

Chapel 101

Brad Chamberlain

CHIUW 2020, May 22, 2020

✉ chapel_info@cray.com
🌐 chapel-lang.org
🐦 @ChapelLanguage

CRAY[®]
a Hewlett Packard Enterprise company



What is Chapel?

Chapel: A modern parallel programming language

- portable & scalable
- open-source & collaborative

Goals:

- Support general parallel programming
- Make parallel programming at scale far more productive



What does “Productivity” mean to you?

Recent Graduates:

“something similar to what I used in school: Python, Matlab, Java, ...”

Seasoned HPC Programmers:

“that sugary stuff that I don’t need because I ~~was born to suffer~~”

want full control to ensure performance”

Computational Scientists:

“something that lets me focus on my science 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.”

Comparing Chapel to Other Languages

CRAY
a Hewlett Packard Enterprise company

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]

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,
CHI UW 2018 keynote:
*Why Languages Matter
More Than Ever*]

Outline

- ✓ Context and Motivation
- Chapel and Productivity
 - A Brief Tour of Chapel Features
 - Summary and Resources

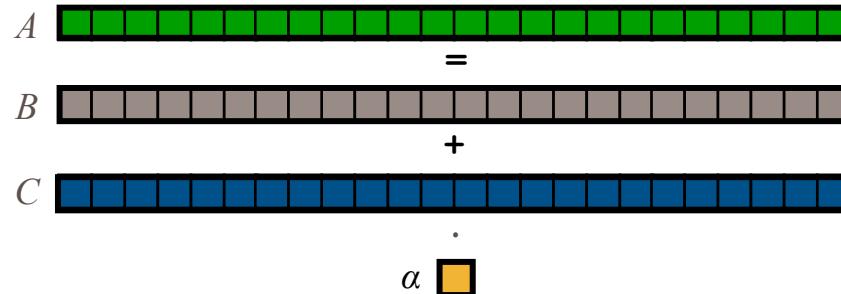


STREAM Triad: a trivial parallel computation

Given: m -element vectors A, B, C

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

In pictures:

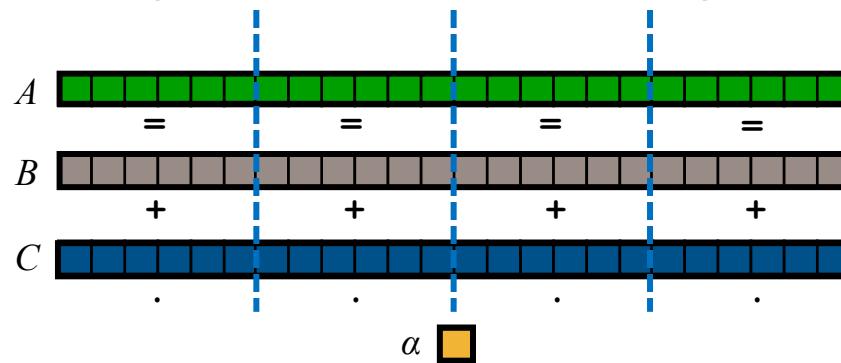


STREAM Triad: a trivial parallel computation

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):

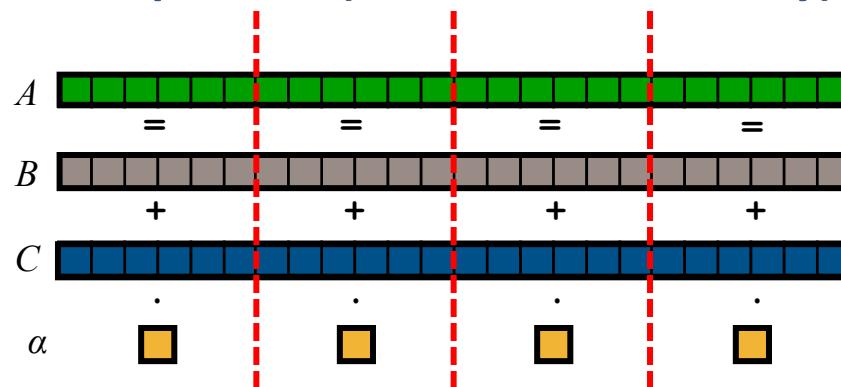


STREAM Triad: a trivial parallel computation

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):

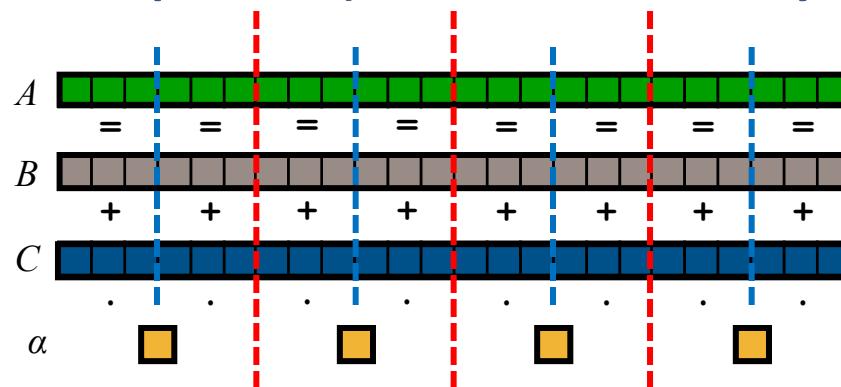


STREAM Triad: a trivial parallel computation

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):



STREAM Triad: C + MPI

```
#include <hpcc.h>

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

int HPCC_StarStream(HPCC_Parms *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_Parms *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;
}
```



STREAM Triad: C + MPI + OpenMP

```
#include <hpcc.h>
#ifndef _OPENMP
#include <omp.h>
#endif

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

int HPCC_StarStream(HPCC_Parms *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_Parms *params, int doIO) {
    register int j;
    double scalar;

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

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

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

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

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

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

    return 0;
}
```



