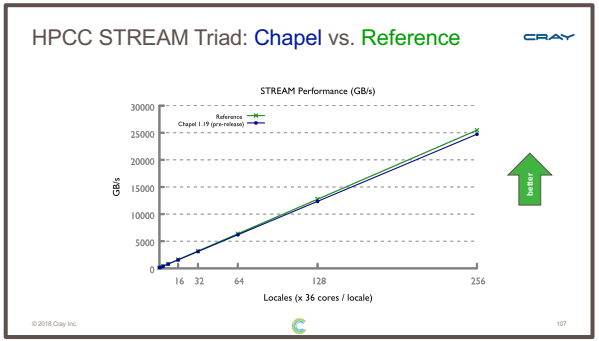
HPCC RA

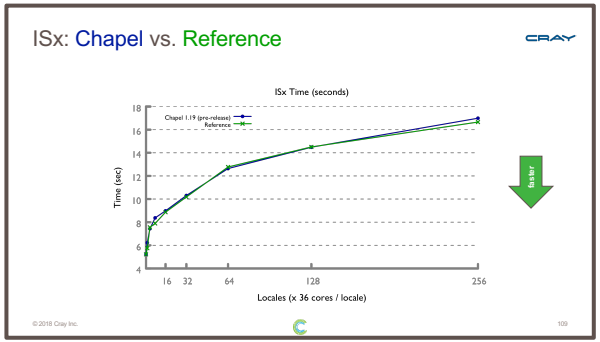


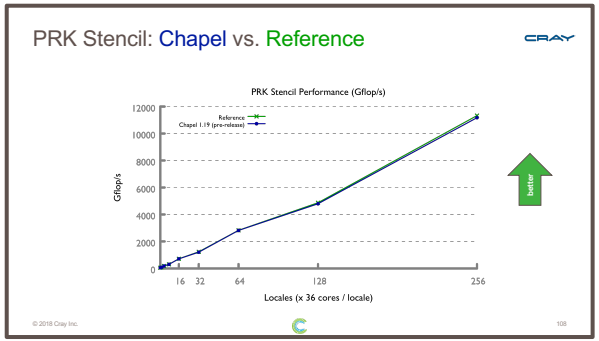
STREAM Triad

ISx

PRK Stencil







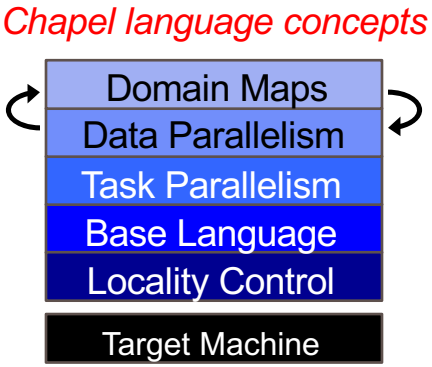
Nightly performance tickers online at:
https://chapel-lang.org/perf-nightly.html

© 2019 Cray, a Hewlett Packard Enterprise company

A Brief Tour of Chapel Features

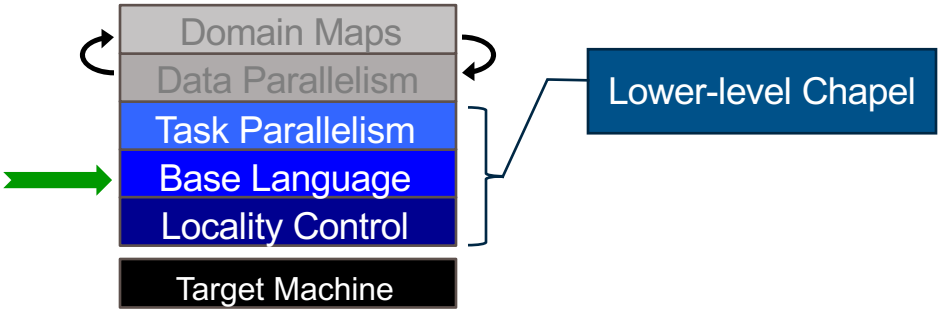
© 2019 Cray, a Hewlett Packard Enterprise company

Chapel Feature Areas



© 2019 Cray, a Hewlett Packard Enterprise company

Base Language



© 2019 Cray, a Hewlett Packard Enterprise company

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
…
```

© 2019 Cray, a Hewlett Packard Enterprise company

Base Language Features, by example

Configuration declarations
(support command-line overrides)
`./fib --n=1000000`

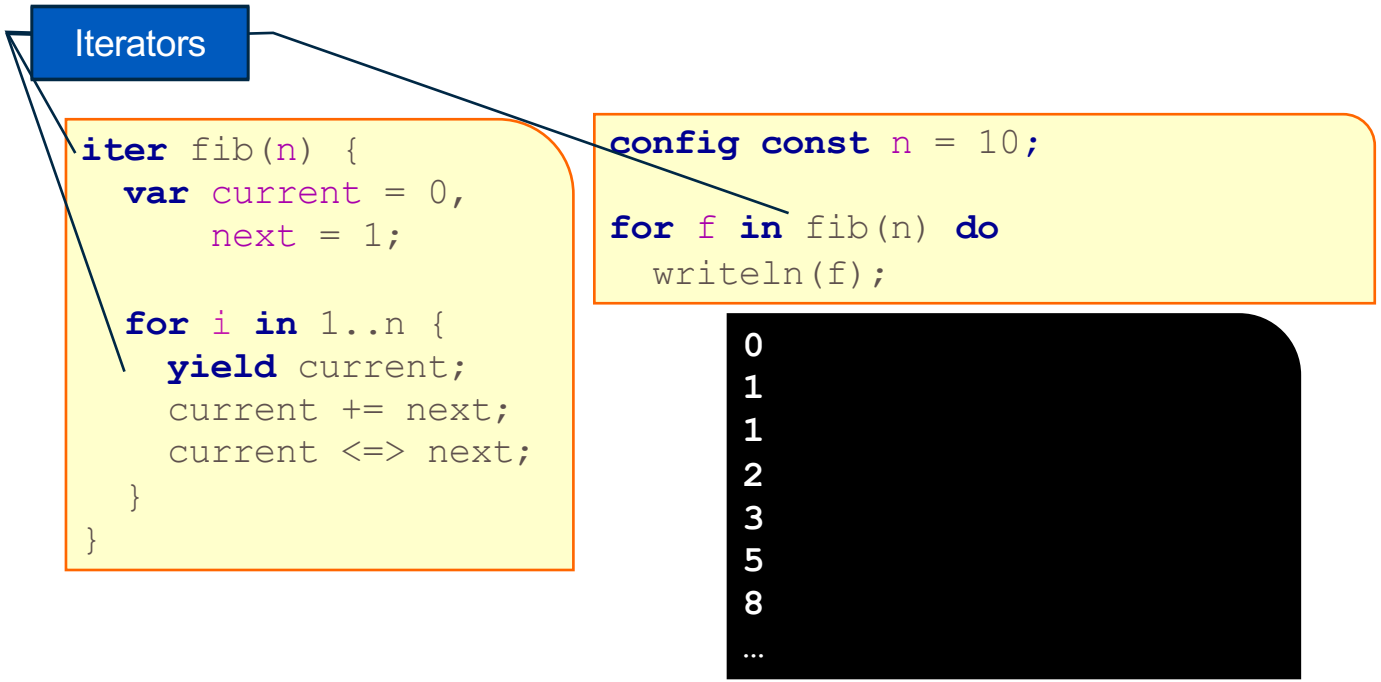
```
iter fib(n) {
  var current = 0,
      next = 1;

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

```
config const n = 10;

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

```
0
1
1
2
3
5
8
…
```

© 2019 Cray, a Hewlett Packard Enterprise company

Base Language Features, by example



© 2019 Cray, a Hewlett Packard Enterprise company

Base Language Features, by example

Static type inference for:
- arguments
- return types
- variables

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

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

```
config const n = 10;

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

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

© 2019 Cray, a Hewlett Packard Enterprise company

Base Language Features, by example


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

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

```
config const n: int = 10;

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

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

© 2019 Cray, a Hewlett Packard Enterprise company

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
…
```

