Chapel Comes of Age: A Language for Productivity, Parallelism, and Performance

Brad Chamberlain
HPC Knowledge Meeting `19
June 14, 2019
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 Principal Engineer at Cray Inc.
• The technical lead / a founding member of the Chapel project
What is Chapel?

**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
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”

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.”
## Why Consider New Languages at all?

| Syntax               | • High level, elegant syntax  
<table>
<thead>
<tr>
<th></th>
<th>• Improve programmer productivity</th>
</tr>
</thead>
</table>
| 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*]
Outline

✓ Context and Motivation

Chapel and Productivity

• A Brief Tour of Chapel Features
• Arkouda: NumPy over Chapel
• Summary and Resources
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]
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:**

```
\[
\begin{align*}
A & = \quad \\
B & + \quad \\
C & \cdot \;
\end{align*}
\]
```
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 (shared memory / multicore):
STREAM Triad: a trivial parallel computation

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

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

In pictures, in parallel (distributed memory):
STREAM Triad: a trivial parallel computation

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

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

In pictures, in parallel (distributed memory multicore):
#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;
    }

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

    scalar = 3.0;

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

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

    return 0;
}
STREAM Triad: C + MPI + OpenMP

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

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

int HPCC_StraStarstream(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;
}
```
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;
}

typedef struct {
  int a, b, c;
} HPCC_Params;

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).
", VectorSize);
      fflush( outFile );
    }
    return 1;
  }

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

  scalar = 3.0;
  #ifdef _OPENMP
  #pragma omp parallel for
  #endif
  for (j=0; j<VectorSize; j++)
    a[j] = b[j]+scalar*c[j];

  HPCC_free(c);
  HPCC_free(b);
  HPCC_free(a);
  return 0;
}

STREAM Triad: Chapel

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?
HPCC STREAM Triad: Chapel vs. C+MPI+OpenMP
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
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
/* 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++];
          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);
  }
  /* send remaining updates in buckets */
do {
    MPI_Test(&inreq, &have_done, &status);
    if (have_done) {
      if (status.MPI_TAG == UPDATED_TAG) {
        MPI_Get_count(&status, tparams.dtype64, &recvUpdates);
        bufferBase = 0;
        for (j=0; j < recvUpdates; j++) {
          inmsg = LocalRecvBuffer[bufferBase++];
          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(&inreq, &have_done, &status);
  if (have_done) {
    if (status.MPI_TAG == FINISHED_TAG) {
      if (NumberReceiving > 0) {
        /* send our done messages */
        for (proc_count = 0; proc_count < tparams.NumProcs; ++proc_count) {
            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_Test(inreq, &have_done, &status);
        if (have_done) {
          if (status.MPI_TAG == UPDATED_TAG) {
            MPI_Get_count(&status, tparams.dtype64, &recvUpdates);
            bufferBase = 0;
            for (j=0; j < recvUpdates; j++) {
              inmsg = LocalRecvBuffer[bufferBase++];
              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);
          }
          /* send our done messages */
        } while (have_done && NumberReceiving > 0);
      }
    }
  }
/* 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) {
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);
*/
HPCC RA: Chapel vs. C+MPI

RA Performance (GUPS)

- Chapel 1.19
- MPI (bucketing)

Locales (x 36 cores / locale)

GUPS
/* 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;
        }
    }
}
```

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);

    if ( 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 */
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.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[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);
        } 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[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);
        } 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 our done messages */
    for (proc_count = 0; proc_count < tparams.NumProcs; ++proc_count) {
            MPI_REQUEST_NULL; continue; }
        /* send garbage - who cares, no one will look at it */
        MPI_Isend(&Ran, 0, tparams.dtype64, (int)proc_count, 
        FINISHED_TAG, MPI_COMM_WORLD, &outreq);
        pendingUpdates -= 1;
    }
    /* 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[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);
}

forall (r in RAStream()) do 
    HPCC_Table[LocalOffset] ^= inmsg;
forall (_, r) in zip(Updates, RAStream()) do 
    T[r & indexMask].xor(r);

forall (_, r) in zip(Updates, RAStream()) do 
    HPCC_Table[LocalOffset] ^= inmsg;
forall (_, r) in zip(Updates, RAStream()) do 
    T[r & indexMask].xor(r);

forall (_, r) in zip(Updates, RAStream()) do 
    HPCC_Table[LocalOffset] ^= inmsg;
forall (_, r) in zip(Updates, RAStream()) do 
    T[r & indexMask].xor(r);
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]
HPC Patterns: Chapel vs. Reference