STREAM Triad: Chapel

```
#include <hpcc.h>
#ifndef _OPENMP
#include <omp.h>
#endif

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

int HPCC_StarStream(HPCC_Parms *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_IN
```

```
)
```

```
        if (!a || !b || !c) {
            use ...;

            config const m = 1000,
                        alpha = 3.0;

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

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

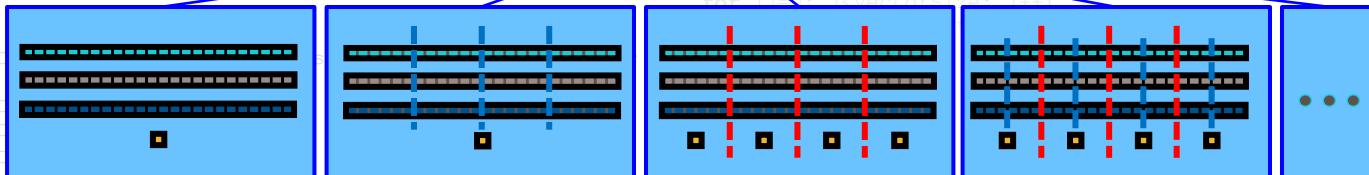
            B = 2.0;
            C = 1.0;

            A = B + alpha * C;
```

```
int HPCC_Stream(HPCC_Parms *params, int doIO) {
    register int j;
    double scalar;
    VectorSize = HP
    a = HPCC_XMALLOC
    b = HPCC_XMALLOC
    c = HPCC_XMALLOC
}
```

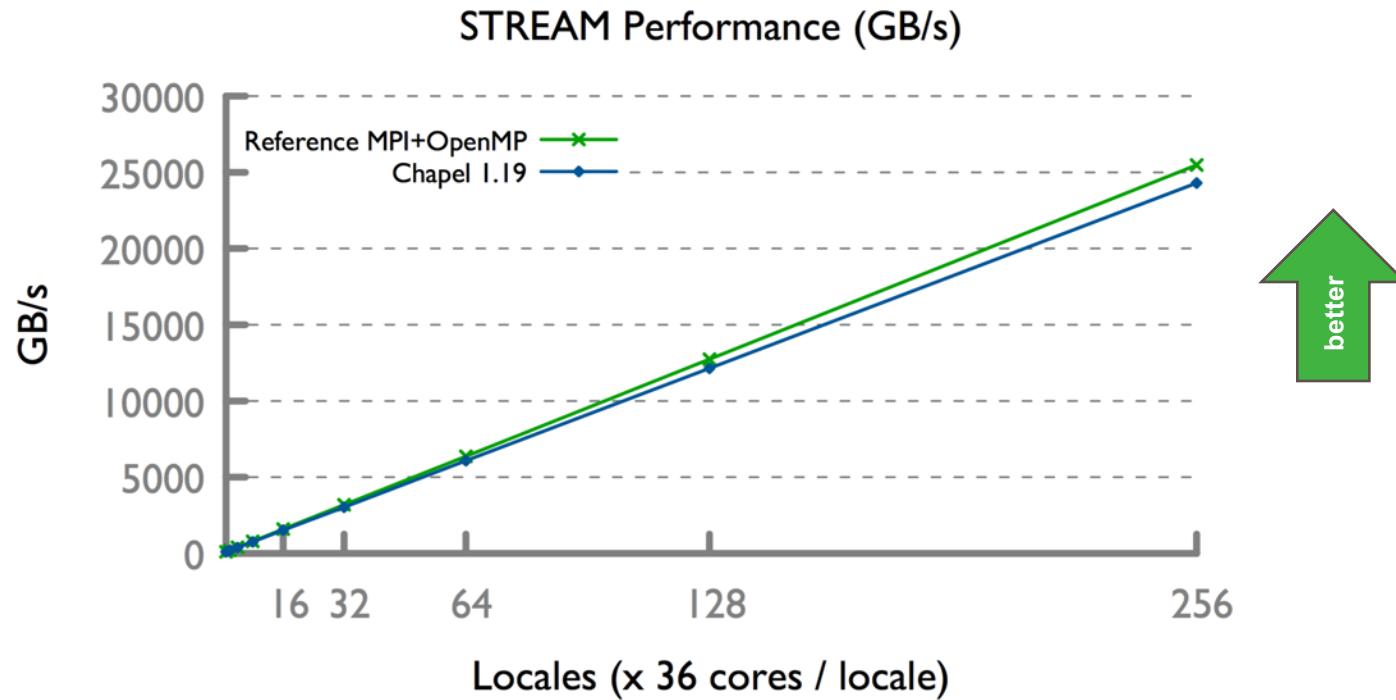
```
#pragma OPENMP for
#endif
for (j=0; j < VectorSize; j++) {
    scalar = 0.0;
    for (i=0; i < m; i++)
        scalar += a[i] * b[i];
    c[i] = scalar;
}
```

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



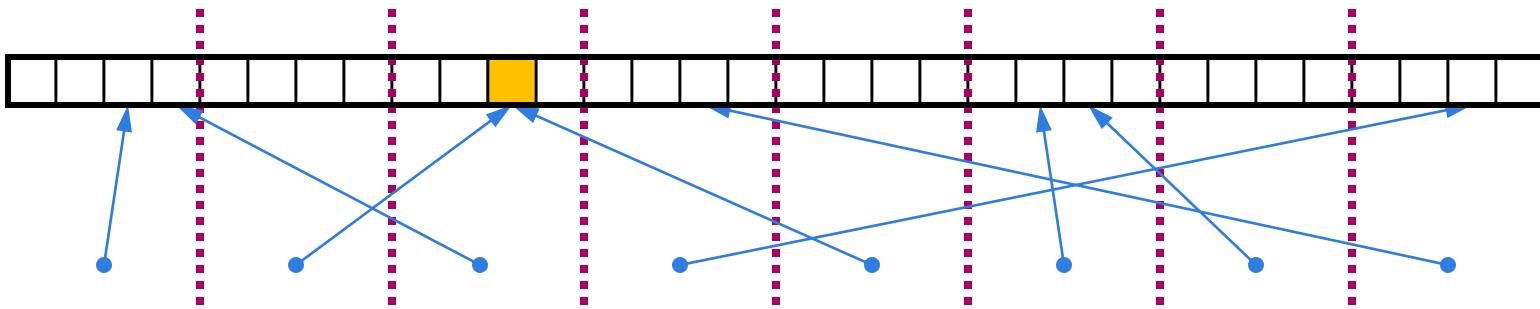
HPCC STREAM Triad: Chapel vs. C+MPI+OpenMP

CRAY
a Hewlett Packard Enterprise company



HPCC Random Access (RA)