© 2019 Cray, a Hewlett Packard Enterprise company

Base Language Features, by example

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

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

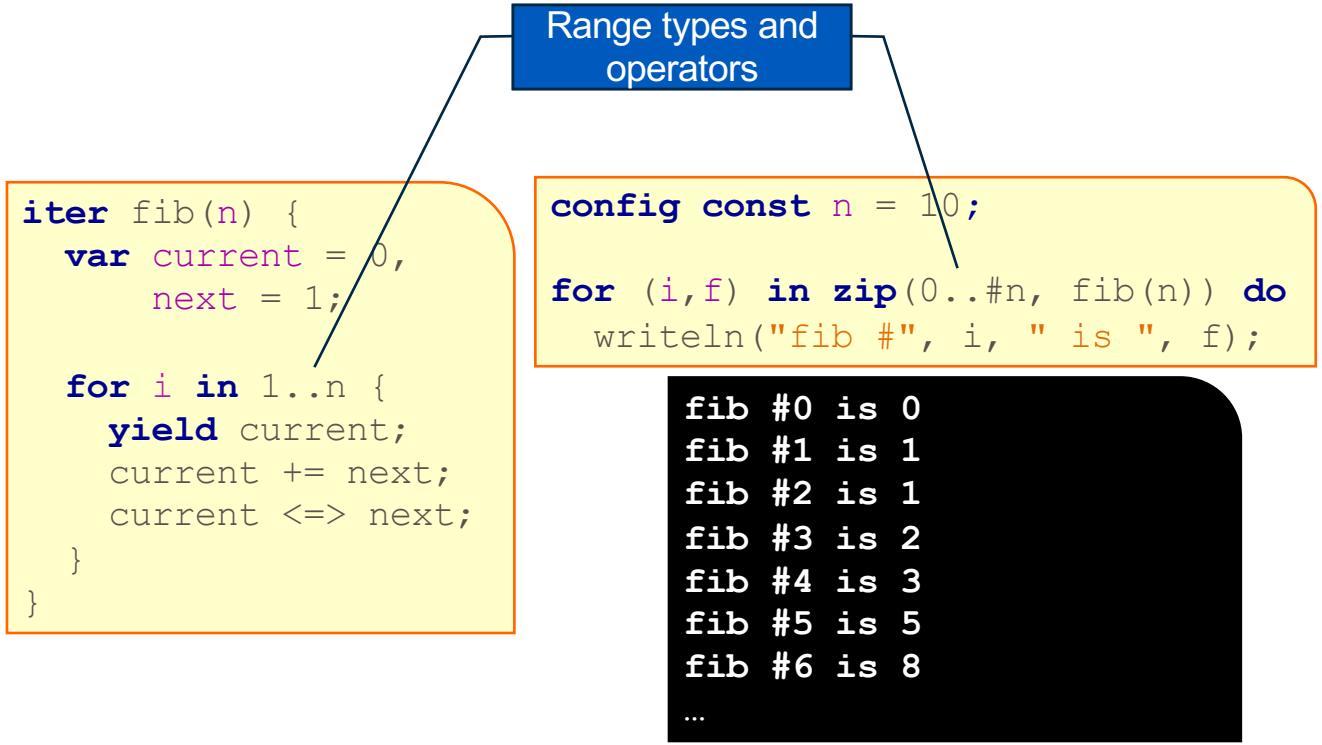
Zippered iteration

```
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
…
```

© 2019 Cray, a Hewlett Packard Enterprise company

Base Language Features, by example



© 2019 Cray, a Hewlett Packard Enterprise company

Base Language Features, by example

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

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

Tuples

```
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
…
```

© 2019 Cray, a Hewlett Packard Enterprise company

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
…
```

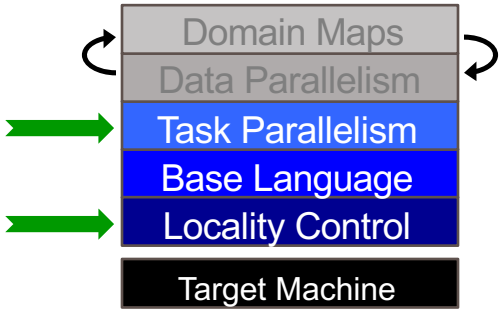
© 2019 Cray, a Hewlett Packard Enterprise company

Other Base Language Features

- **Object-oriented programming** (value- and reference-based)
  - Managed objects and lifetime checking
  - Nilable vs. non-nilable class variables
- **Generic programming / polymorphism**
- **Error-handling**
- **Compile-time meta-programming**
- **Modules** (supporting namespaces)
- **Procedure overloading / filtering**
- **Arguments:** default values, intents, name-based matching, type queries
- and more…

© 2019 Cray, a Hewlett Packard Enterprise company

Task Parallelism and Locality Control

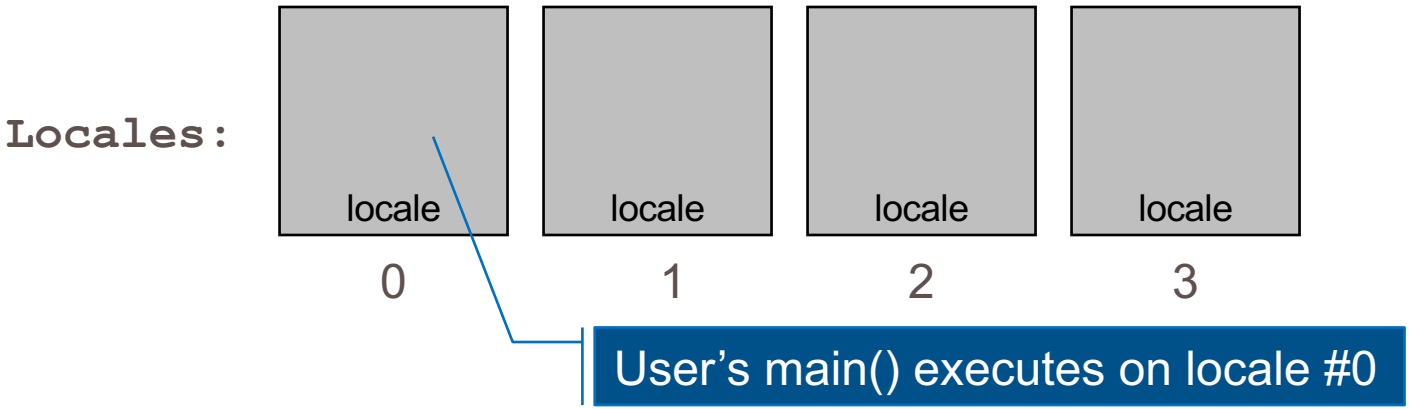


© 2019 Cray, a Hewlett Packard Enterprise company

Locales, briefly

- Locales can run tasks and store variables
  - Think “compute node”
  - Number of locales specified on execution command-line

```
> ./myProgram --numLocales=4     # or `-nl 4`
```



© 2019 Cray, a Hewlett Packard Enterprise company

Task Parallelism and Locality, by example

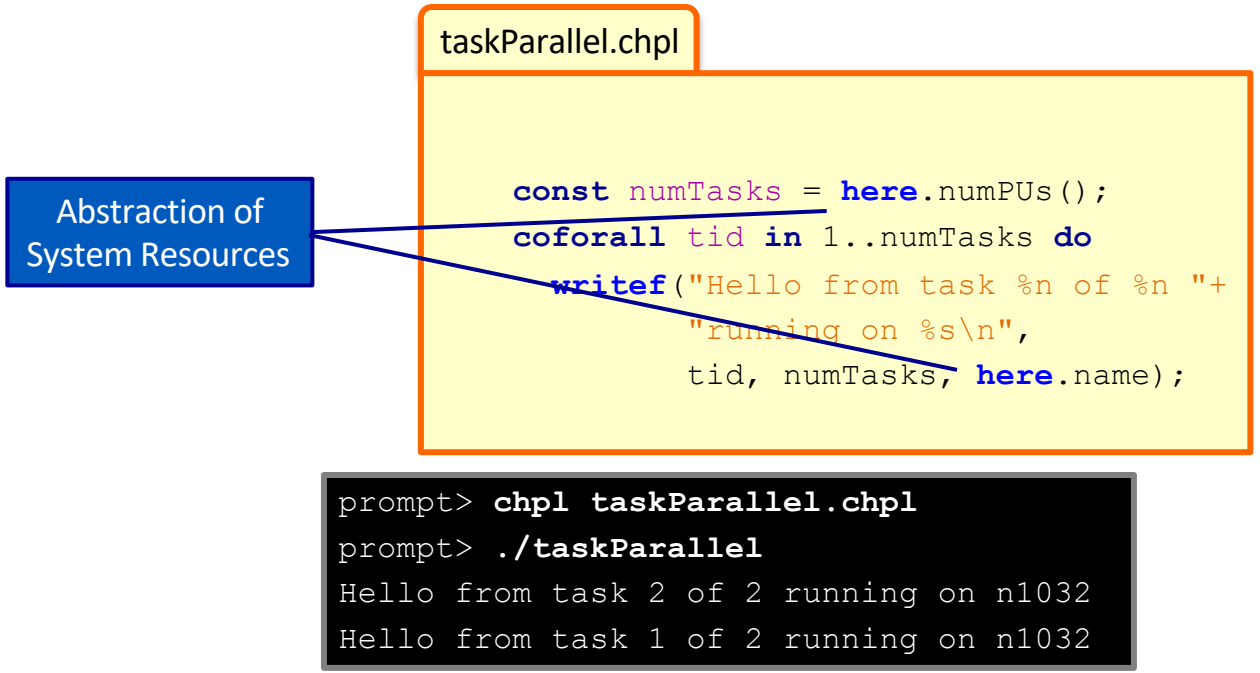
taskParallel.chpl

