Chapel Unblocked:

Recent Communication Optimizations in Chapel

Elliot Ronaghan

CHIUW 2019

June 22, 2019

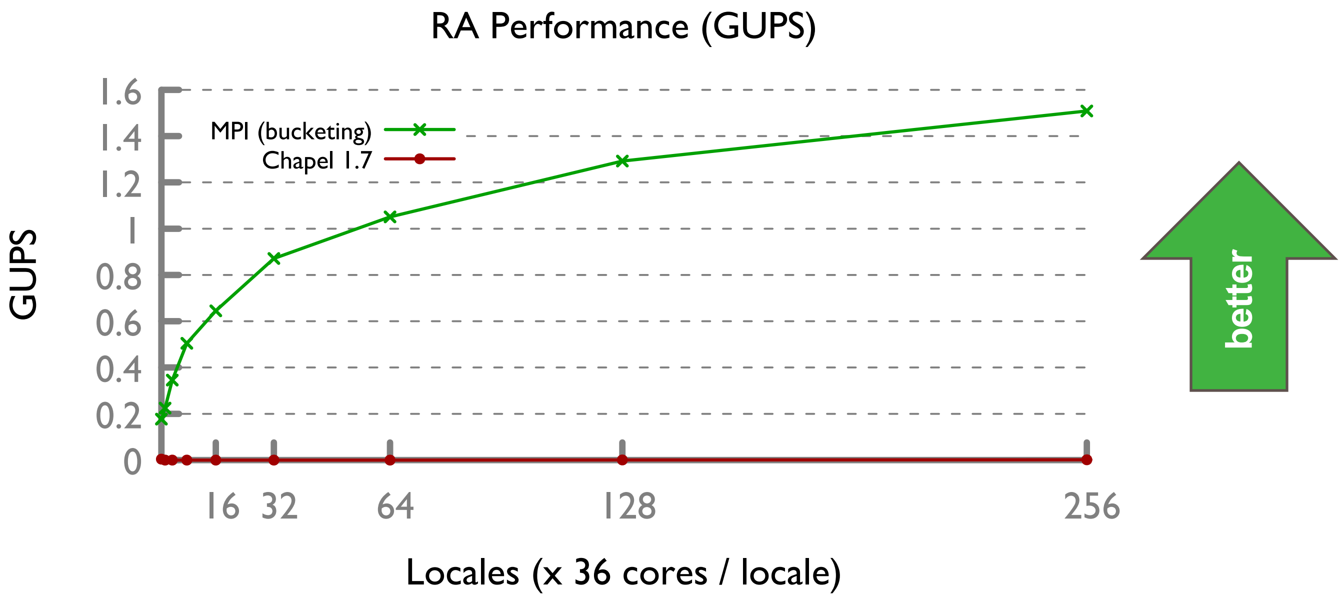
chapel_info@cray.com
chapel-lang.org
@ChapelLanguage