Data Structure: distributed table



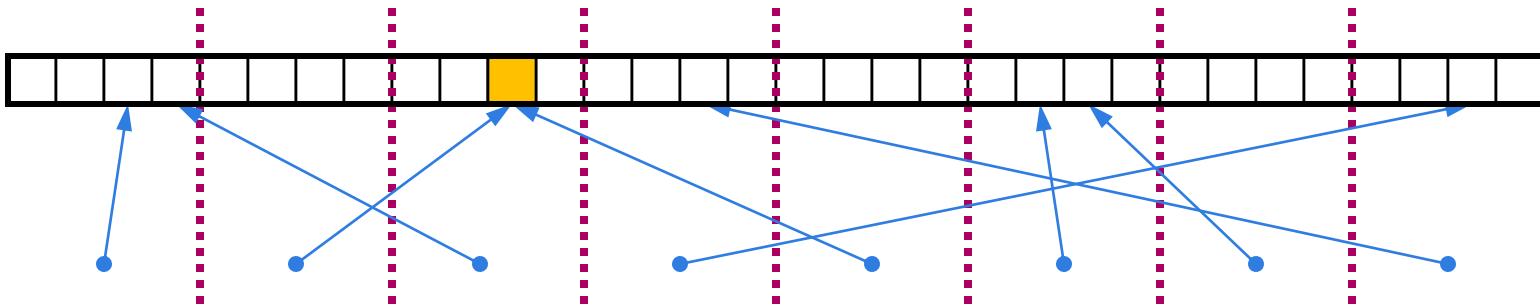
Computation: update random table locations in parallel

Two variations:

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

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

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_NUL;
        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_NUL;
    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_NUL; 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: MPI kernel comment vs. Chapel

```
/* 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 & (TABSIZ
```

Chapel Kernel

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

MPI Comment

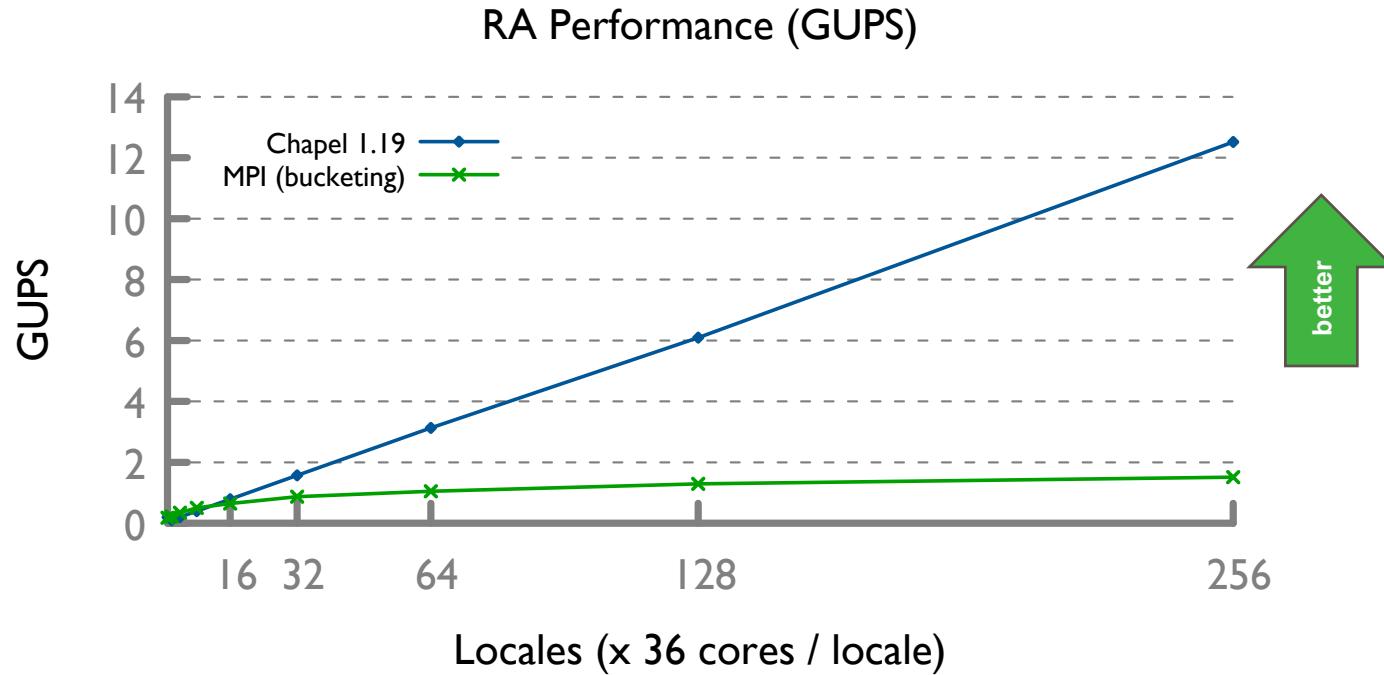
```
/* Perform updates to main table. The scalar equivalent is:
*
*   for (i=0; i<NUPDATE; i++) {
*     Ran = (Ran << 1) ^ (((s64Int) Ran < 0) ? POLY : 0);
*     Table[Ran & (TABSIZ
```

```
statuses;
```



HPCC RA: Chapel vs. C+MPI

CRAY
a Hewlett Packard Enterprise company



HPCC RA: MPI vs. Chapel

```
/* 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,
          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) ) + 1;
      else
        WhichPe = ( (GlobalOffset - tparams.Remainder) /
                    tparams.MinlocalTableSize );
      if (WhichPe == tparams.MyProc) {
        LocalOffset = (Ran & (tparams.TableSize - 1)) -
                      tparams.GlobalStartMyProc;
        HPCC_Table[LocalOffset] ^= Ran;
      }
    }
  }
```

Chapel Kernel

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

```
/* Done with our done messages */
if (proc_count > 0 & proc_count < tparams.NumProcs & t+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);

/* Done everyone else up... */
if (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: MPI vs. Chapel

```

/* 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);
    }
  }
}
  
```

Chapel Kernel

```

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

```

      pendingUpdates);
      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);
  }
  /* 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);
}
  
```



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

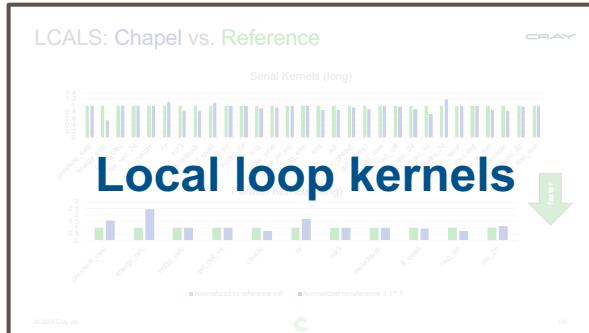
- Language defines what is easy and hard
- Influences algorithmic thinking