```
const numTasks = here.numPUs();
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
prompt> ./taskParallel
Hello from task 2 of 2 running on n1032
Hello from task 1 of 2 running on n1032
```

© 2019 Cray, a Hewlett Packard Enterprise company

Task Parallelism and Locality, by example



© 2019 Cray, a Hewlett Packard Enterprise company

Task Parallelism and Locality, by example

High-Level Task Parallelism

taskParallel.chpl

```
const numTasks = here.numPUs();
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
prompt> ./taskParallel
Hello from task 2 of 2 running on n1032
Hello from task 1 of 2 running on n1032
```

© 2019 Cray, a Hewlett Packard Enterprise company

Task Parallelism and Locality, by example

So far, this is a shared memory program

Nothing refers to remote locales, explicitly or implicitly

taskParallel.chpl

```
const numTasks = here.numPUs();
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
prompt> ./taskParallel
Hello from task 2 of 2 running on n1032
Hello from task 1 of 2 running on n1032
```

© 2019 Cray, a Hewlett Packard Enterprise company

Task Parallelism and Locality, by example

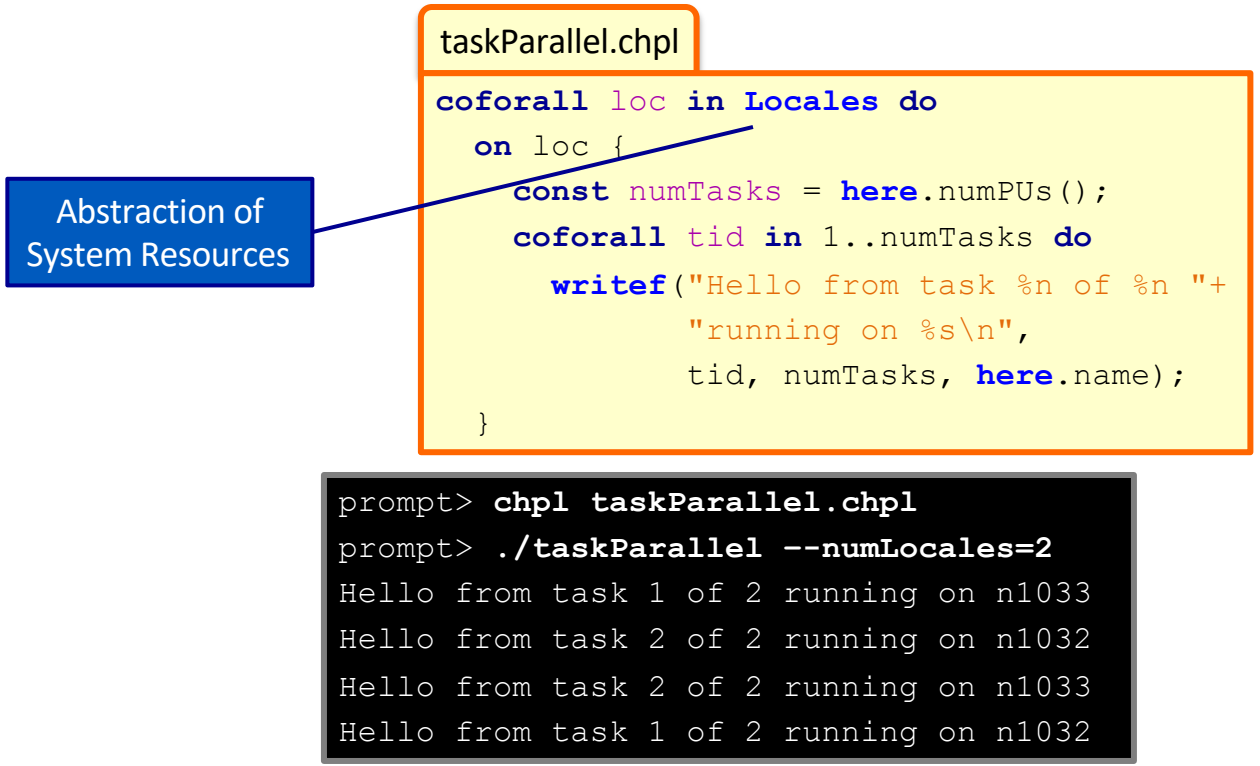
taskParallel.chpl

```
coforall loc in Locales do
  on loc {
    const numTasks = here.numPUs();
    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
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
```

© 2019 Cray, a Hewlett Packard Enterprise company

Task Parallelism and Locality, by example



© 2019 Cray, a Hewlett Packard Enterprise company

Task Parallelism and Locality, by example

High-Level Task Parallelism

taskParallel.chpl

```
coforall loc in Locales do
  on loc {
    const numTasks = here.numPUs();
    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
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
```

© 2019 Cray, a Hewlett Packard Enterprise company

Task Parallelism and Locality, by example

taskParallel.chpl

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

Control of Locality/Affinity

```
prompt> chpl taskParallel.chpl
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
```

© 2019 Cray, a Hewlett Packard Enterprise company

Task Parallelism and Locality, by example

taskParallel.chpl

```
coforall loc in Locales do
  on loc {
    const numTasks = here.numPUs();
    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
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
```

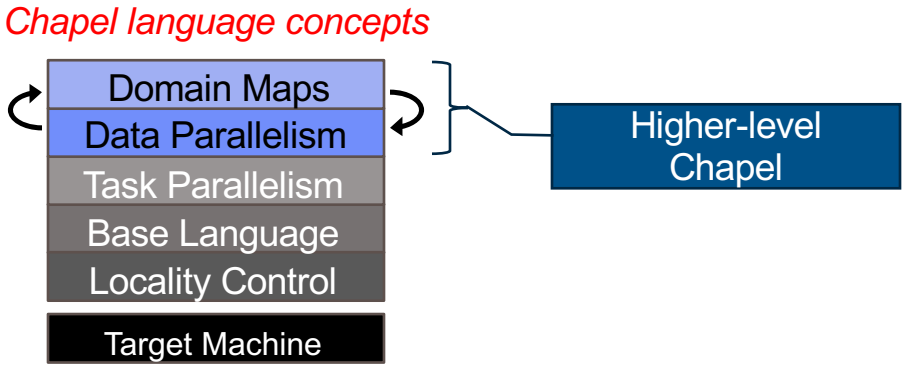
© 2019 Cray, a Hewlett Packard Enterprise company

Other Task Parallel Features

- **begin / cobegin statements:** other ways of creating tasks
- **atomic / synchronized variables:** for sharing data & coordination
- **task intents / task-private variables:** ways of referring to variables within tasks

© 2019 Cray, a Hewlett Packard Enterprise company

Data Parallelism in Chapel



© 2019 Cray, a Hewlett Packard Enterprise company

Data Parallelism, by example

dataParallel.chpl

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

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