Local loop kernels

Embarrassing/Pleasing Parallelism

Bucket-Exchange Pattern

Stencil Boundary Exchanges

Global Random Updates

Nightly performance tickers online at: https://chapel-lang.org/perf-nightly.html
HPC Patterns: Chapel vs. Reference

LCALS: Chapel vs. Reference

HPCC RA

STREAM

Triad

ISx

PRK

Stencil

HPCC STREAM Triad: Chapel vs. Reference

ISx: Chapel vs. Reference

PRK Stencil: Chapel vs. Reference

Nightly performance tickers online at:
https://chapel-lang.org/perf-nightly.html
A Brief Tour of Chapel Features
Chapel Feature Areas

Chapel language concepts

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

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

Lower-level Chapel
Base Language Features, by example

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

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

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

0
1
1
2
3
5
8
...
Base Language Features, by example

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

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

config const n = 10;

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

Configuration declarations
(support command-line overrides)
./fib --n=1000000
Base Language Features, by example

Iterators

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

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

0
1
1
2
3
5
8
...
Base Language Features, by example

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

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

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

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

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

Explicit types also supported
Base Language Features, by example

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

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

```
0
1
1
2
3
5
8...
```
Base Language Features, by example

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

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

```plaintext
config const n = 10;

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

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

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

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

```
config const n = 10;

for (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`
- ...

Range types and operators
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
...```
iter fib(n) {
    var current = 0,
        next = 1;

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

cfg const n = 10;

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

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
...
Other Base Language Features

• **Object-oriented programming** (value- and reference-based)
  • Managed objects and lifetime checking
  • Nillable vs. non-nillable 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…
Task Parallelism and Locality Control

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

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

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

Locales:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

User’s main() executes on locale #0
Task Parallelism and Locality, by example

```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
```
Task Parallelism and Locality, by example

**Abstraction of System Resources**

```chpl
const numTasks = here.numPUs();
coforall tid in 1..numTasks do
  wri"Hello from task %n of %n "+
"Running on %s
", 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
```
Task Parallelism and Locality, by example

```chapl
const numTasks = here.numPUs();
coforall tid in 1..numTasks do
  writesf("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
```
Task Parallelism and Locality, by example

So far, this is a shared memory program

Nothing refers to remote locales, explicitly or implicitly

```
const numTasks = here.numPUs();
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
prompt> ./taskParallel
Hello from task 2 of 2 running on n1032
Hello from task 1 of 2 running on n1032
```
Task Parallelism and Locality, by example

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

```
prompt> chpl taskParallel.chpl
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 and Locality, by example

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

Abstraction of System Resources

```
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
```
Task Parallelism and Locality, by example

```
coforall loc in Locales do
  on loc {
    const numTasks = here.numPUs();
    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
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 and Locality, by example

```chapl
coforall loc in Locales do
  on loc {
    const numTasks = here.numPUs();
    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
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 and Locality, by example

taskParallel.chpl

coforall loc in Locales do
  on loc {
    const numTasks = here.numPUs();
    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
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
Other Task Parallel Features

• **atomic / synchronized variables**: for sharing data & coordination
• **begin / cobegin statements**: other ways of creating tasks
Data Parallelism in Chapel

Chapel language concepts

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

Higher-level Chapel
Data Parallelism, by example

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

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

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

```
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);
```
Data Parallelism, by example

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

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

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

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

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

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

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

So far, this is a shared memory program

Nothing refers to remote locales, explicitly or implicitly
### Distributed Data Parallelism, by example

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

```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
```
1. Distributed Data Parallelism, by example

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

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

Examples:

- "steve"
- "lee"
- "sung"
- "david"
- "jacob"
- "albert"
- "brad"
STREAM Triad and HPCC RA Kernel, revisited

