



Chapel: Task-Based Communication in a Productive Language

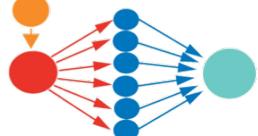
Elliot Ronaghan, Chapel Team, Cray Inc.

SIAM PP18

March 8 2018



SIAM Conference on
Parallel Processing
for Scientific Computing



COMPUTE

STORE

ANALYZE

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.



What is Chapel?



Chapel: A productive parallel programming language

- portable
- open-source
- a collaborative effort



Goals:

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



COMPUTE

|

STORE

|

ANALYZE

Chapel and Productivity



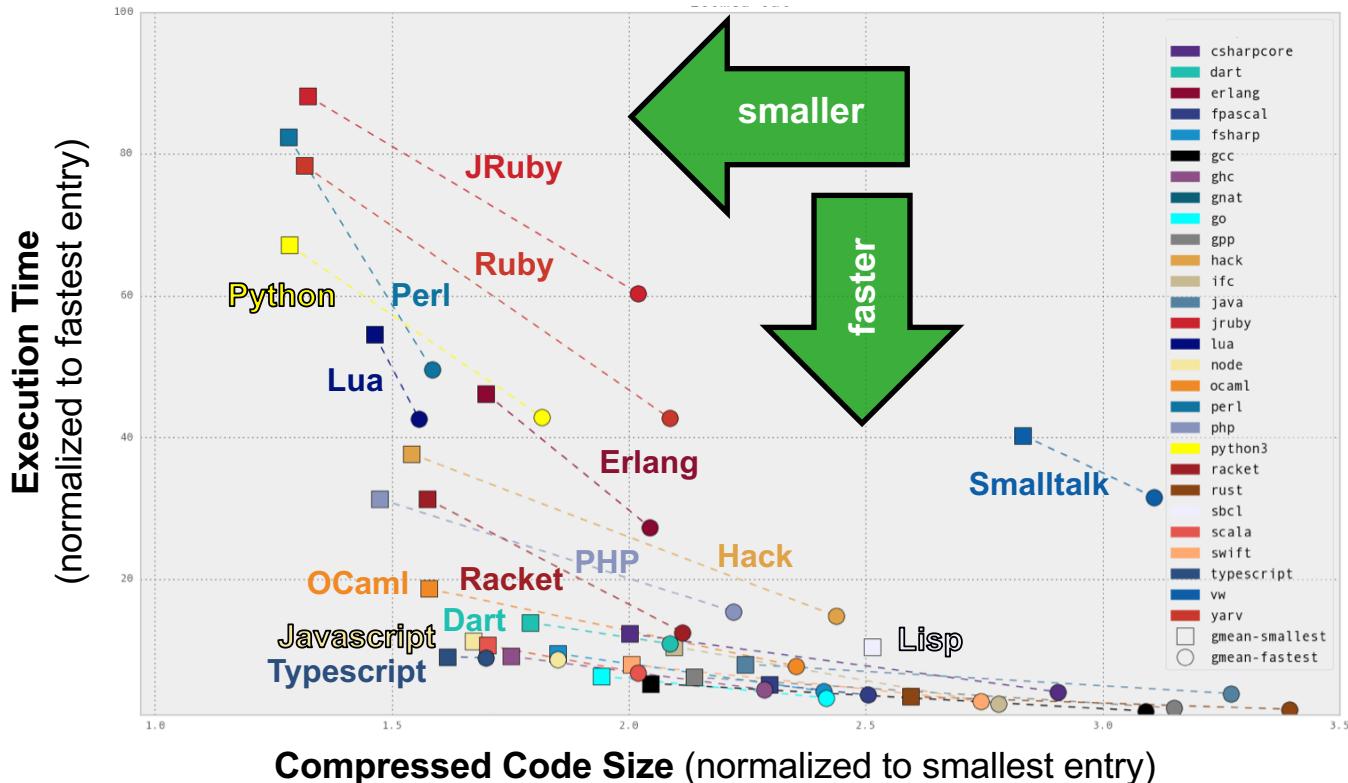
- **Chapel strives to be...**

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



CLBG Cross-Language Summary

(Oct 2017 standings)



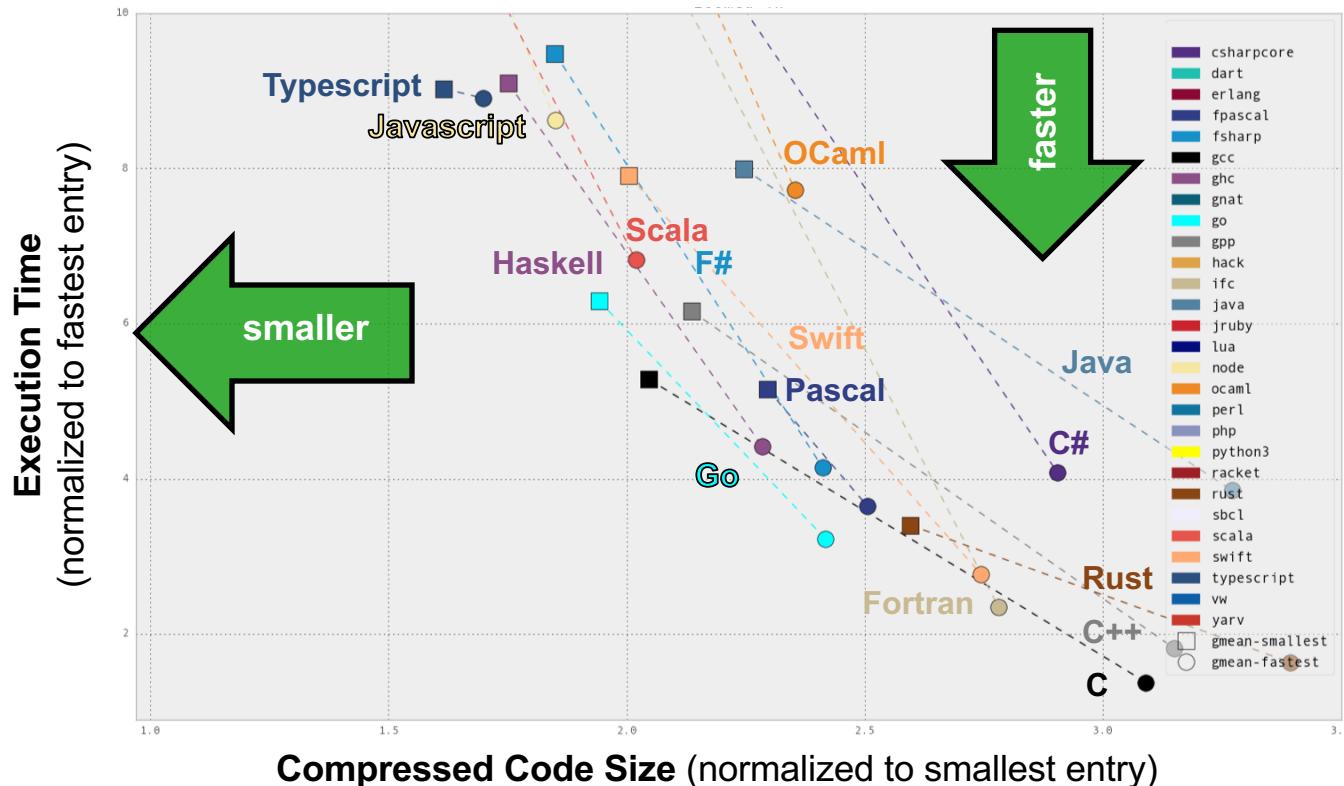
COMPUTE

STORE

ANALYZE

CLBG Cross-Language Summary

(Oct 2017 standings, zoomed in)