```
prompt> chpl dataParallel.chpl
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
```

© 2019 Cray, a Hewlett Packard Enterprise company

Data Parallelism, by example

Domains (Index Sets)

dataParallel.chpl

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

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

```
prompt> chpl dataParallel.chpl
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
```

© 2019 Cray, a Hewlett Packard Enterprise company

Data Parallelism, by example

Arrays

dataParallel.chpl

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

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

```
prompt> chpl dataParallel.chpl
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
```

© 2019 Cray, a Hewlett Packard Enterprise company

Data Parallelism, by example

Data-Parallel Forall Loops

dataParallel.chpl

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

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

```
prompt> chpl dataParallel.chpl
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
```

© 2019 Cray, a Hewlett Packard Enterprise company

Data Parallelism, by example

So far, this is a shared memory program

Nothing refers to remote locales, explicitly or implicitly

dataParallel.chpl

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

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

```
prompt> chpl dataParallel.chpl
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
```

© 2019 Cray, a Hewlett Packard Enterprise company

Distributed Data Parallelism, by example

Domain Maps
(Map Data Parallelism to the System)

dataParallel.chpl

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

```
prompt> chpl dataParallel.chpl
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
```

© 2019 Cray, a Hewlett Packard Enterprise company

Distributed Data Parallelism, by example

dataParallel.chpl

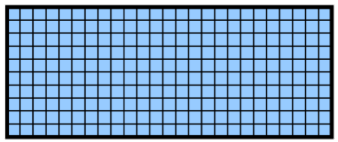
```
use CyclicDist;
config const n = 1000;
var D = {1..n, 1..n}
        dmapped Cyclic(startIdx = (1,1));
var A: [D] real;
forall (i,j) in D do
  A[i,j] = i + (j - 0.5)/n;
writeln(A);
```

```
prompt> chpl dataParallel.chpl
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
```

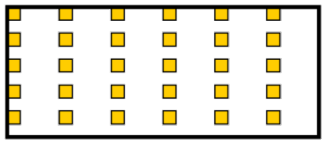
© 2019 Cray, a Hewlett Packard Enterprise company

Other Data Parallel Features

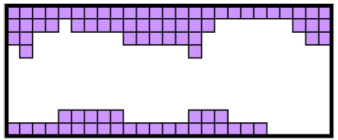
- **Parallel Iterators and Zippering**
- **Slicing:** refer to subarrays using ranges / domains
- **Promotion:** execute scalar functions in parallel using array arguments
- **Reductions:** collapse arrays to scalars or subarrays
- **Scans:** parallel prefix operations
- **Several Domain/Array Types**



dense



strided



sparse

associative

© 2019 Cray, a Hewlett Packard Enterprise company

STREAM Triad and HPCC RA Kernel, revisited

```
use …;

config const m = 1000,
             alpha = 3.0;

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

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

B = 2.0;
C = 1.0;