```plaintext
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);
```
Arkouda: NumPy over Chapel
Arkouda: Context

Motivation: Say you’ve got…
…a bunch of Python programmers
…HPC-scale problems to solve
…access to HPC systems
How should you leverage these Python programmers to get your work done?

Concept: Develop Python libraries that are implemented in Chapel
⇒ get performance, as with Python-over-C, but also parallelism + scalability
Even Better: use NumPy interfaces to make it trivial / familiar for users
Sample use from Jupyter

```python
import arkouda as ak

ak.v = False
ak.connect(server="localhost", port=5555)

4.2.5
psp = tcp://localhost:5555

ak.v = False
N = 10**8 # 10**8 = 100M * 8 == 800MiB # 2**25 * 8 == 256MiB
A = ak.arange(0, N, 1)
B = ak.arange(0, N, 1)
C = A+B
print(ak.info(C), C)

name:"id_3" dtype:"int64" size:100000000 ndim:1 shape:(100000000) itemsize:8
[0 2 4 ... 199999994 199999996 199999998]

S = (N*(N-1))/2
print(2*S)
print(ak.sum(C))

999999990000000.0
999999990000000.0
```

© 2019 Cray Inc.
Arkouda Accomplishments

By taking this approach, this user was able to:

- interact with a distributed, running Chapel program from Python within Jupyter
- run the same back-end program on...
  - a Cray XC
  - an HPE Superdome X
  - an Infiniband cluster
  - a Mac laptop
- compute on TB-sized arrays in seconds
- with ~1 month of effort
“Why Chapel?”

• High level — makes for less code
  • Great support for array operations and distributed arrays
  • Direct support for synchronized/atomic variables
  • Close to “Pythonic” (for a statically typed language)
    • Provides a gateway for data scientists ready to go beyond Python

• Portability and Scalability
  • Same code runs on shared- or distributed-memory systems
  • “From Raspberry Pi to Cray XC”

• Integrates with (distributed) numerical libraries (e.g., FFTW, FFTW-MPI)
Summary and Resources
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 Python programmers
  • as a native language: similarly readable / writeable, yet scalable
  • as an implementation option for Python libraries
Chapel Central

https://chapel-lang.org

- downloads
- presentations
- papers
- resources
- documentation

The Chapel Parallel Programming Language

What is Chapel?
Chapel is a modern programming language that is...
- parallel: contains first-class concepts for concurrent and parallel computation
- productive: designed with programmability and performance in mind
- portable: runs on laptops, clusters, the cloud, and HPC systems
- scalable: supports locality-oriented features for distributed memory systems
- open-source: hosted on GitHub, permissively licensed

New to Chapel?
As an introduction to Chapel, you may want to...
- read a blog article or buy a chapter
- watch an overview talk or browse its slides
- download the release
- browse sample programs
- view other resources to learn how to trivially write distributed programs like this:

```chapel
use CyclicDist; // use the Cyclic distribution library
config const n = 100; // use --n=vals when executing to override this default
forall i in [1..n] mapped Cyclic(startid=i) do
    writef("Hello from iteration ", i, " of ", n, ", running on node ", here.id);
```

What's Hot?
- Chapel 1.17 is now available—download a copy or browse its release notes
- The advance program for CHIUW 2018 is now available—hope to see you there!
- Chapel is proud to be a Rails Girls Summer of Code 2018 organization
- Watch talks from ACCU 2017, CHIUW 2017, and ATPESC 2016 on YouTube
- Browse slides from SIAM PP18, NWCPP, SeaLang, SC17, and other recent talks
- Also see: What's New?
Chapel Online Documentation

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

- [https://stackoverflow.com/questions/tagged/chapel](https://stackoverflow.com/questions/tagged/chapel)
- [https://github.com/chapel-lang/chapel/issues](https://github.com/chapel-lang/chapel/issues)
- [https://gitter.im/chapel-lang/chapel](https://gitter.im/chapel-lang/chapel)

read-only mailing list: chapel-announce@lists.sourceforge.net (~15 mails / year)
Chapel Social Media (no account required)

http://twitter.com/ChapelLanguage

http://facebook.com/ChapelLanguage

https://www.youtube.com/channel/UCHmm27bYihknK5mU7ZzPGsQ/
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](#)
Chapel Comes of Age: Making Scalable Programming Productive

Bradford L. Chamberlain, Elliot Horowitz, Ben Alpert, Lynda Duncan, Michael Frazier, Ben Hindenburg, David Ives, David Kozens, Vasuki Laxminarayan, Prashant Sabharwal, and Greg Titze
Cray Inc.
Seattle, WA, USA
cray.info@cray.com

Abstract—Chapel is a programming language whose goal is to offer productive, general-purpose parallel computing at scale. Chapel’s approach can be described as combining the strengths of Fortran, Python, C++, and MPI in a single language. Five years ago, the DARPA High Productivity Computing Systems (HPCS) program launched Chapel as a research tool to explore new programming paradigms with the goal of improving Chapel’s current limitations. This paper follows up on our 2013 DAR paper by summarizing the progress made by the Chapel project since that time. Specifically, Chapel’s performance now compares well with hand-coded implementations in the target domains of FETI, HPC, LAPACK, MPI, and OpenMP, and other key technologies. Transformational features that have been implemented in Chapel (such as automatic parallelization and nested parallelism) were not set of tools available to Chapel users back then. This paper also highlights the experiences of using Chapel in large-scale implementations so diverse as astrophysics and artificial intelligence.

Keyword: Parallel-Programming; Computer language

1. INTRODUCTION

Chapel is a parallel programming language designed to support productive, general-purpose parallel computing at scale. Chapel’s design goal is to develop a language whose code is easier to read and write than Fortran, but which supports the performance of Fortran and the scalability of MPI. Chapel also aims to complement C in terms of portability, and with C++ it attains flexibility and extensibility. Chapel is designed to be a general-purpose tool 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 integration to yield a high-performance, low-overhead, scalable component of the Cray system. In addition, Chapel programs with a substantial fraction of code written in Fortran, C, and C++ from other vendors. Chapel is being developed in an open-source manner under the Apache 2.0 license, and is hosted at GitHub.

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 2017, at which point Chapel was a compelling prototype, having successfully demonstrated several target domains on a large-scale supercomputer. Chapel’s major goals were to support valid and valid parallelism in a single language, through the use of Chapel’s support for nested parallelism and nested parallelism.

The goal of this paper is to show the current state of the Chapel project, how the language has been implemented and modified, and the set of tools available to Chapel users back then. This paper also highlights the experiences of using Chapel in large-scale implementations so diverse as astrophysics and artificial intelligence.

Under HPCS, Chapel also successfully supported the expansion 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 multi-core, multi-node, and heterogeneous computing within a single, unified language.

Chapel’s implementation under HPCS demonstrated that the language could be implemented parallelly 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, data, and acyclic 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 departure from Fortran. Chapel’s development is still in progress, and development continues at a more moderate pace. Chapel’s development is an open-source effort, and much of the code is shared with the open-source community. Chapel is being developed in an open-source manner under the Apache 2.0 license, and is hosted at GitHub.

The paper’s contribution is to describe the results of this two-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 states of Chapel version 1.17, released last month, with Chapel version 1.15, which was released five years ago in April 2013.

© 2019 Cray Inc.
The 6th Annual Chapel Implementers and Users Workshop
  • sponsored by ACM SIGPLAN
  • held in conjunction with PLDI 2019, FCRC 2019
  • June 22-23, Phoenix AZ

Keynote: Anshu Dubey (Argonne)

Programming Abstractions for Orchestration of HPC Scientific Computing
Community Talks:

• Arkouda
• hybrid CPU-GPU computations with Chapel
• Stencil computations: getting performance for a student exercise
• Chapel Graph Library

Cray talks:

• Chapel’s use in AI / HPO
• interoperability improvements for Python, C, and Fortran
• recent performance optimizations
• radix sort in Chapel

Also: state of the project, lightning talks, coding day
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 Python programmers
  • as a native language: similarly readable / writeable, yet scalable
  • as an implementation option for Python libraries
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.
THANK YOU