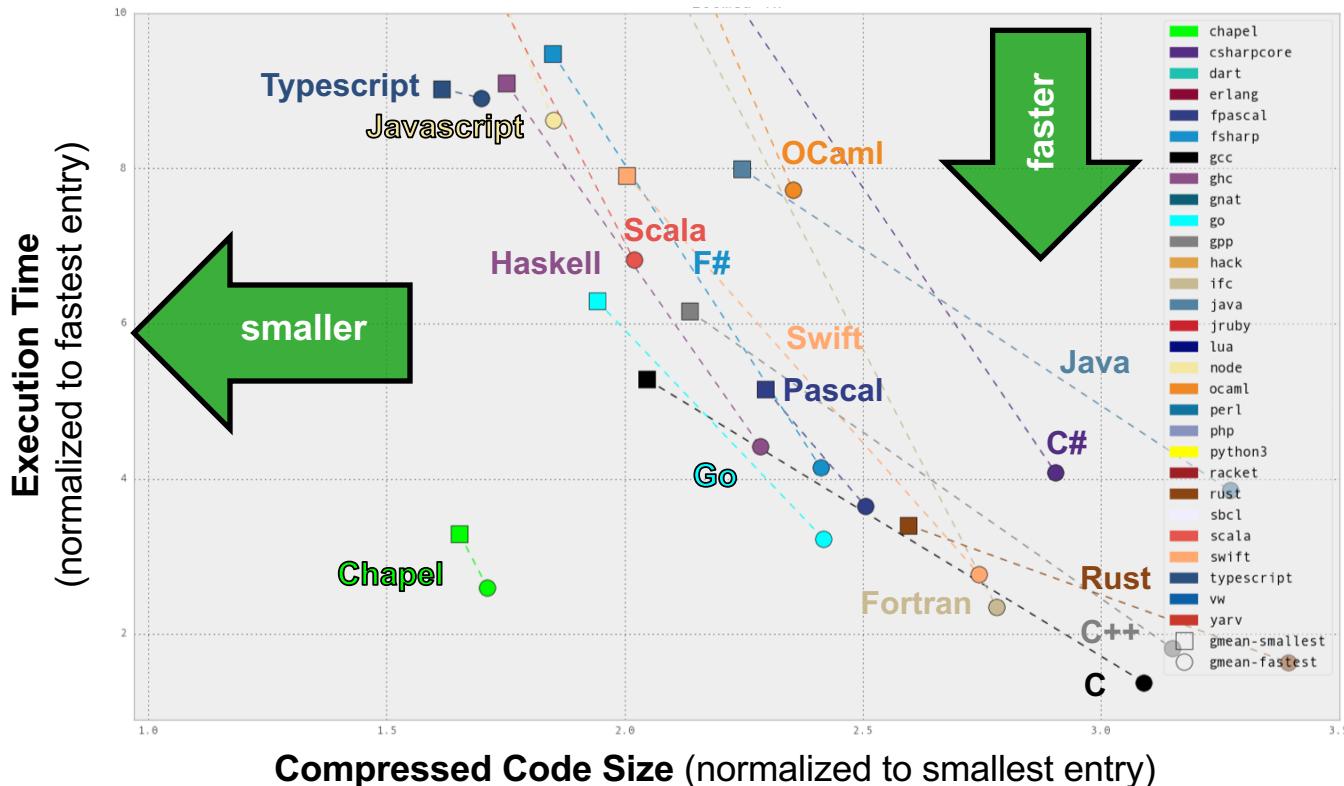
COMPUTE

STORE

ANALYZE

CLBG Cross-Language Summary

(Oct 2017 standings, zoomed in)



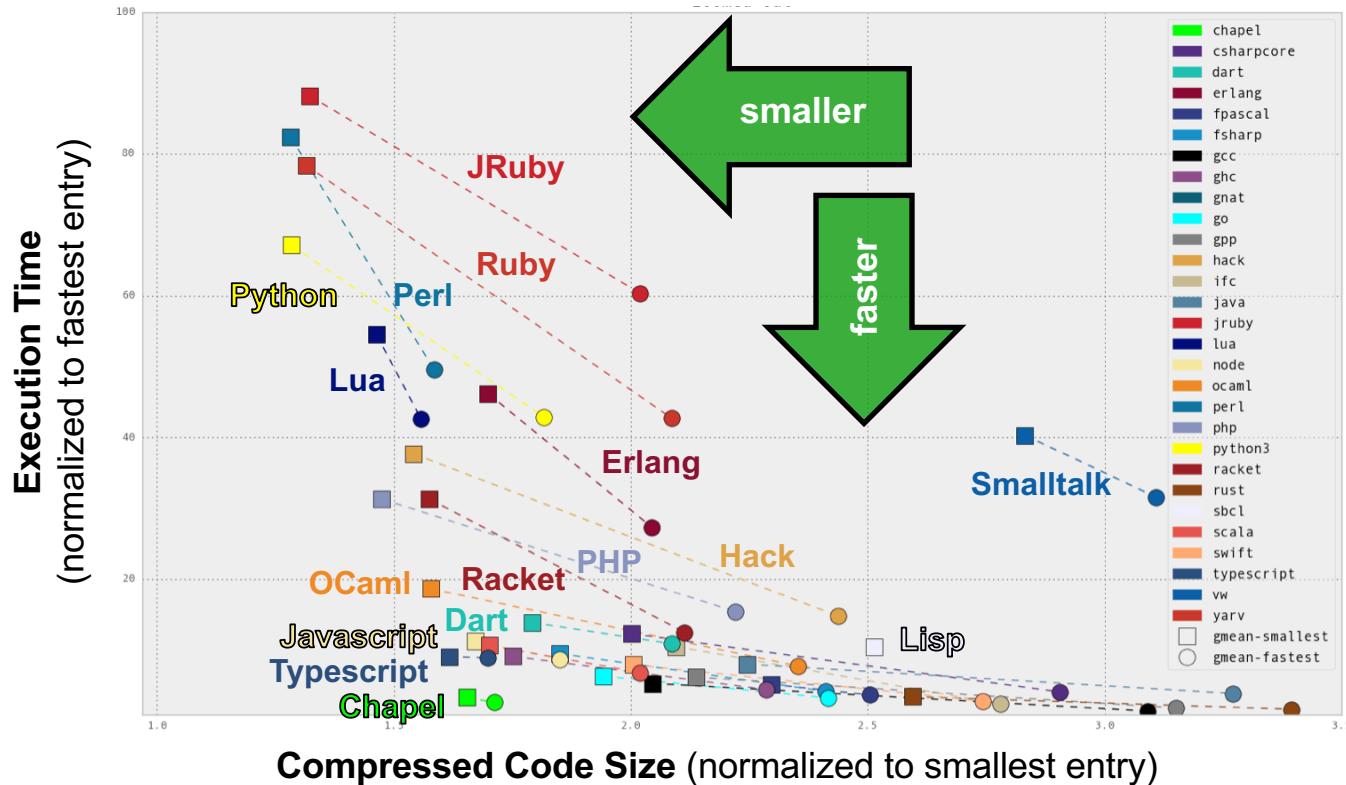
COMPUTE

STORE

ANALYZE

CLBG Cross-Language Summary

(Oct 2017 standings)

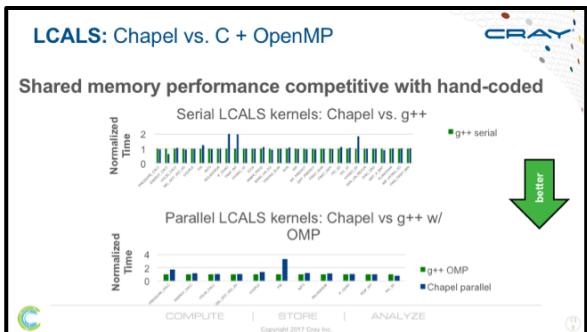
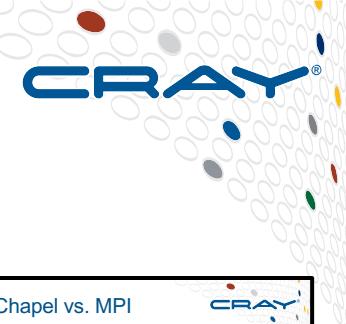


COMPUTE

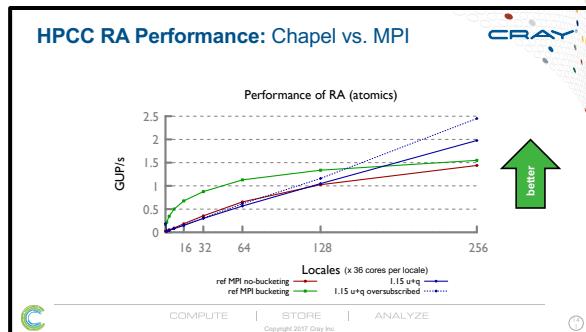
STORE

ANALYZE

Chapel Performance: HPC Benchmarks

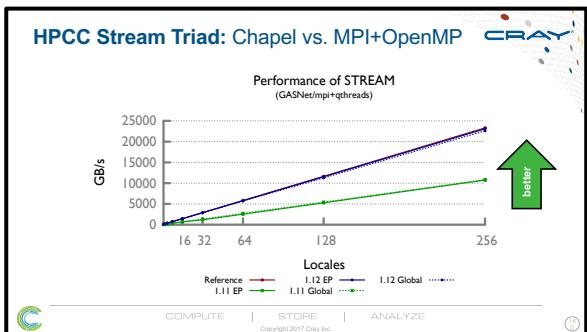


LCALS



STREAM Triad

PRK Stencil



ISx weakISO Total Time

Nodes	SHMEM (s)	Chapel (s)	MPI (s)
1	4.8	4.8	4.8
2	8.5	8.8	8.5
4	8.8	9.0	8.8
8	9.0	9.5	9.5
16	9.5	10.0	10.0
32	11.5	11.5	11.5
64	12.5	12.5	12.5

Stencil PRK Scalability

Stencil PRK Performance (weak scaling)

Locales	MPI+OpenMP (GFlops/s)	Chapel (GFlops/s)
16	~1000	~1000
32	~2500	~2500
64	~4000	~4000
128	~6500	~6500
256	~11000	~11500

better

COMPUTE

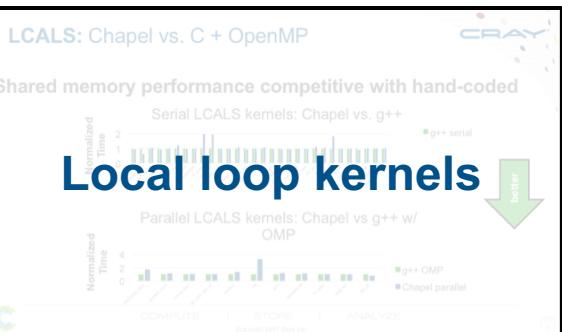
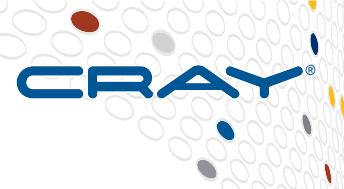
STORE

ANALYZE

Copyright 2018 Cray Inc.