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

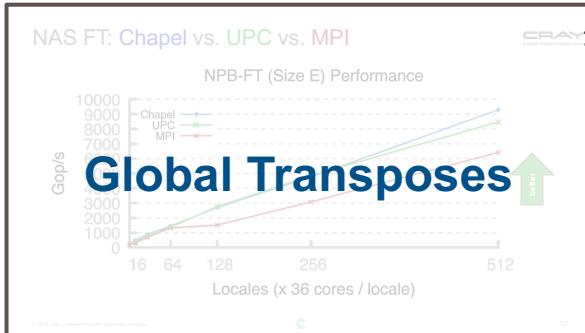
HPC Patterns: Chapel vs. Reference

CRAY
a Hewlett Packard Enterprise company

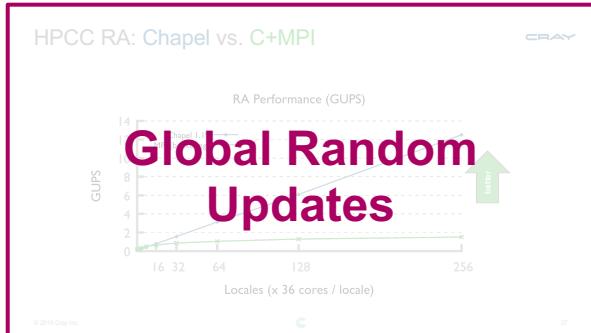
LCALS



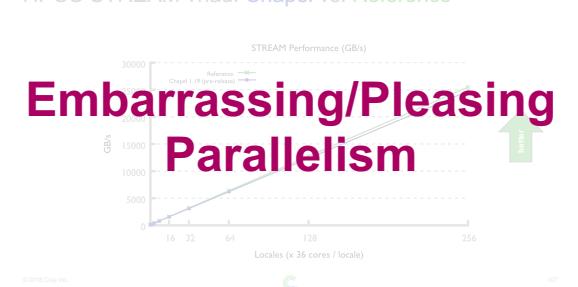
NAS FT



HPCC RA



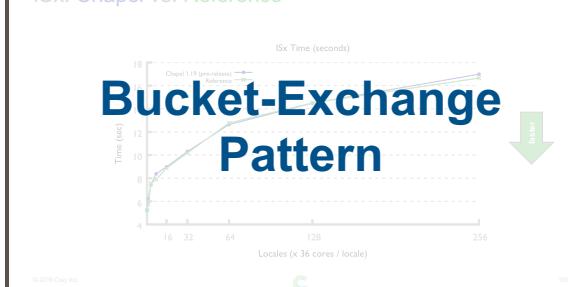
HPCC STREAM Triad: Chapel vs. Reference



STREAM Triad

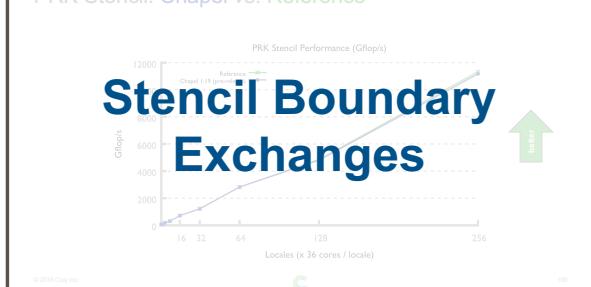
© 2020 Cray, a Hewlett Packard Enterprise company

ISx: Chapel vs. Reference



ISx

PRK Stencil: Chapel vs. Reference

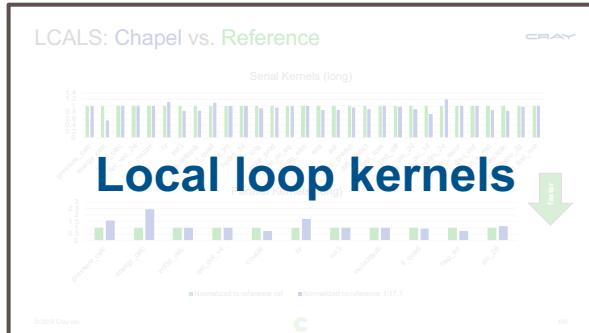


PRK Stencil

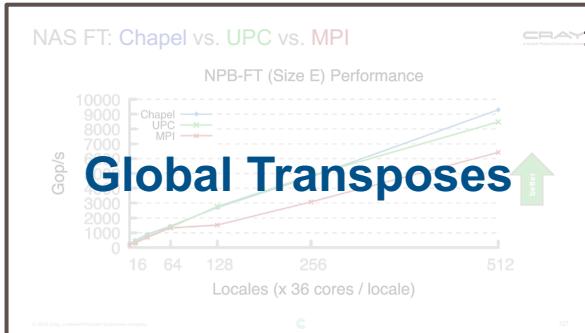
HPC Patterns: Chapel vs. Reference

CRAY
a Hewlett Packard Enterprise company

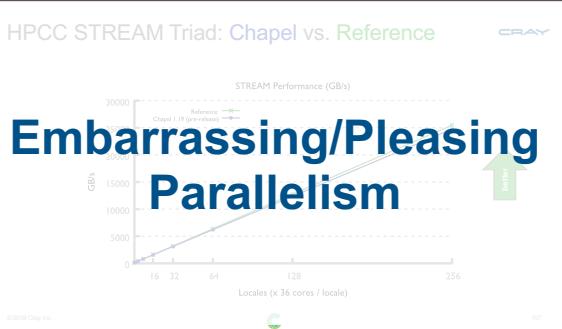
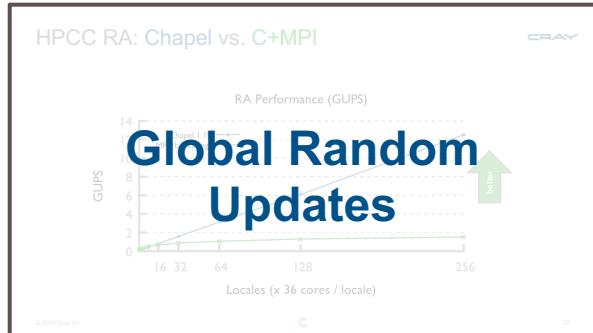
LCALS