A = B + alpha * C;
```

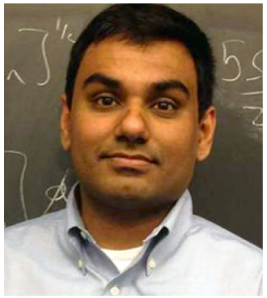
```
forall (_, r) in zip(Updates, RAStream()) do
  T[r & indexMask].xor(r);
```

© 2019 Cray, a Hewlett Packard Enterprise company

Recent Uses of Chapel

© 2019 Cray, a Hewlett Packard Enterprise company

Recent and Notable Chapel Use Cases



Simulation of Ultralight Dark Matter
Nikhil Padmanabhan et al.
Yale University

3D Computational Fluid Dynamics
Simon Bourgault-Côté, Matthieu Parenteau, et al.
École Polytechnique Montréal

Chapel Hypergraph Library (CHGL)
Louis Jenkins, Marcin Zalewski, et al.
PNNL

Arkouda: NumPy at Scale
Mike Merrill, Bill Reus, et al.
US DOD

© 2019 Cray, a Hewlett Packard Enterprise company

Simulating Ultralight Dark Matter

Who: Nikhil Padmanabhan, et al., *Yale University*

What: numerically intensive simulations that make predictions for the phenomenology of Ultralight Dark Matter (ULDM) using pseudo-spectral methods based on FFTs

Why: reproduce the expressiveness of an existing Python version, yet with performance and scalability

For more information:

- see Nikhil's upcoming presentation at the PAW-ATM 2019 workshop at SC19

© 2019 Cray, a Hewlett Packard Enterprise company

Chapel CFD Computations

Who: Simon Bourgault-Côté, Matthieu Parenteau, …, *École Polytechnique Montréal*

What: a 3D unstructured Reynolds Average Navier-Stokes (RANS) flow solver for use in aircraft design research

Why Chapel:

- simplifies development of distributed memory code:
  - easy to write and read
  - much more compact and flexible than MPI
- results in performance equivalent to other distributed C++ flow solvers
- interfaces with external libraries fairly easily

Challenges: requires care to avoid performance pitfalls (e.g., unintentional or excessive remote accesses)

© 2019 Cray, a Hewlett Packard Enterprise company

Chapel Hypergraph Library

Who: Louis Jenkins, Marcin Zalewski, et al., *Pacific Northwest National Laboratory*

What: a library supporting computations on HyperGraphs using a distributed, scalable representation

For more information:

- **GitHub repository:** https://github.com/pnnl/chgl
- **HPEC paper:** **Chapel HyperGraph Library (CHGL)**. Louis Jenkins, Tanveer Bhuiyan, Sarah Harun, Christopher Lightsey, David Mentgen, Sinan Aksoy, Timothy Stavenger, Marcin Zalewski, Hugh Medal, and Cliff Joslyn. *2018 IEEE High Performance Extreme Computing Conference (HPEC '18)*. September 25–27, 2018.

Example Use:

- **High Performance Hypergraph Analytics of Domain Name System Relationships**. Cliff Joslyn, Sinan Aksoy, Dustin Arendt, Louis Jenkins, Brenda Praggastis, Emilie Purvine, Marcin Zalewski. *HICSS Symposium on Cybersecurity Big Data Analytics*. Jaunary 8, 2019.

© 2019 Cray, a Hewlett Packard Enterprise company

Arkouda: NumPy at Massive Scale

Who: Mike Merrill, Bill Reus, et al., *U.S. Department of Defense*

What: a Python library supporting NumPy operations running over Chapel on large-scale systems

For more information, see:

- the following slides
- Bill's CLSAC 2019 presentation from which they were excerpted
- Mike's upcoming presentation at the PAW-ATM 2019 workshop at SC19
- the open source release, coming this month

© 2019 Cray, a Hewlett Packard Enterprise company

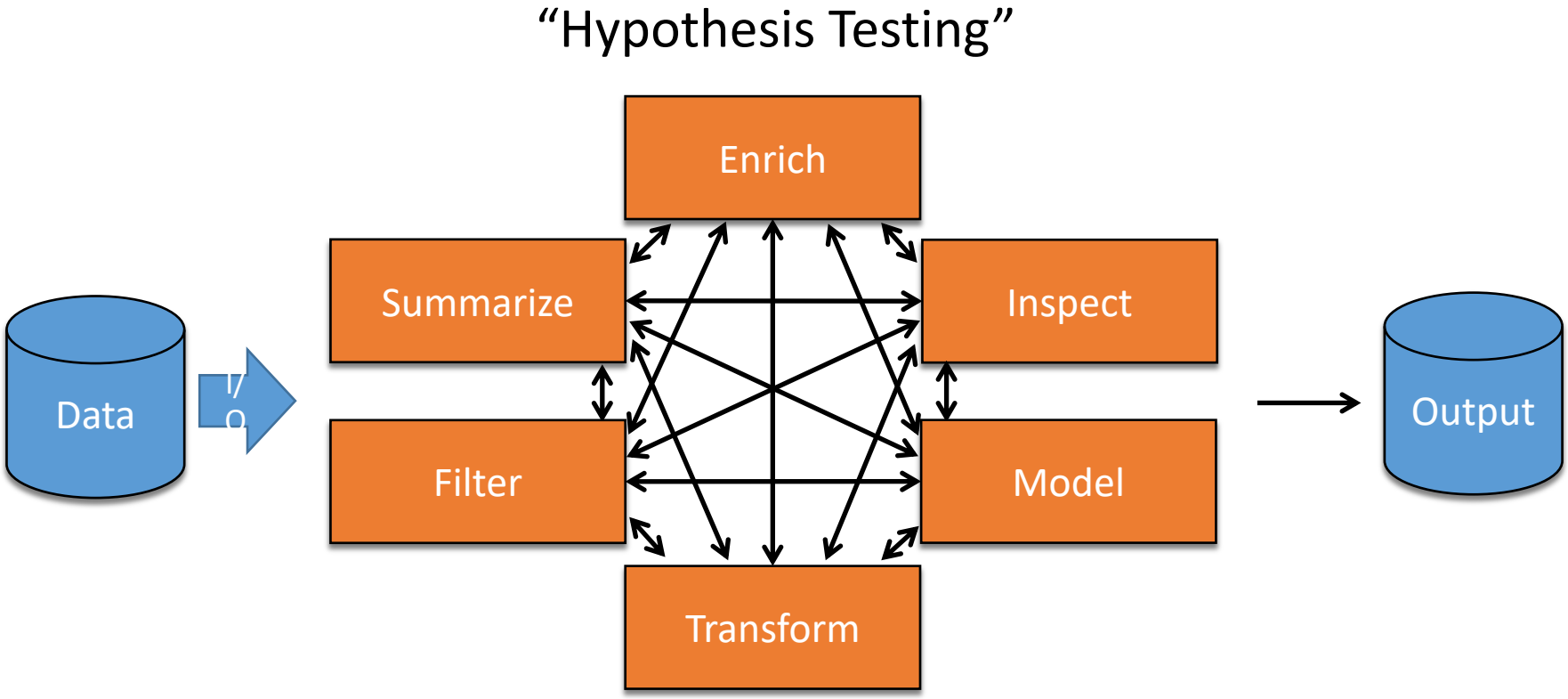
Data Science Needs Interactive Supercomputing

Dr. William Reus

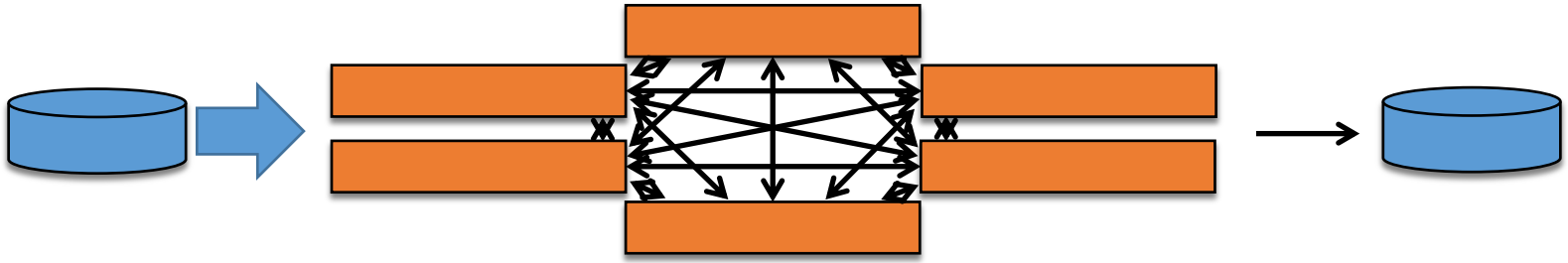
US Department of Defense

presented at CLSAC 2019, October 9, 2019

(Data) Science is Interactive



Implications for Computing



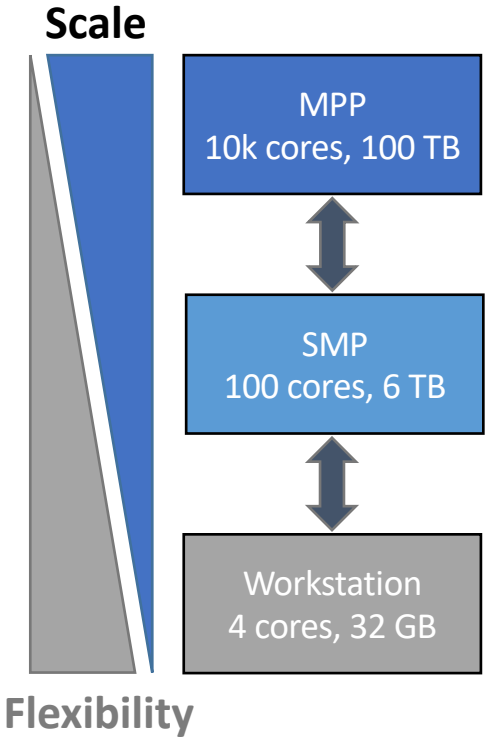
- Stay in memory
- Compute in small, reversible steps
- Enable introspection (code and state)
- Use other people's code
- Avoid boilerplate
- Maximize $\frac{t_{thinking}}{t_{thinking}+t_{coding}+t_{waiting}}$

So, basically Python…

…but fast

Interactive Computational Ladder

- We need the upper tier
  - Cybersecurity data >> 6 TB
- But hardware is the easy part
  - Need serious data engineering
  - Need to rethink job scheduling
  - Need an **HPC shell**



An HPC Shell for Data Science

Load Terabytes of data…

… into a familiar, interactive UI …

… where standard data science operations …

… execute within the human thought loop …

… and interoperate with optimized libraries.

Arkouda

Load Terabytes of data…

… into a familiar, interactive UI …

… where standard data science operations …

… execute within the human thought loop …

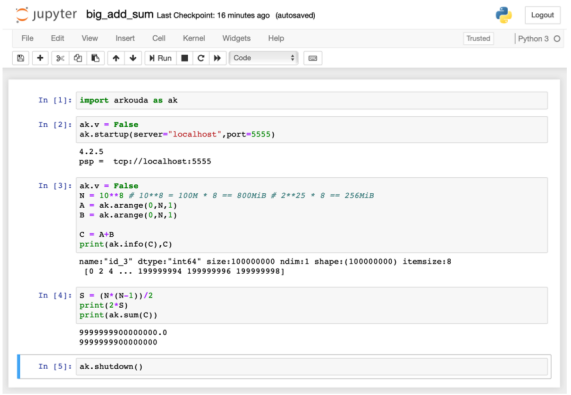
… and interoperate with optimized libraries.

Arkouda: an HPC shell for data science

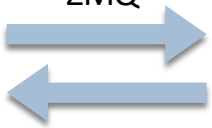
- Chapel backend (server)
- Jupyter/Python frontend (client)
- NumPy-like API

Arkouda Design

Jupyter/Python3



ZMQ



Chapel-Based Server

MPP
SMP
Cluster
Workstation
Laptop

Arkouda Startup

1) In terminal:

```
> arkouda_server -nl 96

server listening on hostname:port
```

2) In Jupyter:

```
In [2]: import arkouda as ak
        ak.connect(hostname, port)