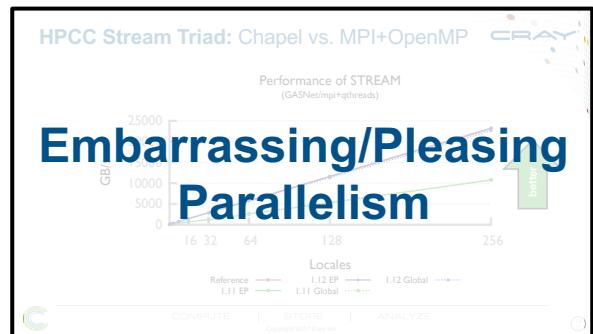
Nightly performance graphs online
at: <https://chapel-lang.org/perf>

Chapel Performance: HPC Benchmarks

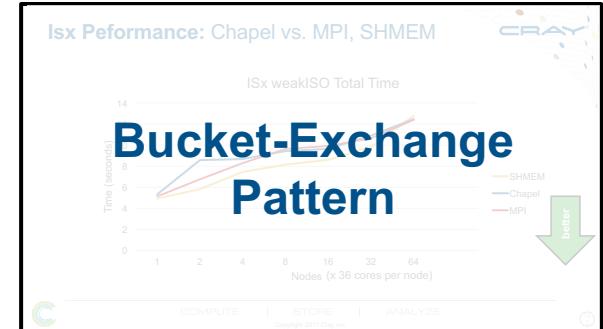


LCALS

HPCC RA

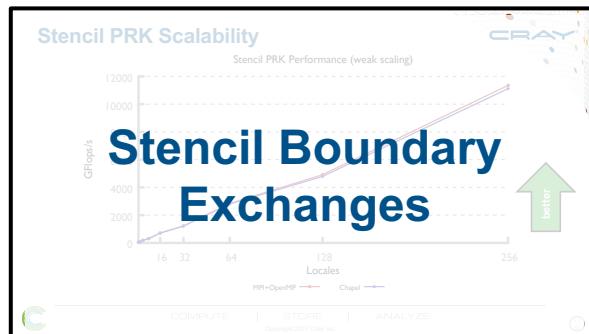
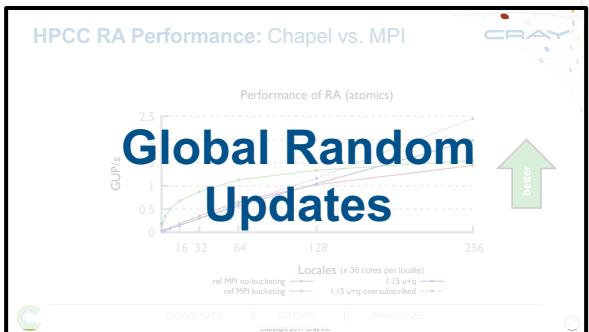


STREAM
Triad



ISx

PRK
Stencil



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Nightly performance graphs online
at: <https://chapel-lang.org/perf>



The Chapel Team at Cray (May 2017)



14 full-time employees + 2 summer interns + 2–4 GSoC students



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Chapel Community Partners



Lawrence Berkeley
National Laboratory



Yale

(and several others...)

<https://chapel-lang.org/collaborations.html>



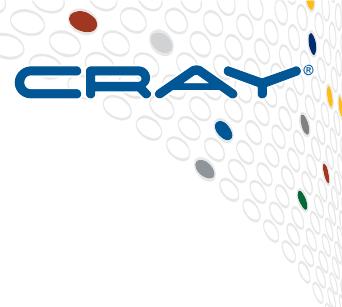
COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Plan for this talk



- Chapel by comparison: Random Access
- Runtime overview
- Performance optimizations enabled by runtime



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.



Chapel by Comparison



COMPUTE

|

STORE

|

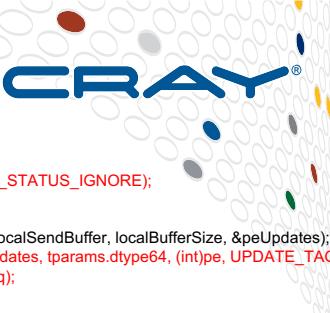
ANALYZE

- **Random Access (RA) benchmark**

- make random xor-updates to a distributed array of integers
- stresses fine-grained communication (in its purest form)
- benchmark allows up to 1% of updates to be missed/dropped



Random Access (GUPS) Kernel: MPI



```

/* 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-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 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);
}
} while (have_done && NumberReceiving > 0);

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

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

```



COMPUTE

STORE

ANALYZE

Random Access (GUPS) Kernel: Chapel



MPI Comment

```
MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
          MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
          MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
          MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);

while (i < Se
/* receive n
do {
  MPI_Test(
if (have_c
if (status
  MPI_G
bufferB
for (j=0;
  inmsg
  Local_
  HPCC
}
} else if (
/* we g
Number
} 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 {
```

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
     *      }
     */
}

MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
          MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq),
          MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
          MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);

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



COMPUTE

STORE

ANALYZE

Random Access (GUPS) Kernel: Chapel



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-1)] ^= Ran;  
*      }  
*/
```

Chapel Code

```
forall (_ , r) in zip(Updates, RASTream()) do
    T[r & indexMask] ^= 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-1)] ^= Ran;
 *      }
 * /
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 < ZEROTAG);
    GlobalOffset = Ran & (tparams.TableSize - 1);
    if (GlobalOffset < tparams.Top)
        WhichPe = (GlobalOffset / (tparams.MPISize));
    else
        WhichPe = ((GlobalOffset - tparams.Remainder) / tparams.MinLocalTableSize);
}

if (WhichPe == tparams.MyProc) {
    LocalOffset = (Ran & (tparams.TableSize - 1)) - tparams.GlobalStartMyProc;
    HPCC_Table[LocalOffset] ^= Ran;
}

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;
            if (inmsg < ZEROTAG) {
                MPI_Set_count(&status, tparams.dtype64, &recvUpdates);
                bufferBase = 0;
                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);
```

Chapel Code

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



Random Access (GUPS) Kernel: Chapel



Chapel RMO

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

```
MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
          MPI_ANY_SOURCE, MPI_ANY_TAG);

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) {
                /* 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 (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(&peUpdates, tparams.dtype64, (int)pe, UPDATE_TAG);

    MPI_Test(&outreq, &have_done, MPI_STATUS_IGNORE);
    localSendBuffer, localBufferSize, &peUpdates);
    updates, tparams.dtype64, (int)pe, UPDATE_TAG);
}

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_Waitall( tparams.NumProcs, tparams.finish_req, tparams.finish_statuses);

    /* send our done messages */
    for (proc_count = 0; proc_count < tparams.NumProcs ; ++proc_count) {
        if (proc_count == tparams.MyProc) { tparams.finish_req[params.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);
}
```



COMPUTE

|

STORE

|

ANALYZE

Random Access (GUPS) Kernel: Chapel



Chapel RMO

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

Chapel On-stmt

```
forall (_, r) in zip(Updates, RASTream()) do
    on TableDist.idxToLocale[r & indexMask] do
        T[r & indexMask] ^= r;
```

```
MPI_Irecv(&LocalRecvBuffer, localBufferSize, &params.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) {
                /* we got a done message. Thanks for playing... */
                NumberReceiving--;
            } else {
                MPI_Abort(MPI_COMM_WORLD, -1 );
            }
            MPI_Irecv(&LocalRecvBuffer, localBufferSize,
                      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 {

    MPI_Irecv(&LocalRecvBuffer, localBufferSize, &params.dtype64,
              MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
    MPI_Irecv(&LocalRecvBuffer, localBufferSize, &peUpdates,
              MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
    MPI_Test(&inreq, &have_done, MPI_STATUS_IGNORE);

    localSendBuffer, localBufferSize, &peUpdates);
    updates, tparams.dtype64, (int)pe, UPDATE_TAG);
}

/* 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_Wait(&finish_req[proc_count], MPI_STATUS_IGNORE);
    proc_count, FINISHED_TAG,
    finish_req + proc_count);
}

MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
          MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);

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);
```



COMPUTE

STORE

ANALYZE

Random Access (GUPS) Kernel: Chapel



Chapel RMO

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

Chapel On-stmt

```
forall (_, r) in zip(Updates, RASTream()) do
    on TableDist.idxToLocale[r & indexMask] do
        T[r & indexMask] ^= r;
```

Chapel Atomic

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

```
MPI_Irecv(&LocalRecvBuffer, localBufferSize, &params.dtype64,
          MPI_ANY_SOURCE, MPI_ANY_TAG,
```

```
HPCC_InsertUpdate(Rastream);
```

```
MPI_Test(&outtag, &have_done, MPI_STATUS_IGNORE);
```

```
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) {
                /* we got a done message. Thanks for
                   NumberReceiving--;
            } else {
                MPI_Abort(MPI_COMM_WORLD, -1);
            }
            MPI_Irecv(&LocalRecvBuffer, localBufferSize,
                      MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
        }
    } while (have_done && NumberReceiving > 0);
}
```

```
if (pendingUpdates < maxPendingUpdates) {
    Ran = (Ran << 1) ^ ((s64)int) Ran < ZEROCARD ? ZEROCARD : 1;
    GlobalOffset = Ran & (tparams.TableSize - 1);
    if (GlobalOffset < tparams.Top)
        WhichPe = (GlobalOffset / (tparams.MiniBatchSize));
    else
        WhichPe = ((GlobalOffset - tparams.RunningOffset) / (tparams.MiniBatchSize));
}
```

```
if (WhichPe == tparams.MyProc) {
    LocalOffset = (Ran & (tparams.TableSize - 1)) - tparams.GlobalStartMyProc;
    HPCC_Table[LocalOffset] ^= Ran;
}
```

```
localSendBuffer, localBufferSize, &peUpdates);
dates, tparams.dtype64, (int)pe, UPDATE_TAG,
i);
```

```
/* 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; }
    /* card garbage... who cares, no one will look at it */
    proc_count, FINISHED_TAG,
    finish_req + proc_count);
}
```

```
pe = HPCC_GetUpdates(Buckets, LocalSendBuffer, localBufferSize, &peUpdates);
MPI_Isend(&LocalSendBuffer, peUpdates, tparams.dtype64, (int)pe, UPDATE_TAG,
          MPI_COMM_WORLD);
pendingUpdates -= pe;
```