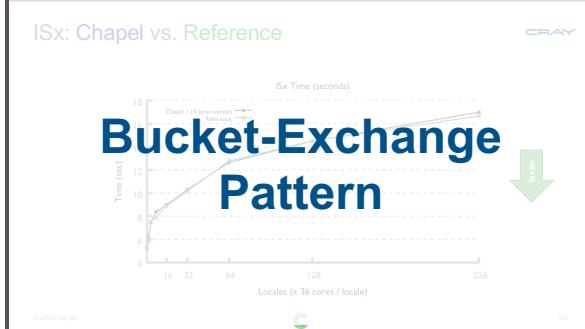
NAS FT



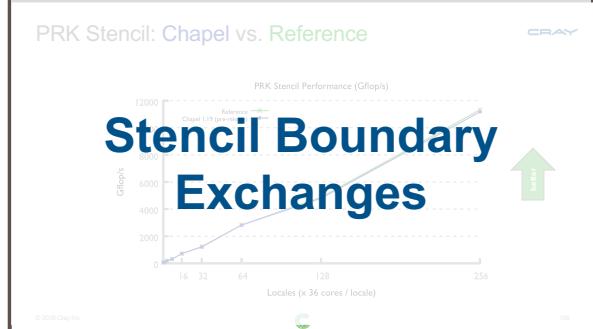
HPCC RA



STREAM Triad



ISx

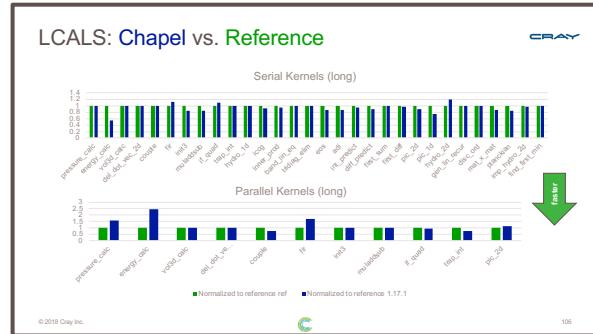


PRK Stencil

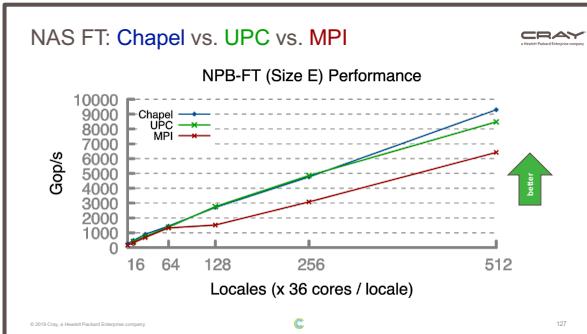
HPC Patterns: Chapel vs. Reference

CRAY
a Hewlett Packard Enterprise company

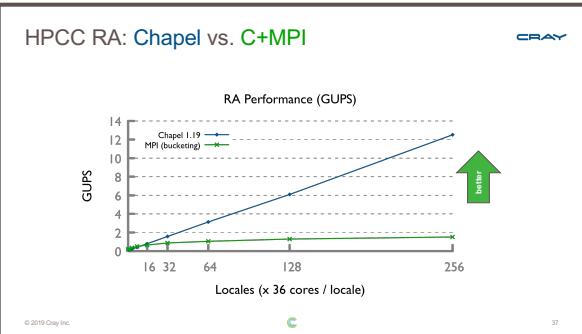
LCALS



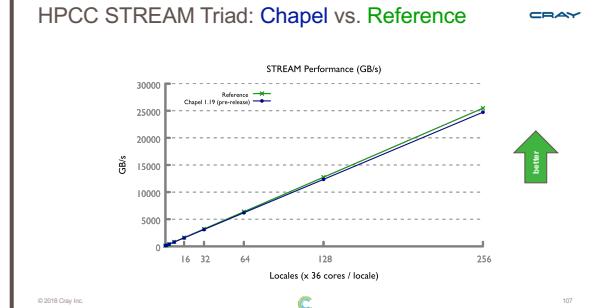
NAS FT



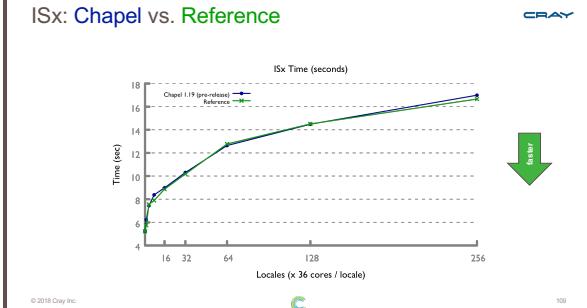
HPCC RA



HPCC STREAM Triad: Chapel vs. Reference



ISx: Chapel vs. Reference



STREAM Triad

© 2020 Cray, a Hewlett Packard Enterprise company

ISx

More on Chapel performance online at:
<https://chapel-lang.org/performance.html>

PRK Stencil

25



Notable Applications of Chapel

CRAY
a Hewlett Packard Enterprise company

ChplUltra: Simulating Ultralight Dark Matter
Nikhil Padmanabhan et al.
Yale University



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



CHGL: Chapel Hypergraph Library
Jesun Firoz, Cliff Joslyn, et al.
PNNL



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



ChOp: Chapel-based Optimization
Tiago Carneiro, Noureddine Melab, et al.
INRIA Lille, France



CrayAI: Distributed Machine Learning
Cray, a Hewlett Packard Enterprise Company

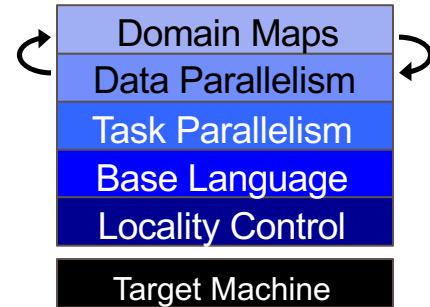
For more information, see: <https://chapel-lang.org/poweredsby.html>

A Brief Tour of Chapel Features

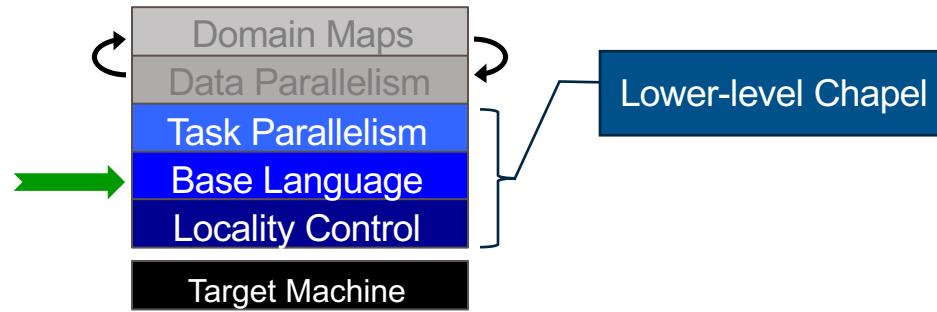


Chapel Feature Areas

Chapel language concepts



Base Language



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

```
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);
```