        4.2.5
        psp =  tcp://nid00104:5555
        connected to tcp://nid00104:5555
```

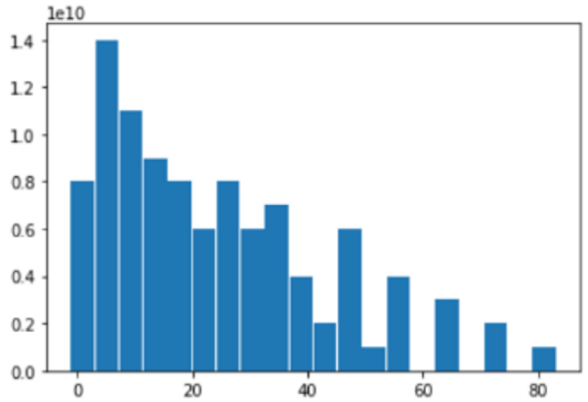
Data Exploration with Arkouda and NumPy

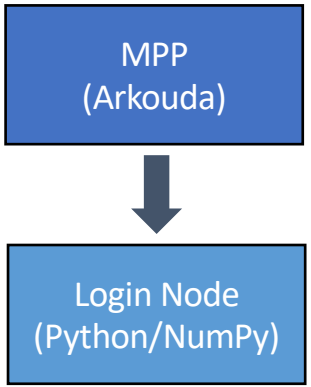
In [9]:
```
A = ak.randint(0, 10, 10**11)
B = ak.randint(0, 10, 10**11)
C = A * B
hist = ak.histogram(C, 20)
Cmax = C.max()
Cmin = C.min()
```

executed in 3.96s, finished 13:45:28 2019-09-12

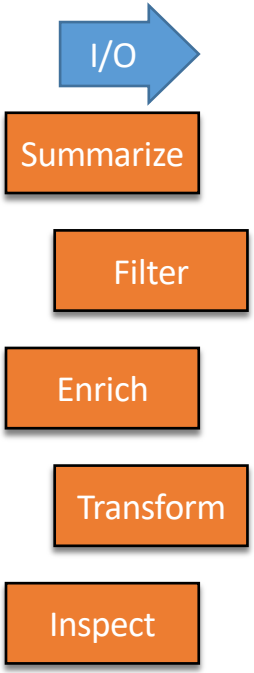
In [10]:
```
bins = np.linspace(Cmin, Cmax, 20)
_ = plt.bar(bins, hist.to_ndarray(), width=(Cmax-Cmin)/20)
```

executed in 193ms, finished 13:45:28 2019-09-12





Hypothesis Testing on 50 Billion Records



Operation	Example	Approximate Time (seconds)
Read from disk	A = ak.read_hdf()	30-60
Scalar Reduction	A.sum()	< 1
Histogram	ak.histogram(A)	< 1
Vector Ops	A + B, A == B, A & B	< 1
Logical Indexing	A[A == val]	1 - 10
Set Membership	ak.in1d(A, set)	1
Gather	B = Table[A]	30 - 300
Group by Key	G = ak.GroupBy(A)	60
Aggregate per Key	G.aggregate(B, 'sum')	15
Get Item	print(A[42])	< 1
Export to NumPy	A[:10**6].to_ndarray()	2

- A, B are 50 billion-element arrays
- Timings measured on real data
- Hardware: Cray XC40
  - 96 nodes
  - 3072 cores
  - 24 TB
  - Lustre filesystem

What about Model ?

- Vision: Expose HPC libraries to Python via Arkouda
  - FFT
  - Tensor decomposition
  - Graph algorithms
  - Solvers
  - CHGL (Chapel HyperGraph Library from PNNL)
  - Anything you could link into a Chapel application (via C or LLVM)
- Need to standardize a distributed array interface with the HPC community

Arkouda Design

- Why Chapel?
  - High-level language with C-comparable performance
  - Parallelism is a first-class citizen
  - Great distributed array support
  - Portable code: from laptop up to supercomputer

Future Directions

- Open source release
- Tactical functionality
  - Strings and/or categorical dtype
  - Actual DataFrame class
  - Segmented arrays for sparse linear algebra (e.g. GraphBLAS)
- Strategic goals
  - Integration of Parallel Libraries
  - Multi-user support

Summary and Resources

© 2019 Cray, a Hewlett Packard Enterprise company

Summary of this Talk

Chapel cleanly and orthogonally supports…

…expression of parallelism and locality

…specifying how to map computations to the system

Chapel is powerful:

- supports succinct, straightforward code
- can result in performance that competes with (or beats) C+MPI+OpenMP

Chapel is attractive to scientists and Python programmers

- as a native language: similarly readable / writeable, yet scalable
- as an implementation option for Python libraries

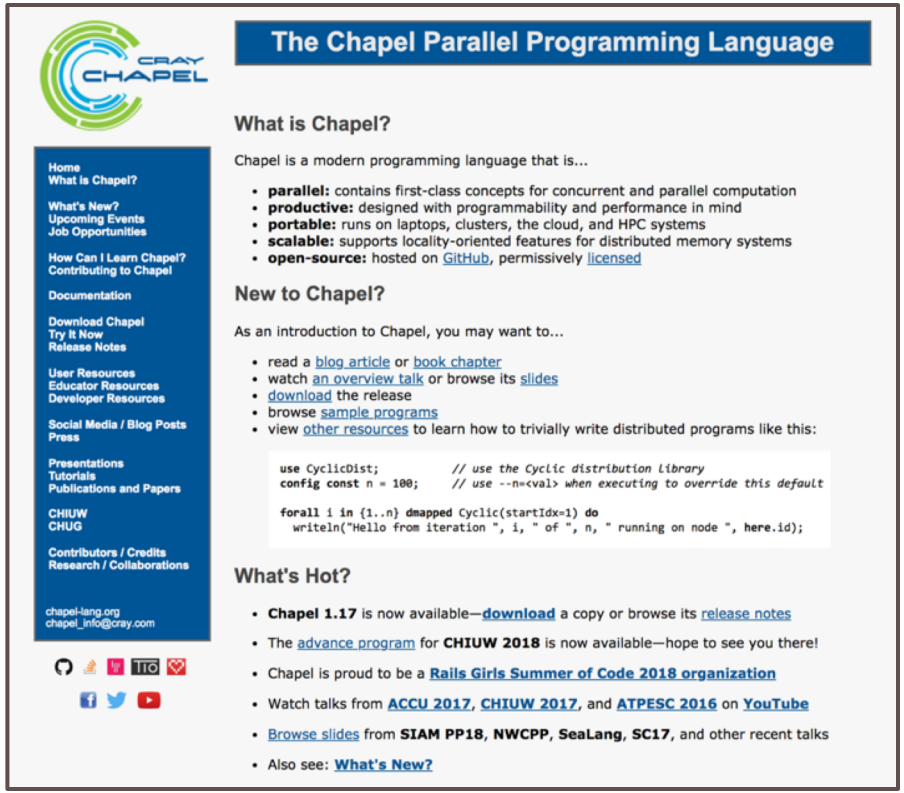
© 2019 Cray, a Hewlett Packard Enterprise company

Chapel Central

CRAY
a Hewlett Packard Enterprise company

https://chapel-lang.org

- downloads
- presentations
- papers
- resources
- documentation

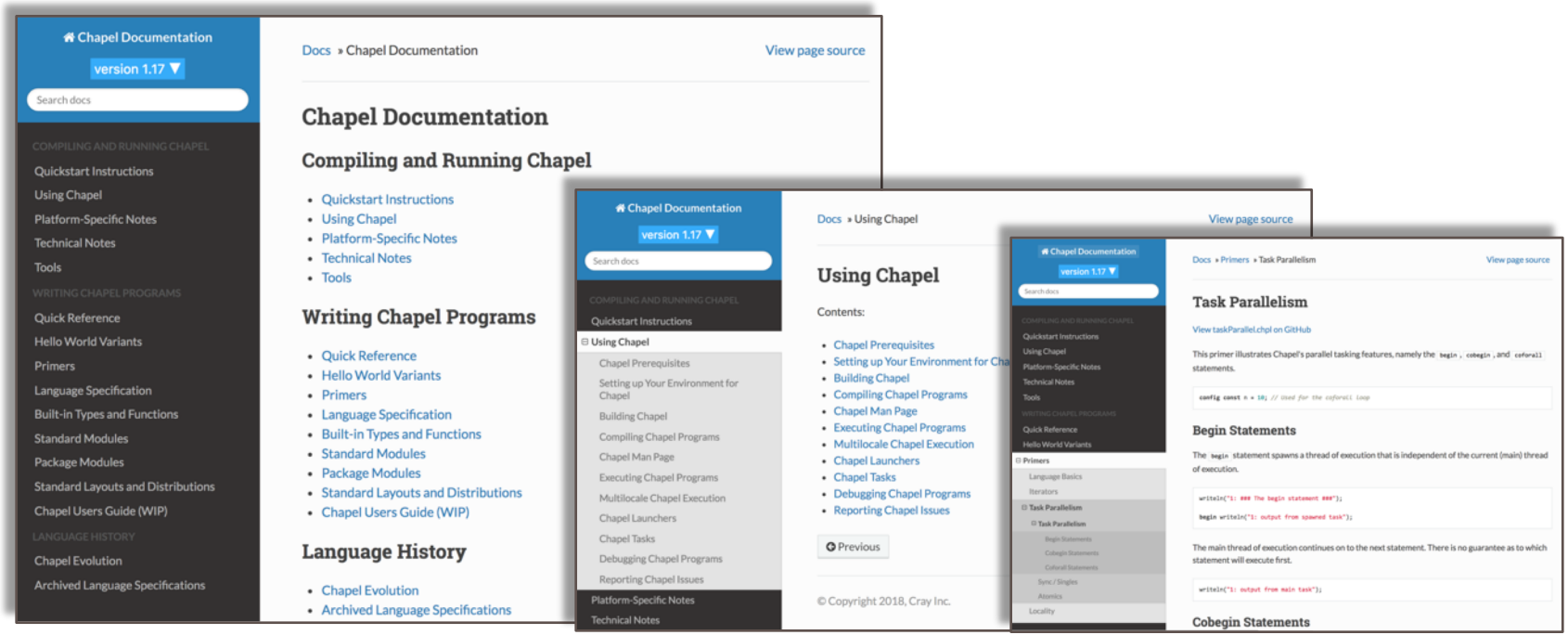


© 2019 Cray, a Hewlett Packard Enterprise company

Chapel Online Documentation

CRAY
a Hewlett Packard Enterprise company

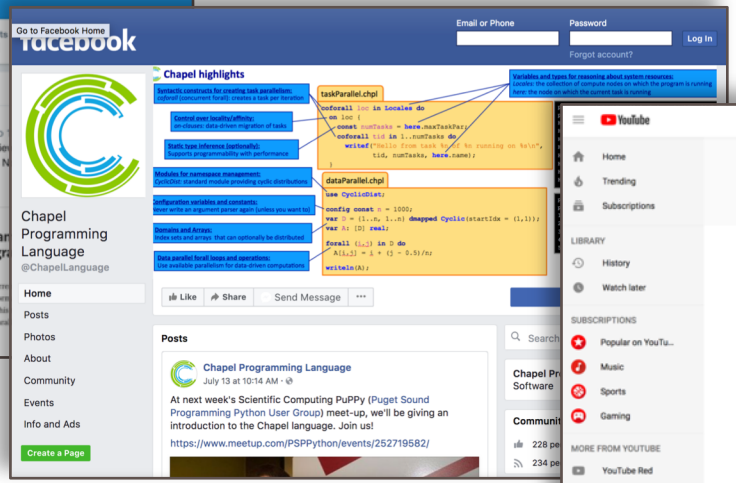
https://chapel-lang.org/docs: ~200 pages, including primer examples



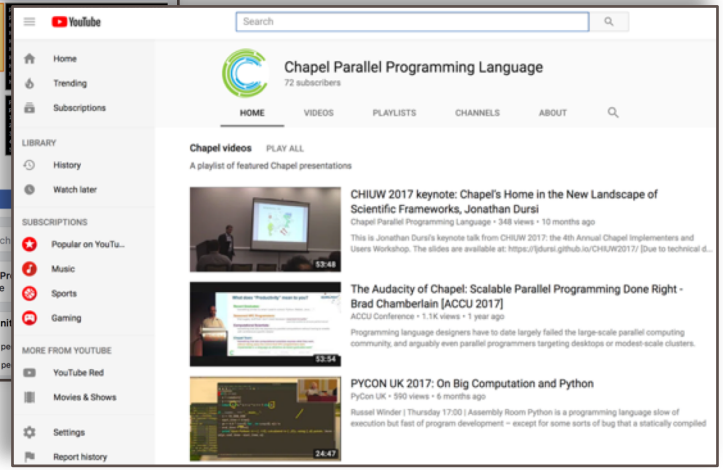
© 2019 Cray, a Hewlett Packard Enterprise company

Chapel Social Media (no account required)

http://twitter.com/ChapelLanguage



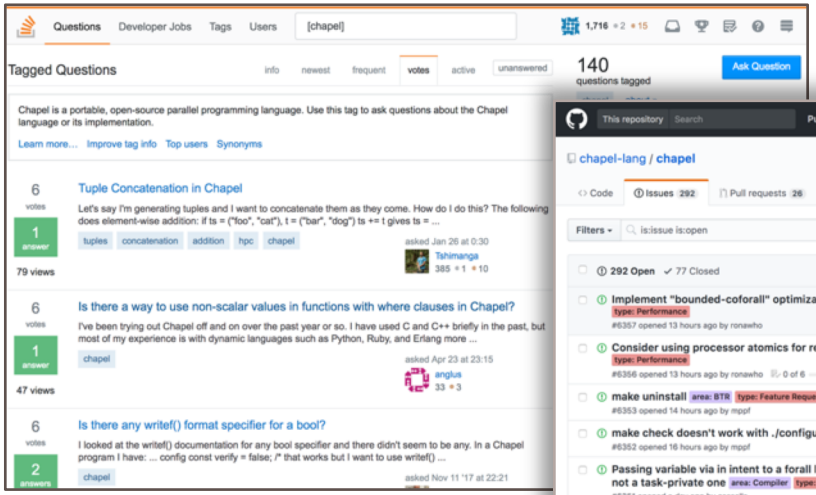
http://facebook.com/ChapelLanguage



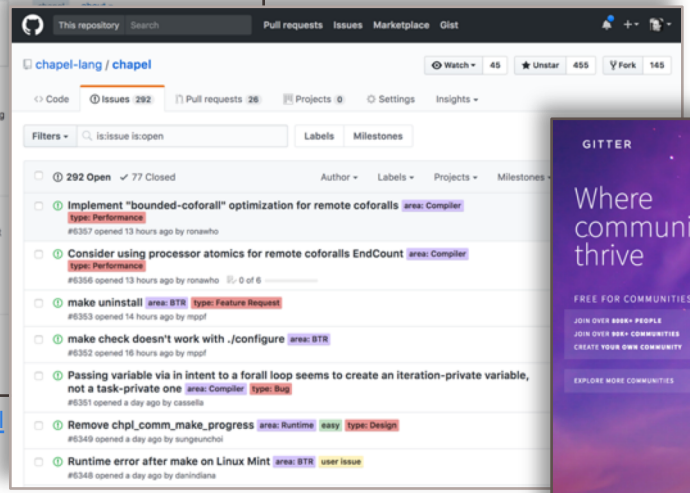
https://www.youtube.com/channel/UCHmm27bYjhknK5mU7ZzPGsQ/

© 2019 Cray, a Hewlett Packard Enterprise company

Chapel Community



https://stackoverflow.com/questions/tagged/chapel



https://github.com/chapel-lang/chapel/issues

https://gitter.im/chapel-lang/chapel

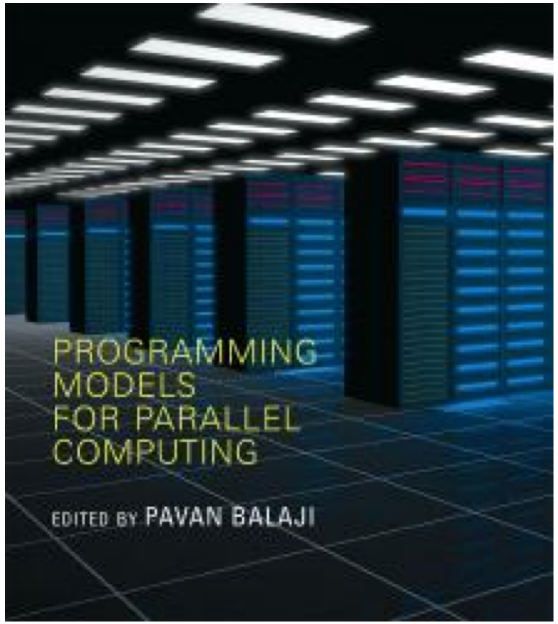
read-only mailing list: chapel-announce@lists.sourceforge.net (~15 mails / year)

© 2019 Cray, a Hewlett Packard Enterprise company

Suggested Reading: Chapel history and overview

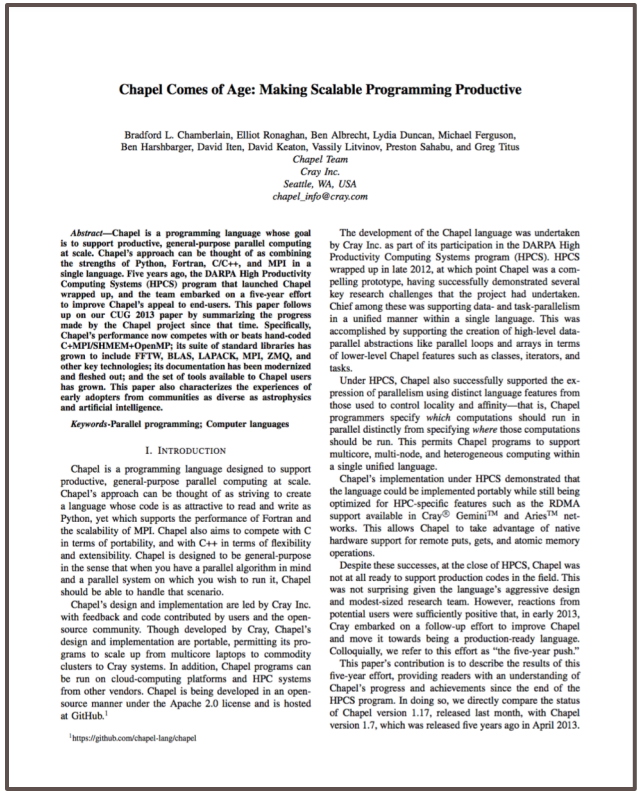
Chapel chapter from *Programming Models for Parallel Computing*

- a detailed overview of Chapel's history, motivating themes, features
- published by MIT Press, November 2015
- edited by Pavan Balaji (Argonne)
- chapter is also available online



© 2019 Cray, a Hewlett Packard Enterprise company

Suggested Reading: Recent Progress (CUG 2018)