```
1;
```

```
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;
}
```

```
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... */
}
```

```
Ran = (Ran << 1) ^ ((s64)int) Ran < ZEROCARD ? ZEROCARD : 1;
```

```
GlobalOffset = Ran & (tparams.TableSize - 1);
WhichPe = (GlobalOffset / (tparams.MiniBatchSize));
```

```
if (GlobalOffset < tparams.Top)
    WhichPe = (GlobalOffset / (tparams.MiniBatchSize));
else
    WhichPe = ((GlobalOffset - tparams.RunningOffset) / (tparams.MiniBatchSize));
```

```
HPCC_Table[LocalOffset] ^= Ran;
```

```
LocalOffset = (Ran & (tparams.TableSize - 1)) - tparams.GlobalStartMyProc;
```

```
HPCC_Table[LocalOffset] ^= Ran;
```

```
}
```

```
} while (have_done && NumberReceiving > 0);
```

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



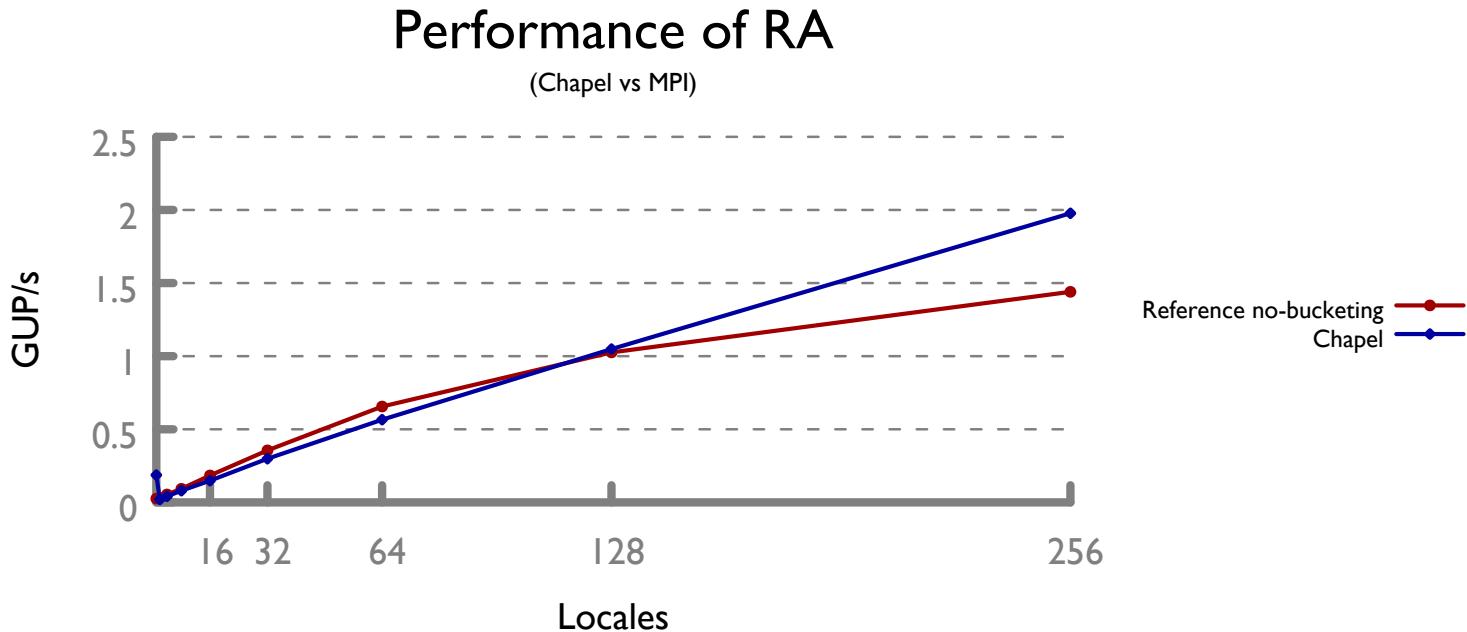
COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

HPCC RA Performance: Chapel vs. MPI



COMPUTE

STORE

ANALYZE

Chapel Runtime



COMPUTE

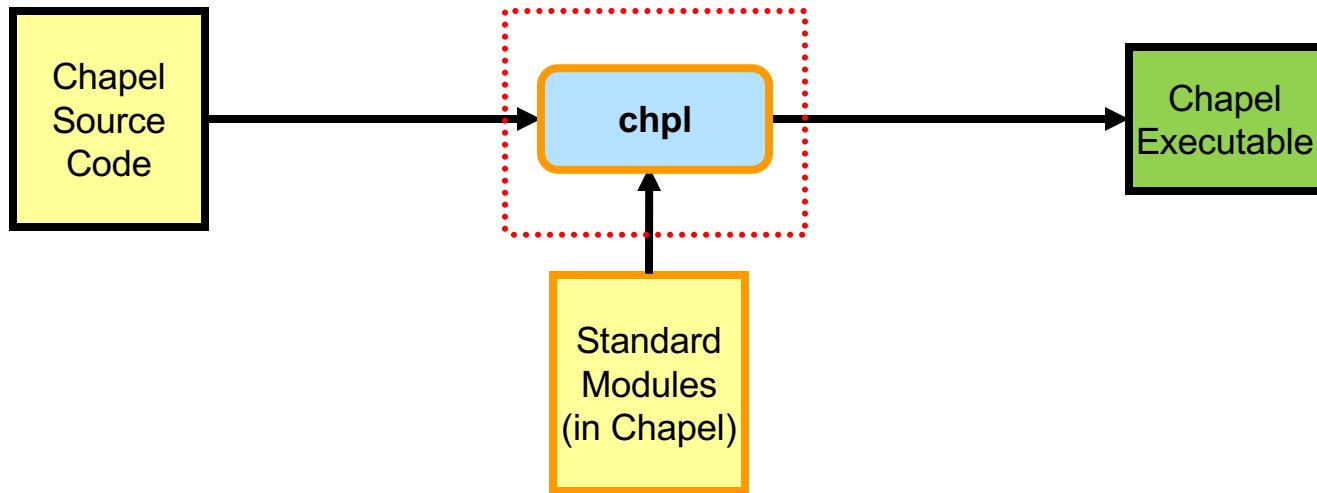


STORE



ANALYZE

Compiling Chapel

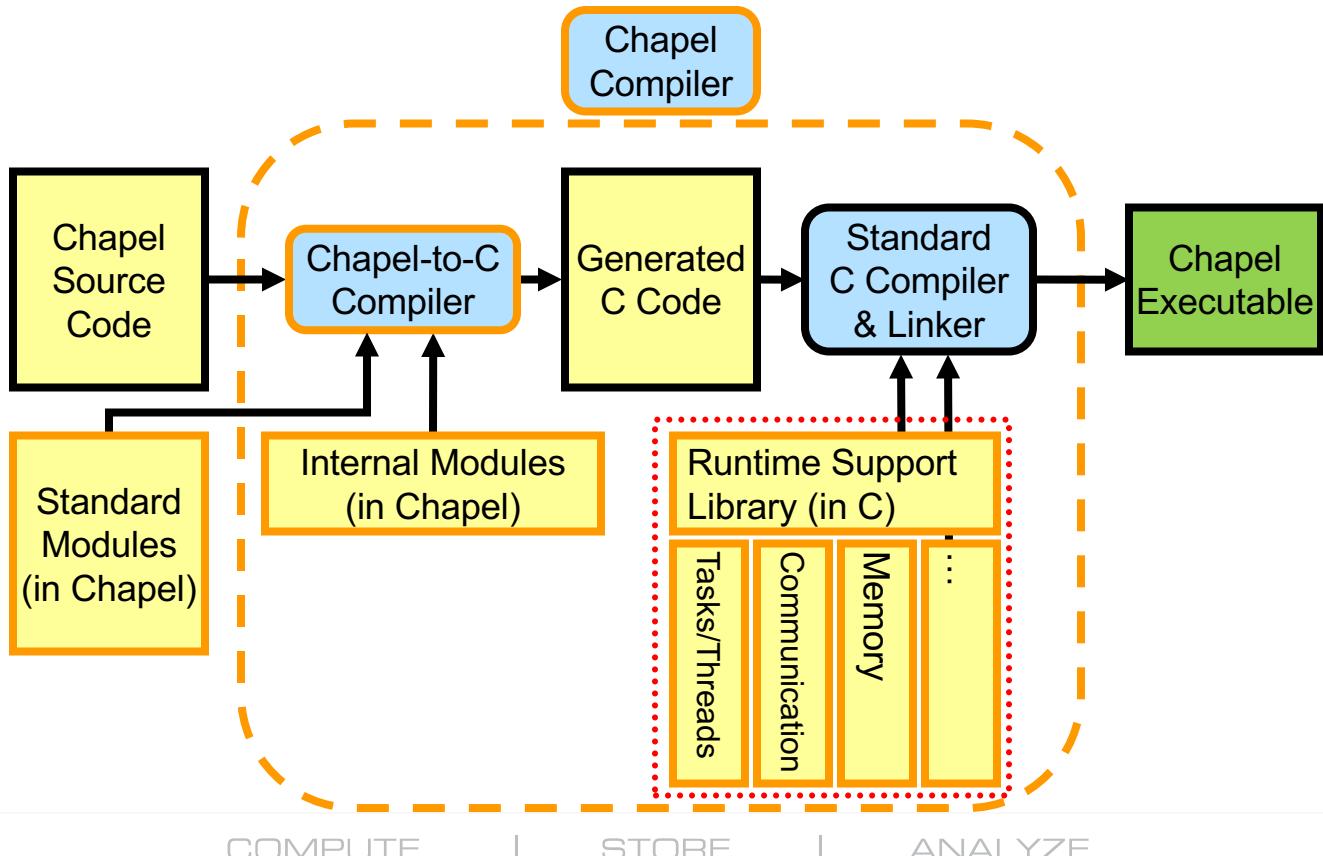


COMPUTE

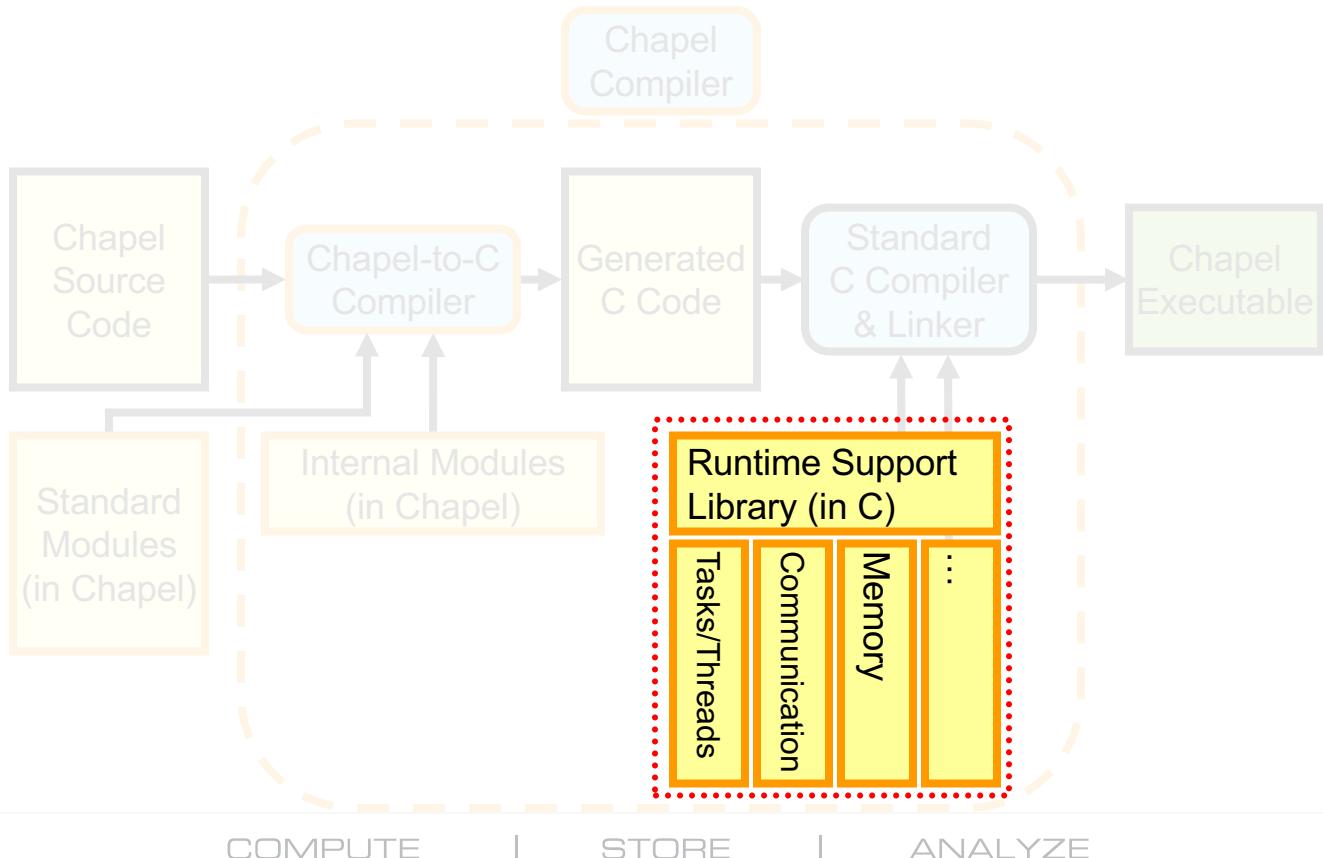
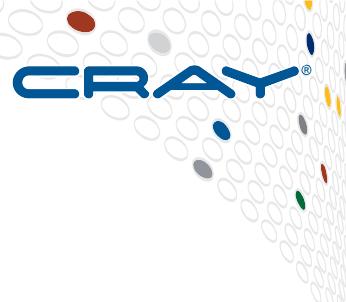
STORE

ANALYZE

Chapel Compilation Architecture



Chapel Compilation Architecture



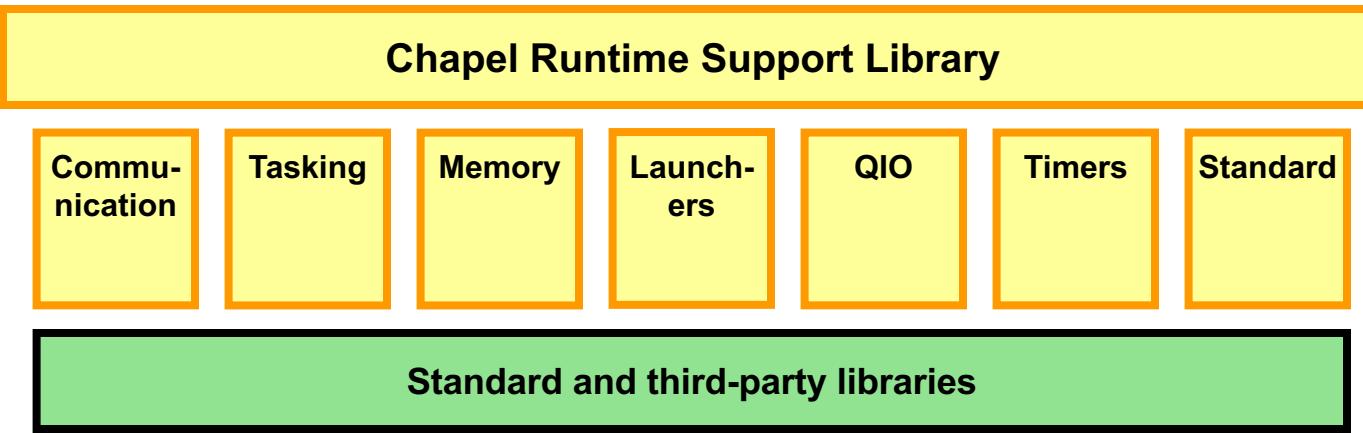
Chapel Runtime



- Lowest level of Chapel software stack