Configurable declarations
(support command-line overrides)
.fib --n=1000000

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

Base Language Features, by example

Iterators

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

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  
...
```

Base Language Features, by example

Explicit types also supported

```
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  
...
```

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

```
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);
```

Zippered iteration

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

Base Language Features, by example

Range types and operators

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

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

```
config const n = 10;

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

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

Base Language Features, by example

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

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

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
...
```

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;

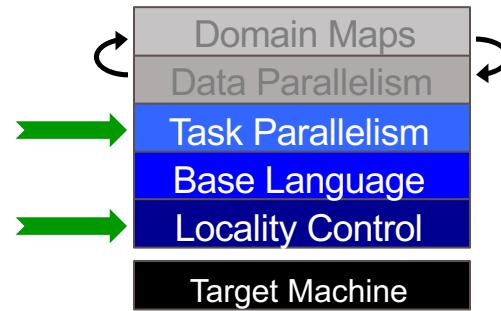
for (i,f) in zip(0..<n, fib(n)) do
    writeln("fib #", i, " is ", f);
```

```
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
...
...
```

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...

Task Parallelism and Locality Control



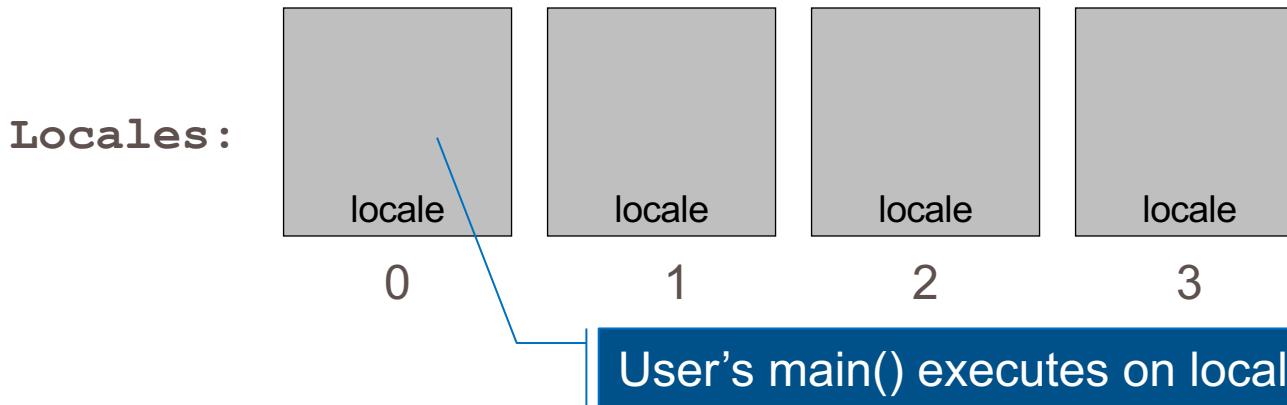
Locales in Chapel

- Locales can run tasks and store variables

- Think “compute node”

- Number of locales specified on executable’s command-line

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



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
```

Task Parallelism and Locality, by example

Abstraction of
System Resources

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
```

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
```

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
```

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
```

Task Parallelism and Locality, by example

Abstraction of
System Resources

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
```

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
```

Task Parallelism and Locality, by example

Control of Locality/Affinity

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
```

Task Parallelism and Locality, by example

taskParallel.chpl

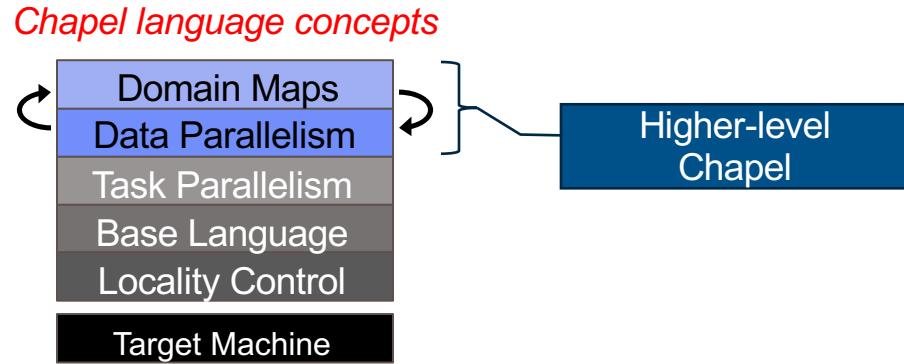
```
coforall loc in Locales do
    on loc {
        const numTasks = here.numPUs();
        coforall tid in 1..numTasks do
            writef("Hello from task %n of %n "+
                "running on %s\n",
                tid, numTasks, here.name);
    }
```

```
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
```

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 having tasks refer to variables

Data Parallelism in Chapel



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
```

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
```

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
```

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
```

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
```

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
```

Distributed Data Parallelism, by example

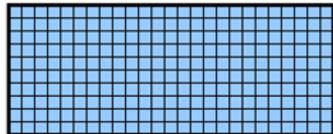
dataParallel.chpl

```
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);
```

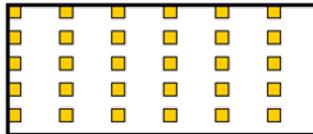
```
prompt> chpl dataParallel.chpl
prompt> ./dataParallel --n=5 --numLocales=4
1.1 1.3 1.5 1.7 1.9
2.1 2.3 2.5 2.7 2.9
3.1 3.3 3.5 3.7 3.9
4.1 4.3 4.5 4.7 4.9
5.1 5.3 5.5 5.7 5.9
```

Other Data Parallel Features

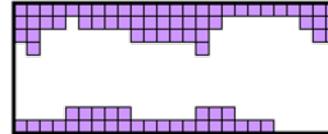
- **Parallel Iterators and Zippering**
- **Slicing:** refer to subarrays using ranges / domains
- **Promotion:** execute scalar functions in parallel using array arguments
- **Reductions:** collapse arrays to scalars or subarrays
- **Scans:** parallel prefix operations
- **Several Domain/Array Types:**



dense



strided



sparse



associative

Summary and Resources



Summary

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

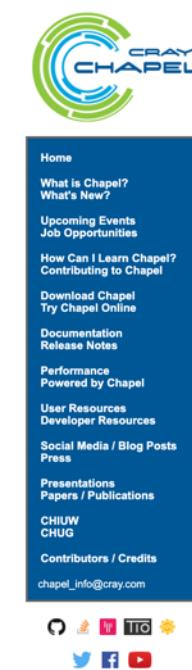
Chapel is attractive to computational scientists and Python programmers

Chapel Homepage

CRAY
a Hewlett Packard Enterprise company

<https://chapel-lang.org>

- downloads
- presentations
- papers
- resources
- documentation



The Chapel Parallel Programming Language

What is Chapel?