Chapel Comes of Age: Making Scalable Programming Productive

Bradford L. Chamberlain, Elliot Ronaghan, Ben Albrecht, Lydia Duncan, Michael Ferguson,
Ben Harshbarger, David Iten, David Keaton, Vassily Litvinov, Preston Sahabu, and Greg Titus
Chapel Team
Cray Inc.
Seattle, WA, USA
chapel_info@cray.com

***Abstract*—Chapel is a programming language whose goal is to support productive, general-purpose parallel computing at scale. Chapel's approach can be thought of as combining the strengths of Python, Fortran, C/C++, and MPI in a single language. Five years ago, the DARPA High Productivity Computing Systems (HPCS) program that launched Chapel wrapped up, and the team embarked on a five-year effort to improve Chapel's appeal to end-users. This paper follows up on our CUG 2013 paper by summarizing the progress made by the Chapel project since that time. Specifically, Chapel's performance now competes with or beats hand-coded C+MPI/SHMEM+OpenMP; its suite of standard libraries has grown to include FFTW, BLAS, LAPACK, MPI, ZMQ, and other key technologies; its documentation has been modernized and fleshed out; and the set of tools available to Chapel users has grown. This paper also characterizes the experiences of early adopters from communities as diverse as astrophysics and artificial intelligence.**

***Keywords*-Parallel programming; Computer languages**

I. Introduction

Chapel is a programming language designed to support productive, general-purpose parallel computing at scale. Chapel's approach can be thought of as striving to create a language whose code is as attractive to read and write as Python, yet which supports the performance of Fortran and the scalability of MPI. Chapel also aims to compete with C in terms of portability, and with C++ in terms of flexibility and extensibility. Chapel is designed to be general-purpose in the sense that when you have a parallel algorithm in mind and a parallel system on which you wish to run it, Chapel should be able to handle that scenario.

Chapel's design and implementation are led by Cray Inc. with feedback and code contributed by users and the open-source community. Though developed by Cray, Chapel's design and implementation are portable, permitting its programs to scale up from multicore laptops to commodity clusters to Cray systems. In addition, Chapel programs can be run on cloud-computing platforms and HPC systems from other vendors. Chapel is being developed in an open-source manner under the Apache 2.0 license and is hosted at GitHub.[1]

[1]https://github.com/chapel-lang/chapel

The development of the Chapel language was undertaken by Cray Inc. as part of its participation in the DARPA High Productivity Computing Systems program (HPCS). HPCS wrapped up in late 2012, at which point Chapel was a compelling prototype, having successfully demonstrated several key research challenges that the project had undertaken. Chief among these was supporting data- and task-parallelism in a unified manner within a single language. This was accomplished by supporting the creation of high-level data-parallel abstractions like parallel loops and arrays in terms of lower-level Chapel features such as classes, iterators, and tasks.

Under HPCS, Chapel also successfully supported the expression of parallelism using distinct language features from those used to control locality and affinity—that is, Chapel programmers specify *which* computations should run in parallel distinctly from specifying *where* those computations should be run. This permits Chapel programs to support multicore, multi-node, and heterogeneous computing within a single unified language.

Chapel's implementation under HPCS demonstrated that the language could be implemented portably while still being optimized for HPC-specific features such as the RDMA support available in Cray® Gemini™ and Aries™ networks. This allows Chapel to take advantage of native hardware support for remote puts, gets, and atomic memory operations.

Despite these successes, at the close of HPCS, Chapel was not at all ready to support production codes in the field. This was not surprising given the language's aggressive design and modest-sized research team. However, reactions from potential users were sufficiently positive that, in early 2013, Cray embarked on a follow-up effort to improve Chapel and move it towards being a production-ready language. Colloquially, we refer to this effort as "the five-year push."