- Supports language concepts and program activities
 - Task creation, communication, memory allocation
- Composed of *layers*
 - Standardized interfaces
 - Interchangeable implementations



Chapel Runtime Organization

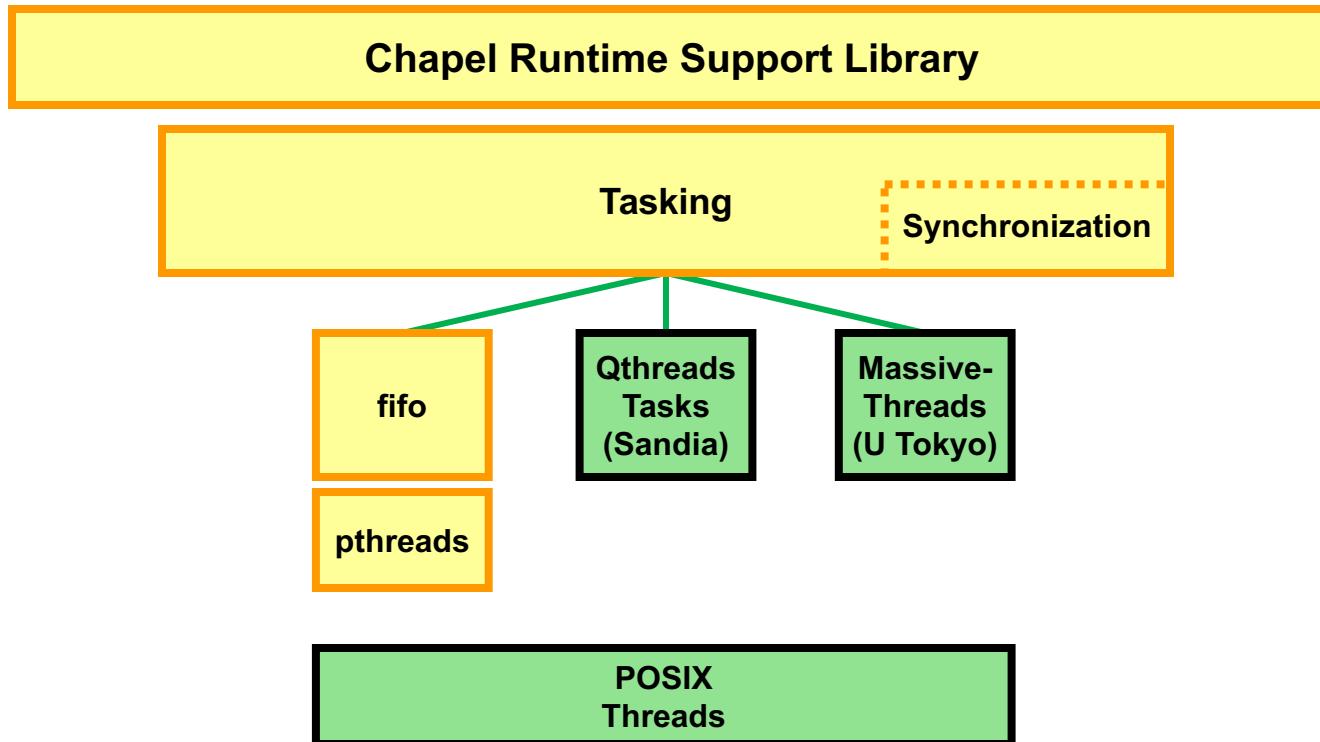
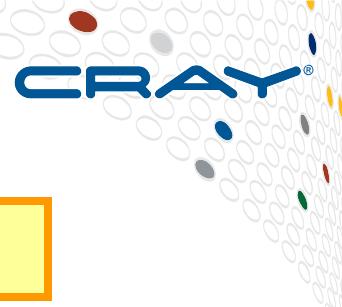


COMPUTE

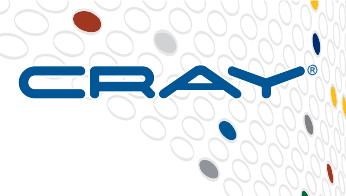
STORE

ANALYZE

Runtime Tasking Layer



Runtime Tasking Layer

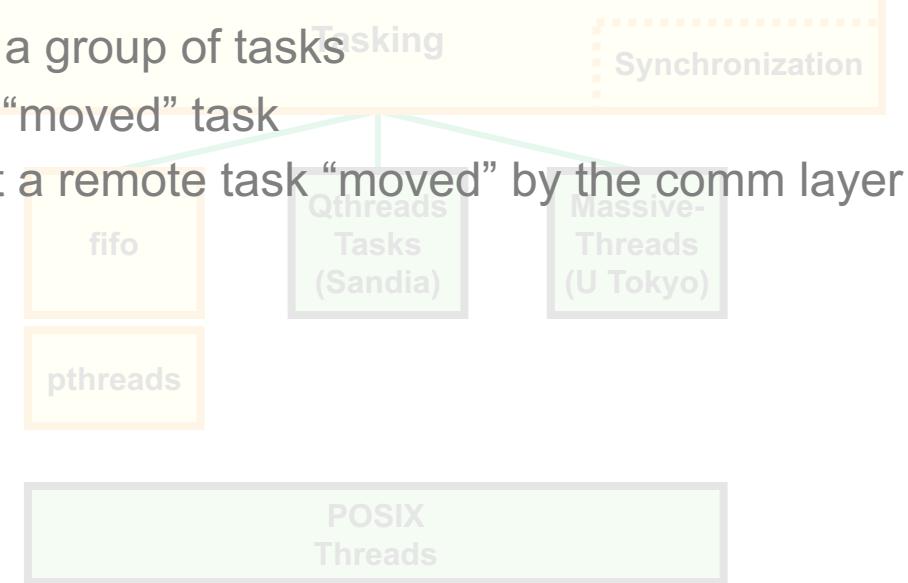


- Supports parallelism

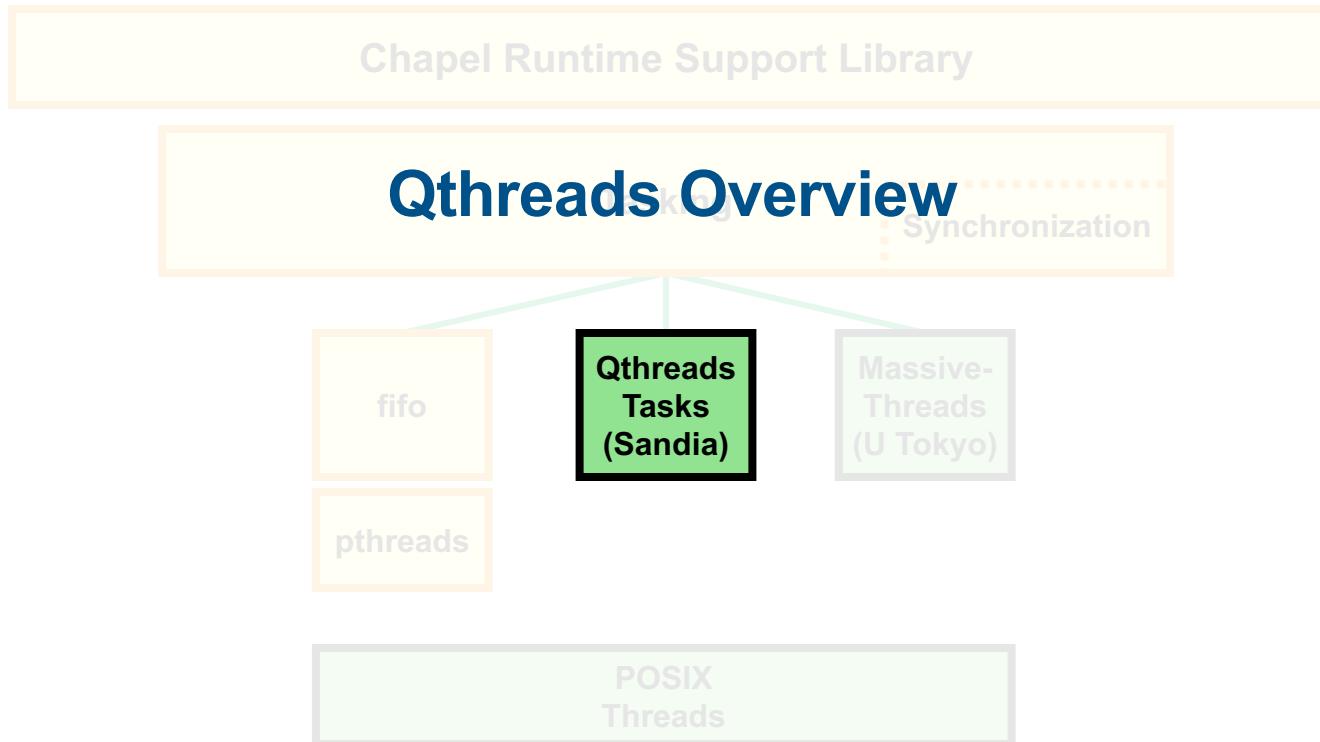
Support Library

- Operations

- Create a group of tasks
- Start a “moved” task
 - Start a remote task “moved” by the comm layer



Runtime Tasking Layer



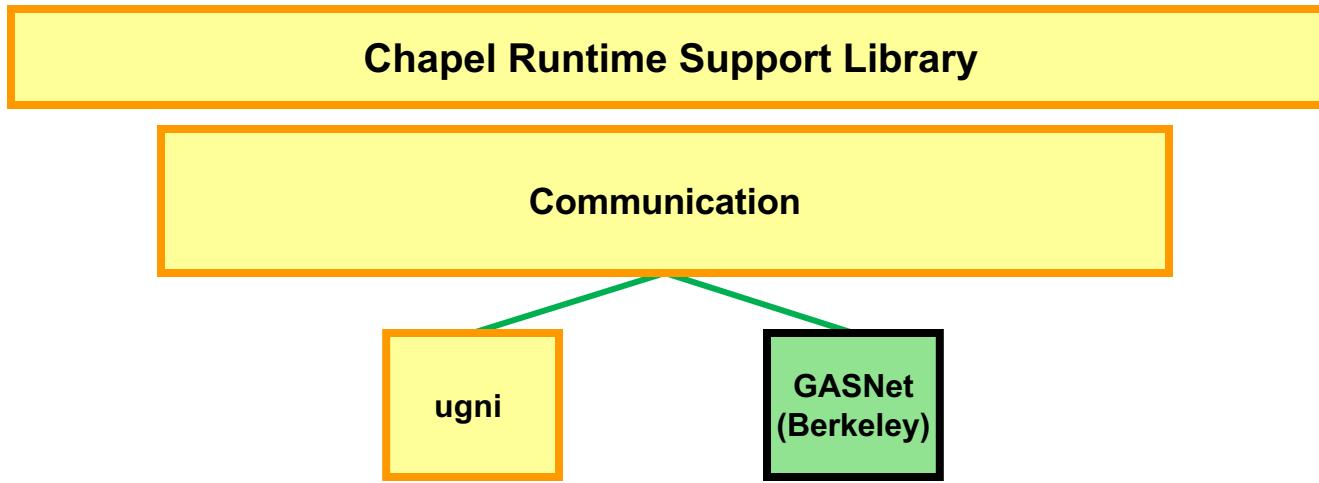
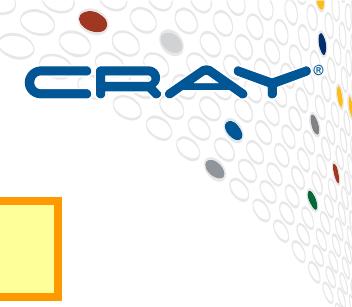
Qthreads Overview



- **Lightweight, locality-aware tasking library**
 - cooperative scheduling
 - qthreads are entirely in user space
 - extremely fast task creation and switching
 - designed to be highly concurrent
 - run millions of qthreads, limited only by available memory
 - locality-aware
 - multiple scheduler options
 - From simple fifo queues to advanced work-stealing schedulers



Runtime Communication Layer



Runtime Communication Layer



- Supports Communication
 - gets, puts, remote-task-creation
- Works with tasking layer (through API)
 - allows arbitrary comm/compute overlap