Chapel is a programming language designed for productive parallel computing at scale.

Why Chapel? Because it simplifies parallel programming through elegant support for:

- distributed arrays that can leverage thousands of nodes' memories and cores
- a global namespace supporting direct access to local or remote variables
- data parallelism to trivially use the cores of a laptop, cluster, or supercomputer
- task parallelism to create concurrency within a node or across the system

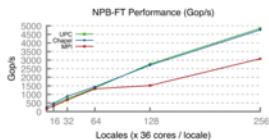
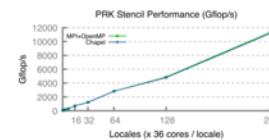
Chapel Characteristics

- productive: code tends to be similarly readable/writable as Python
- scalable: runs on laptops, clusters, the cloud, and HPC systems
- fast: performance **competes with or beats** C/C++ & MPI & OpenMP
- portable: compiles and runs in virtually any *nix environment
- open-source: hosted on [GitHub](#), permissively licensed

New to Chapel?

As an introduction to Chapel, you may want to...

- watch [an overview talk](#) or browse its [slides](#)
- read a [blog-length](#) or [chapter-length](#) introduction to Chapel
- learn about [projects powered by Chapel](#)
- check out [performance highlights](#) like these:



- browse [sample programs](#) or learn how to write distributed programs like this one:

```
use CyclicDist;           // use the Cyclic distribution Library
config const n = 100;      // use --n=<n> when executing to override this default

forall i in {1..n} dmapped Cyclic(startIdx=1) do
    writeln("Hello from iteration ", i, " of ", n, " running on node ", here.id);
```

© 2020 Cray, a Hewlett Packard Enterprise company



64

Chapel Documentation

CRAY
a Hewlett Packard Enterprise company

<https://chapel-lang.org/docs>: ~270 pages, including primer examples

The screenshot displays the Chapel Documentation website with a dark theme. The main navigation bar includes a logo, the text "Chapel Documentation", "version 1.22", a search bar, and a "View page source" link. The left sidebar contains sections for "COMPILE AND RUNNING CHAPEL" (Quickstart Instructions, Using Chapel, Platform-Specific Notes, Technical Notes, Tools), "WRITING CHAPEL PROGRAMS" (Quick Reference, Hello World Variants, Primers, Language Specification, Built-in Types and Functions, Standard Modules, Package Modules, Standard Layouts and Distributions, Chapel Users Guide (WIP)), and "LANGUAGE HISTORY" (Chapel Evolution, Documentation Archives). The main content area shows the "Chapel Documentation" page, followed by "Compiling and Running Chapel" (with a list of sub-topics like Quickstart Instructions, Using Chapel, etc.), "Writing Chapel Programs" (with a list of sub-topics like Quick Reference, Hello World Variants, Primers, etc.), "Language History" (with a list of sub-topics like Chapel Evolution, Documentation Archives), "Using Chapel" (with a list of sub-topics like Chapel Prerequisites, Setting up Your Environment for Chapel, Building Chapel, etc.), and finally "chpl" (with sections for SYNOPSIS, DESCRIPTION, SOURCE FILES, OPTIONS, ENVIRONMENT, BUGS, SEE ALSO, AUTHORS, and a detailed command-line reference). The footer includes a copyright notice for 2020, Hewlett Packard Enterprise, and a small logo.



Chapel Social Media (no account required)

CRAY
a Hewlett Packard Enterprise company



Chapel Language

@ChapelLanguage

A productive parallel programming language designed to scale from supercomputers whose development is being led by @HPE (former

 chapel-lang.org  Joined February 2016

97 Following 456 Followers

[Tweets](#)

[Tweets & replies](#)

[Media](#)

 Pinned Tweet

Chapel Language @ChapelLanguage · 4h
We're just a week away from CHI UW 2020, representing t





Chapel
Programming
Language
@ChapelLanguage

Home

Posts

Videos

Photos

About

Community

Events

Create a Page

The screenshot shows the YouTube channel page for 'Chapel language'. The channel has 144 subscribers and features a logo with a stylized green 'C' and 'P'. The main video thumbnail is titled 'Chapel Comes of Age: a Language for Productivity, Parallelism, and Performance' by Brad Chamberlain (Day). Below it is another thumbnail for 'CHILW 2017 keynote: Chapel's Home in the New Landscape of Scientific Frameworks, Jonathan Dursi'. A third thumbnail at the bottom is titled 'The Audacity of Chapel: Scalable Parallel Programming Done Right - Brad Chamberlain - T-Thread 2017'. On the left sidebar, there are links for Home, Trending, Subscriptions, Library, History, Your videos, Watch later, and Liked videos. The 'Subscriptions' section lists 'Popular on YouTube' with categories for Music, Sports, and Gaming. The 'More from YouTube' section includes links for YouTube Premium, Movies & Shows, and more.

© 2020 Cray, a Hewlett Packard Enterprise company



Chapel Community

CRAY
a Hewlett Packard Enterprise company

Questions Developer Jobs Tags Users [chapel]

Tagged Questions info newest frequent votes active unanswered

Chapel is a portable, open-source parallel programming language. Use this tag to ask questions about the Chapel language or its implementation.