This paper's contribution is to describe the results of this five-year effort, providing readers with an understanding of Chapel's progress and achievements since the end of the HPCS program. In doing so, we directly compare the status of Chapel version 1.17, released last month, with Chapel version 1.7, which was released five years ago in April 2013.

[paper](#) and [slides](#) available at chapel-lang.org

© 2019 Cray, a Hewlett Packard Enterprise company

Summary of this Talk

Chapel cleanly and orthogonally supports…

…expression of parallelism and locality

…specifying how to map computations to the system

Chapel is powerful:

- supports succinct, straightforward code
- can result in performance that competes with (or beats) C+MPI+OpenMP

Chapel is attractive to scientists and Python programmers

- as a native language: similarly readable / writeable, yet scalable
- as an implementation option for Python libraries

© 2019 Cray, a Hewlett Packard Enterprise company

FORWARD LOOKING STATEMENTS

This presentation may contain forward-looking statements that involve risks, uncertainties and assumptions. If the risks or uncertainties ever materialize or the assumptions prove incorrect, the results of Hewlett Packard Enterprise Company and its consolidated subsidiaries ("Hewlett Packard Enterprise") may differ materially from those expressed or implied by such forward-looking statements and assumptions. All statements other than statements of historical fact are statements that could be deemed forward-looking statements, including but not limited to any statements regarding the expected benefits and costs of the transaction contemplated by this presentation; the expected timing of the completion of the transaction; the ability of HPE, its subsidiaries and Cray to complete the transaction considering the various conditions to the transaction, some of which are outside the parties' control, including those conditions related to regulatory approvals; projections of revenue, margins, expenses, net earnings, net earnings per share, cash flows, or other financial items; any statements concerning the expected development, performance, market share or competitive performance relating to products or services; any statements regarding current or future macroeconomic trends or events and the impact of those trends and events on Hewlett Packard Enterprise and its financial performance; any statements of expectation or belief; and any statements of assumptions underlying any of the foregoing. Risks, uncertainties and assumptions include the possibility that expected benefits of the transaction described in this presentation may not materialize as expected; that the transaction may not be timely completed, if at all; that, prior to the completion of the transaction, Cray's business may not perform as expected due to transaction-related uncertainty or other factors; that the parties are unable to successfully implement integration strategies; the need to address the many challenges facing Hewlett Packard Enterprise's businesses; the competitive pressures faced by Hewlett Packard Enterprise's businesses; risks associated with executing Hewlett Packard Enterprise's strategy; the impact of macroeconomic and geopolitical trends and events; the development and transition of new products and services and the enhancement of existing products and services to meet customer needs and respond to emerging technological trends; and other risks that are described in our Fiscal Year 2018 Annual Report on Form 10-K, and that are otherwise described or updated from time to time in Hewlett Packard Enterprise's other filings with the Securities and Exchange Commission, including but not limited to our subsequent Quarterly Reports on Form 10-Q. Hewlett Packard Enterprise assumes no obligation and does not intend to update these forward-looking statements.

© 2019 Cray, a Hewlett Packard Enterprise company

THANK YOU

QUESTIONS?


chapel_info@cray.com


@ChapelLanguage


chapel-lang.org

cray.com


@cray_inc


linkedin.com/company/cray-inc-/