```
chpl_comm_put (...) {  
    done = do_remote_put (...) ;  
    while (!complete(&done)) {  
        chpl_task_yield(); // yield while waiting for network  
    }  
}
```



Communication + Tasking Overview

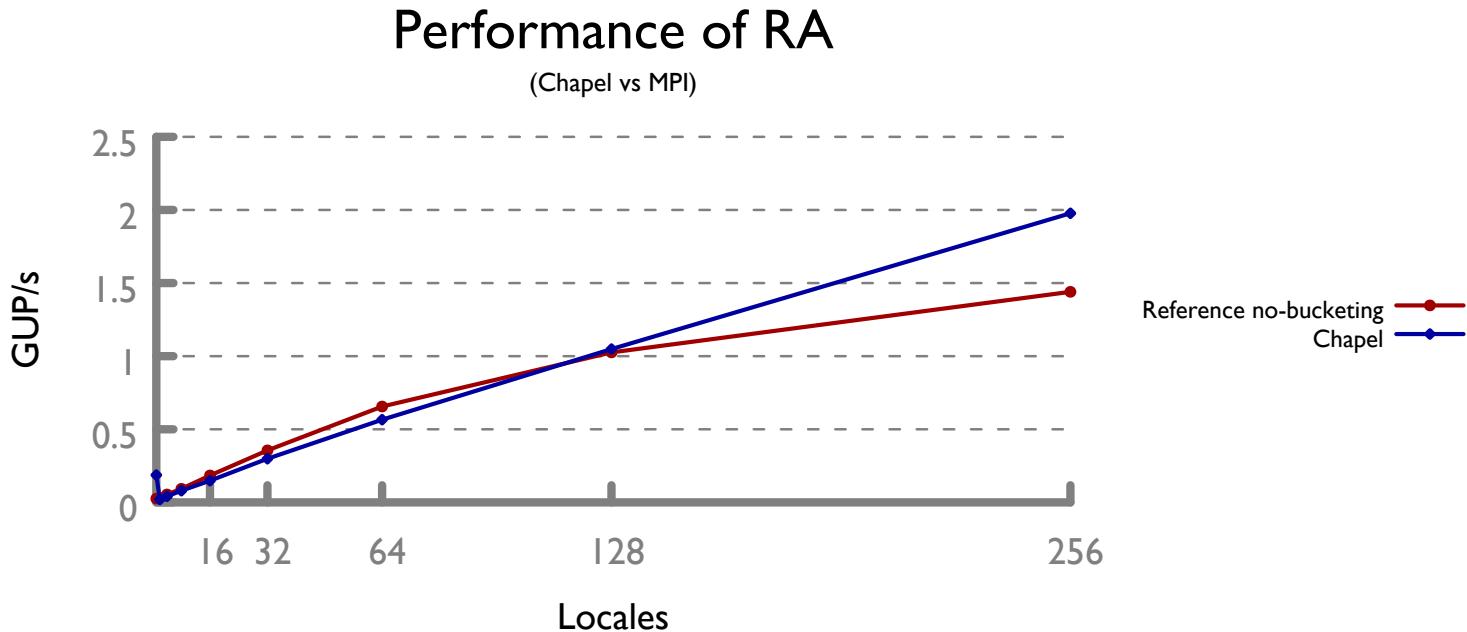


- “Unified” runtime permits many optimizations
 - standardized APIs prevent unnecessary/harmful coupling
- e.g. allows communication and computation overlap
 - in a trivial to implement manner
 - task switching in user-space makes this fast
 - cooperative tasking minimizes overhead for creating many tasks



Comm/compute overlap with RA

HPCC RA Performance: Chapel vs. MPI

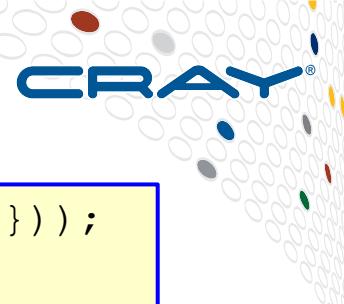


COMPUTE

STORE

ANALYZE

HPCC RA Performance: Chapel vs. MPI

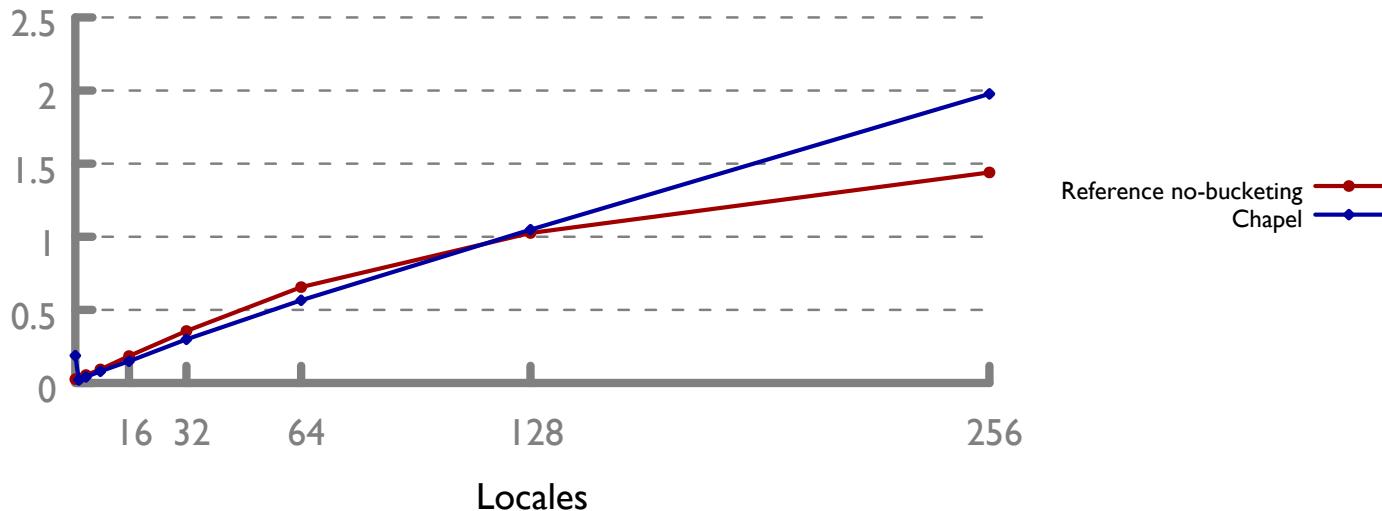
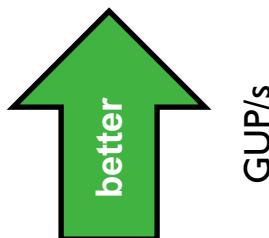


Update Dist Declaration