Learn more... Improve tag info Top users Synonyms

6 Tuple Concatenation in Chapel
Let's say I'm generating tuples and I want to concatenate them as they come. How do I do this? The following does element-wise addition: if `ts = ("foo", "cat"), t = ("bar", "dog") ts += t` gives `ts = ...`
tuples concatenation addition hpc chapel asked Jan 26 at 0:30 by Tashmanga 385 1 10

6 Is there a way to use non-scalar values in functions with where clauses in Chapel?
I've been trying out Chapel off and on over the past year or so. I have used C and C++ briefly in the past, but most of my experience is with dynamic languages such as Python, Ruby, and Erlang more ...
chapel asked Apr 23 at 23:15 by angus 33 3

47 views

6 Is there any writef() format specifier for a bool?
I looked at the writef() documentation for any bool specifier and there didn't seem to be any. In a Chapel program I have: ... config const verify = false; /* that works but I want to use writef() ...
chapel asked Nov 11 '17 at 22:21 by casselle

6 answers

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

This repository Search Pull requests Issues Marketplace Gist

chapel-lang / chapel

Code Issues 292 Pull requests 26 Projects 0 Settings Insights

Filters is:issue is:open Labels Milestones

292 Open 77 Closed

Implement "bounded-coforall" optimization for remote coforalls area: Compiler type: Performance #6357 opened 13 hours ago by ronawho

Consider using processor atomics for remote coforalls EndCount area: Compiler type: Performance #6366 opened 13 hours ago by ronawho 0 of 6

make uninstall area: BTR type: Feature Request #6363 opened 14 hours ago by mpff

make check doesn't work with ./configure area: BTR #6362 opened 16 hours ago by mpff

Passing variable via intent to a forall loop seems to create an iteration-private variable, not a task-private one area: Compiler type: Bug #6351 opened a day ago by casselle

Remove chpl_comm_make_progress area: Runtime easy type: Design #6349 opened a day ago by sunghunchoi

Runtime error after make on Linux Mint area: BTR user issue #6348 opened a day ago by deninanda

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

GITTER

chapel-lang/chapel Chapel programming language | Peak developer hours are 0600-1700 PT

Where communities thrive

FREE FOR COMMUNITIES

JOIN OVER 800+ PEOPLE JOIN OVER 80K COMMUNITIES CREATE YOUR OWN COMMUNITY EXPLORE MORE COMMUNITIES

Brian Dolan @buddha314 what is the syntax for making a copy (not a reference) to an array? May 09 14:34

Michael Ferguson @mpff like in a new variable? May 09 14:40

var A[1..10] int;
var B = A; // makes a copy of A
ref C = A; // refers to A

Brian Dolan @buddha314 oh, got it, thanks! May 09 14:41

Michael Ferguson @mpff proc g(in arr) { /* arr is a copy of the actual argument */ } var A[1..10] int;
f(A);
g(A);

Brian Dolan @buddha314 isn't there a proc fref arr { ... } as well? May 09 14:43

Michael Ferguson @mpff yes. The default intent for array is 'ref' or 'const ref' depending on if the function body modifies it. So that's effectively the default.

Brian Dolan @buddha314 thanks! May 09 14:55

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

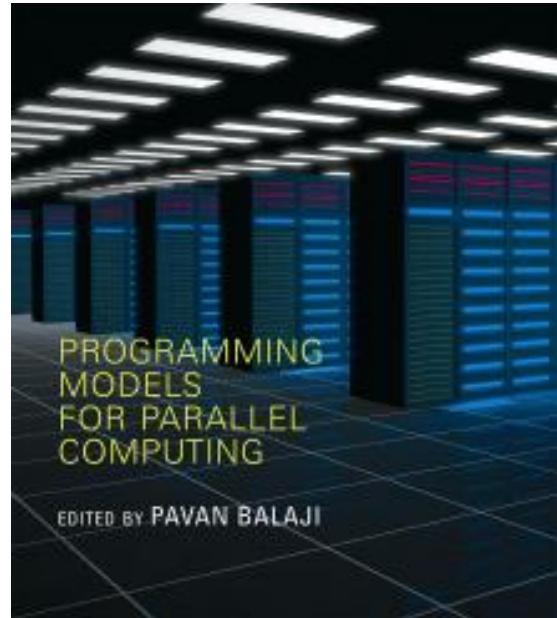
read-only mailing list: chapel-announce@lists.sourceforge.net (~15 mails / year)

Suggested Reading: Historical Overview

CRAY
a Hewlett Packard Enterprise company

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](#)



Suggested Reading: Mid-project Progress (2013–2018)

CRAY
a Hewlett Packard Enterprise company

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. In years since the DARPA High Productivity Computing Systems (HPCS) program that launched Chapel wrapped up, and the team embarked on a five-year effort to implement Chapel's API and to end-users. This paper follows the evolution of Chapel's performance and productivity made by the Chapel project since that time. Specifically, Chapel's performance now competes with or beats hand-coded C/PUSH/PULL. Its suite of distributed libraries has grown to include PETSc, MPI, LAMMPS, 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 researchers 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 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 various 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>

paper and slides available at chapel-lang.org



The collage consists of 16 square images arranged in a 4x4 grid. The top row shows the Cray logo, a city skyline with a tall spire, and three blank blue squares. The middle row shows a city skyline with a church tower reflected in water, followed by four more images of the same scene. The bottom row shows a close-up of the water's surface with reflections, followed by four more images of the water and skyline.

**Chapel Comes of Age:
Productive Parallelism at Scale** 
CUG 2018
Brad Chamberlain, Chapel Team, Cray Inc.

Suggested Reading: The Very Latest

CRAY
a Hewlett Packard Enterprise company

- Chapel release notes: <https://chapel-lang.org/releaseNotes.html>

The screenshot shows the Chapel website's "Release Notes" page. At the top left is the CRAY CHAPEL logo. To its right is a blue header bar with the text "The Chapel Parallel Programming Language". Below the header is a section titled "Release Notes". A sidebar on the left contains links to "Home", "What is Chapel?", "What's New?", "Upcoming Events", "Job Opportunities", "How Can I Learn Chapel?", "Contributing to Chapel", "Download Chapel", "Try Chapel Online", "Documentation", "Release Notes", and "Performance". The main content area starts with a paragraph about the detailed release notes for Chapel 1.21 / 1.22, followed by a bulleted list of improvements. It then encourages users to refer to the "CHANGES.md" file. At the bottom is a link to "Archived Release Notes (for previous releases)".

The Chapel Parallel Programming Language

Release Notes

The following are the detailed release notes for Chapel 1.21 / 1.22:

- Language Improvements
- Library Improvements
- Interoperability Improvements
- Benchmarks and Performance Optimizations
- User Application Optimizations
- Ongoing Efforts
- Proposed Priorities for Chapel 1.23

For further information, you may also want to refer to the [CHANGES.md](#) file.

[Archived Release Notes \(for previous releases\)](#)



Summary

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 computational scientists and Python programmers

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

QUESTIONS?



bradc@cray.com



@ChapelLanguage



chapel-lang.org



cray.com



@cray_inc



linkedin.com/company/cray-inc-/