```
UpdateDist = new dmap (new Block (boundingBox={ 0 .. N_U-1 } ) );
```

Performance of RA

(Chapel vs MPI)

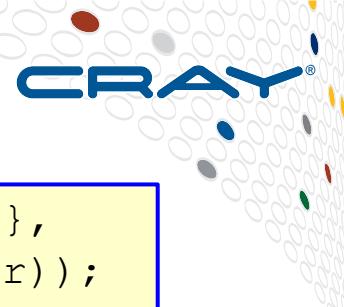


COMPUTE

STORE

ANALYZE

HPCC RA Performance: Chapel vs. MPI

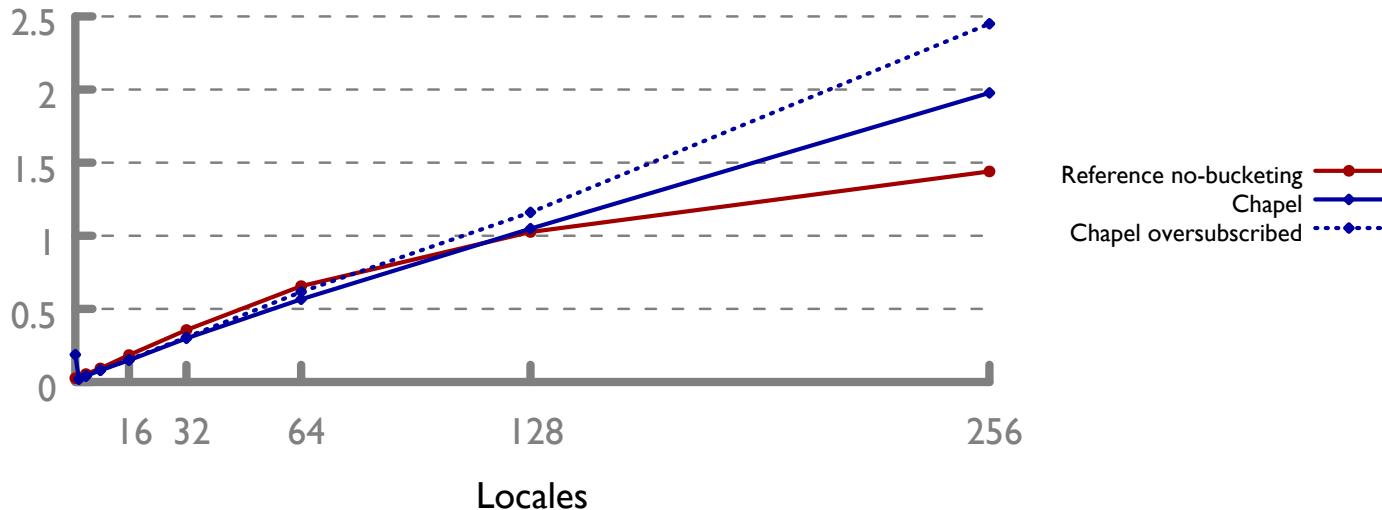
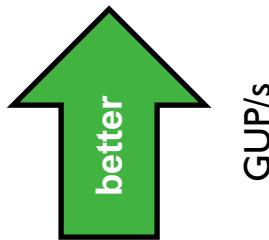


Update Dist Declaration

```
UpdateDist = new dmap (new Block (boundingBox={ 0 .. N_U-1 },  
tasksPerLocale=2*here.maxTaskPar) );
```

Performance of RA

(Chapel vs MPI)



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.



Wrapping Up



COMPUTE

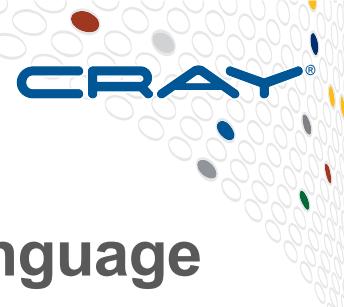


STORE



ANALYZE

Summary



- **Chapel is a productive parallel programming language**
 - productivity enabled by rich tasking and communication runtime
- **Flexible, but cohesive runtime enables optimizations**
 - e.g. comm/compute overlap
 - future avenues: distributed work stealing, comm aggregation
- **Chapel performance can match C+MPI+OpenMP**
 - with improvements in readability, writability, code size



COMPUTE

STORE

ANALYZE



Chapel Resources



COMPUTE



STORE



ANALYZE

Chapel Central: <https://chapel-lang.org/>





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 [book 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:

```
use CyclicDist;           // use the Cyclic distribution Library
config const n = 100;      // use ./a.out --n=<val> to override this default
forall i in {1..n} mapped Cyclic(startIdx=1) do
    writeln("Hello from iteration ", i, " of ", n, " running on node ", here.id);
```

What's Hot?

- Chapel 1.16 is now available—[download](#) a copy today!
- The [CHI UW 2018 call for participation](#) is now available!
- A recent [Cray blog post](#) reports on highlights from CHI UW 2017.
- Chapel is now one of the supported languages on [Try It Online!](#)
- Watch talks from [ACCU 2017](#), [CHI UW 2017](#), and [ATPESC 2016](#) on [YouTube](#).
- [Browse slides](#) from **PADAL**, **EAGE**, **EMBRACE**, **ACCU**, and other recent talks.
- See also: [What's New?](#)



COMPUTE

STORE

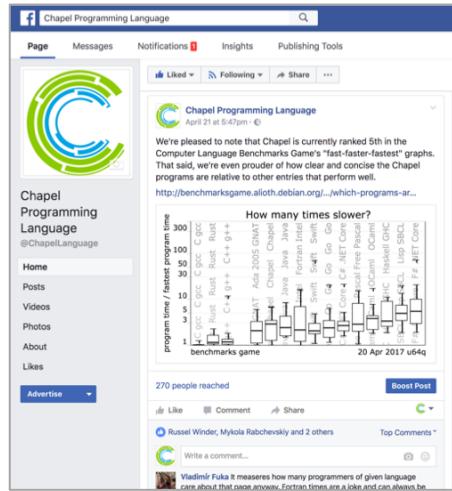
ANALYZE

How to Track Chapel

<http://facebook.com/ChapelLanguage>

<http://twitter.com/ChapelLanguage>

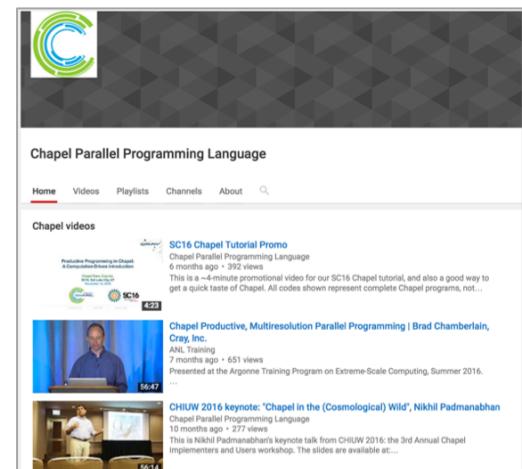
<https://www.youtube.com/channel/UCHmm27bYjhknK5mU7ZzPGsQ/>
chapel-announce@lists.sourceforge.net



The Facebook page for Chapel Programming Language has 129 followers. It features a post from April 21, 2017, at 8:47pm, which reads: "We're pleased to note that Chapel is currently ranked 5th in the Computer Language Benchmarks Game's "fast-faster-fastest" graphs. That said, we're even prouder of how clear and concise the Chapel programs are relative to other entries that perform well." Below the post is a chart titled "Program time / fastest program time" comparing various languages. The chart shows Chapel performing well, often faster than others like C and Fortran.



The Twitter profile for Chapel Language (@ChapelLanguage) has 222 tweets, 12 following, 129 followers, and 32 likes. A recent tweet from April 20, 2017, at 9:44pm encourages users to submit interesting applications to the PAW 2017 workshop at SC17. The profile also includes a link to their GitHub repository: [sourcey/institute.github.io/PAW/](https://github.com/sourcey/institute). The bio states: "Chapel is a productive parallel programming language designed for large-scale computing whose development is being led by @cray_inc".



The YouTube channel for Chapel Parallel Programming Language has 129 subscribers. It features several video uploads, including a promotional video for SC16 Chapel Tutorial, a tutorial on Chapel Productive, Multiresolution Parallel Programming by Brad Chamberlain, and a keynote by Nikhil Padmanabhan from CHI16. The channel also includes a playlist for the 2nd Annual PGAS Applications Workshop held in November 2017.



COMPUTE

STORE

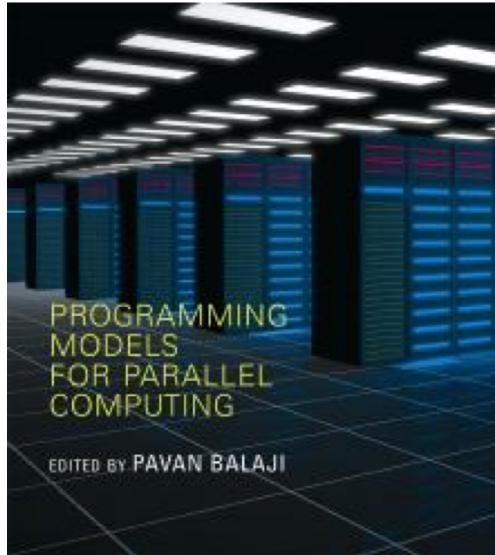
ANALYZE

Suggested Reading (healthy attention spans)



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 now also available [online](#)



Other Chapel papers/publications available at <https://chapel-lang.org/papers.html>



COMPUTE

STORE

ANALYZE

Suggested Reading (short attention spans)



[CHIUW 2017: Surveying the Chapel Landscape](#), Cray Blog, July 2017.

- *a run-down of recent events*

[Chapel: Productive Parallel Programming](#), Cray Blog, May 2013.

- *a short-and-sweet introduction to Chapel*

[Six Ways to Say “Hello” in Chapel](#) (parts [1](#), [2](#), [3](#)), Cray Blog, Sep-Oct 2015.

- *a series of articles illustrating the basics of parallelism and locality in Chapel*

[Why Chapel?](#) (parts [1](#), [2](#), [3](#)), Cray Blog, Jun-Oct 2014.

- *a series of articles answering common questions about why we are pursuing Chapel in spite of the inherent challenges*

[Ten] Myths About Scalable Programming Languages, [IEEE TCSC Blog](#)

(index available on chapel-lang.org “blog posts” page), Apr-Nov 2012.

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



Chapel StackOverflow and GitHub Issues



The screenshot shows two side-by-side web pages. On the left is the Stack Overflow 'chapel' tag page, displaying a list of questions related to the Chapel programming language. On the right is the GitHub repository 'chapel-lang/chapel' showing its list of open issues.

Stack Overflow Questions Page:

- Can one generate a grid of the Locales where a Distribution is mapped?**
If I run the following code: use BlockDist; config const dimension: int = 5; const space = (0..#0..#dimension); const matrixBlock: domain(2) dmapped Block(boundingBox=space) = space
2 votes, 2 answers, 22 views.
- Is '<var> in <distributed variable>' equivalent to 'forall'?**
I noticed something in a snippet of code I was given: var D: domain(2) dmapped Block(bound = Space; var A: [D] int; [a in A] a = a.locale.id; Is [a in A] equivalent to forall a in A a = ...
3 votes, 1 answer, 24 views.
- Get Non-primitive Variables from within a Cobegin - Chapel**
I want to compute some information in parallel and use the result outside the cobegin. To be my requirement is to retrieve a domain (and other non primitive types) like this var a,b; ...
2 votes, 1 answer, 45 views.
- Is there a default String conversion method in Chapel?**
Is there a default method that gets called when I try to cast an object into a string? (E.g. toStdString in Python.) I want to be able to do the following with an array of Objects; ...
3 votes, 1 answer.

Github Issues Page:

- chapel-lang / chapel**
Issues 292, Pull requests 26, Projects 0, Settings, Insights.
- Filters:** is:issue is:open, Labels, Milestones, New issue.
- Issues List:**
 - 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, #6356 opened 13 hours ago by ronawho
 - make uninstall** area: BTR, type: Feature Request, #6353 opened 14 hours ago by mpff
 - make check doesn't work with ./configure** area: BTR, #6352 opened 16 hours ago by mpff
 - Passing variable via in 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 cassella
 - Remove chpl_comm_make_progress** area: Runtime, easy, type: Design, #6349 opened a day ago by sungeunchoi
 - Runtime error after make on Linux Mint** area: BTR, user issue, #6348 opened a day ago by danindiana



COMPUTE

STORE

ANALYZE

Copyright 2018 Cray Inc.

Where to..



Submit bug reports:

[GitHub issues for chapel-lang/chapel](#): public bug forum
chapel_bugs@cray.com: for reporting non-public bugs

Ask User-Oriented Questions:

[StackOverflow](#): when appropriate / other users might care
[#chapel-users \(irc.freenode.net\)](#): user-oriented IRC channel
chapel-users@lists.sourceforge.net: user discussions

Discuss Chapel development

chapel-developers@lists.sourceforge.net: developer discussions
[#chapel-developers \(irc.freenode.net\)](#): developer-oriented IRC channel

Discuss Chapel's use in education

chapel-education@lists.sourceforge.net: educator discussions

Directly contact Chapel team at Cray: chapel_info@cray.com



Questions?



COMPUTE

|

STORE

|

ANALYZE

Legal Disclaimer



Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publically announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

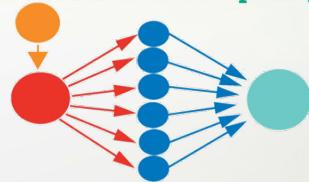
Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, and URIKA. The following are trademarks of Cray Inc.: ACE, APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, THREADSTORM. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT, and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used in this document are the property of their respective owners.





SIAM Conference on
Parallel Processing
for Scientific Computing



CRAY
THE SUPERCOMPUTER COMPANY