CRAY

CRAY CHAPEL

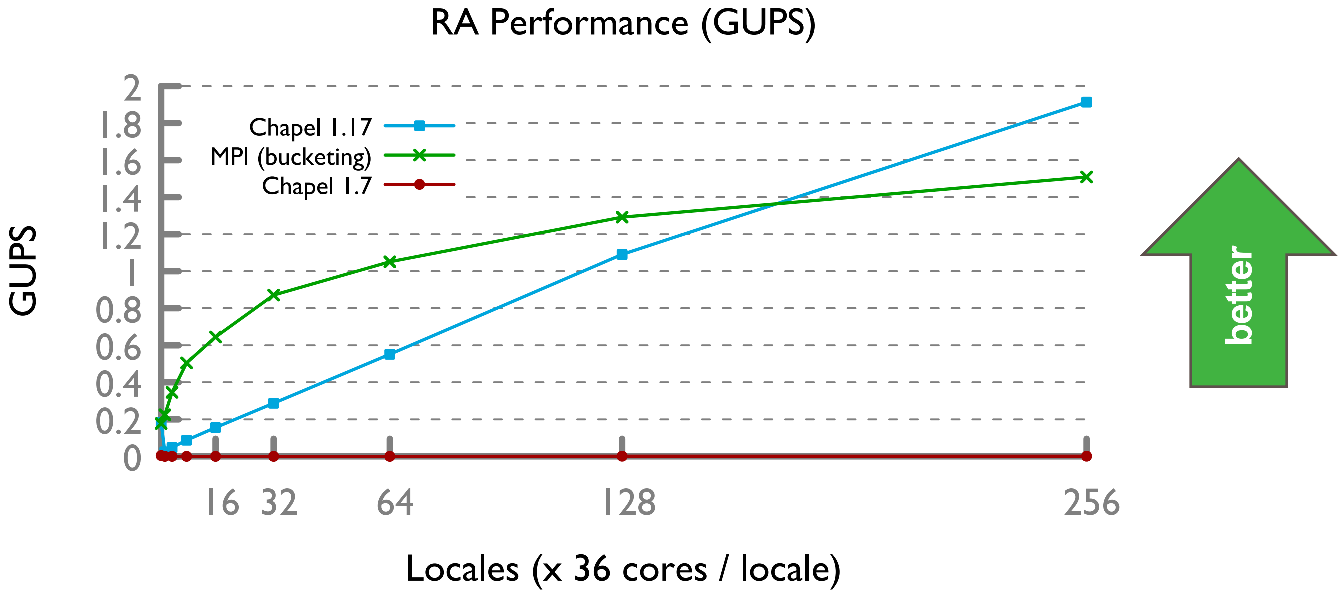
CHIUW 2018 Performance Summary

- In Chapel 1.7 performance was very far off from reference MPI/UPC/SHMEM



CHIUW 2018 Performance Summary

- With 1.17 many applications could achieve performance parity
  - However, still possible to fall off a performance cliff for other applications



Plan for this Talk

- We have implemented dozens of significant optimizations since last year
  - Our performance optimizations are largely benchmark driven

- This talk will focus on 3 key benchmarks and communication optimizations
  - ISx -- Bulk communication optimizations
  - Stream -- Remote task spawning optimizations
  - Random Access -- Fine-grained communication optimizations

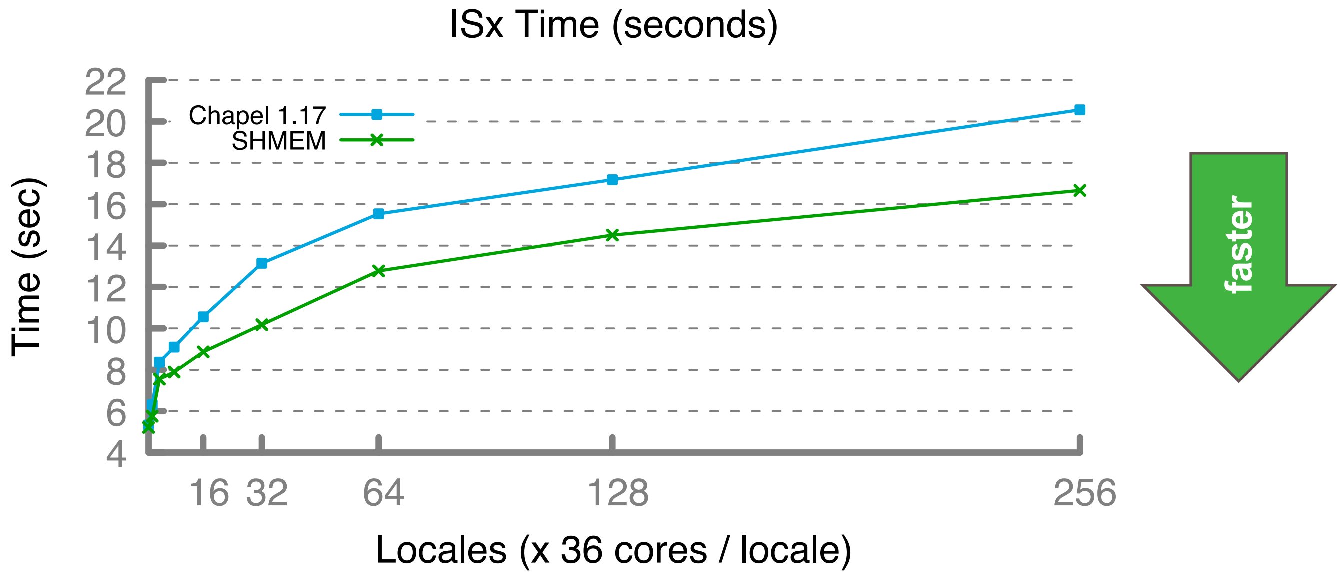
ISx Optimization

ISx: Background

- Scalable Integer Sort benchmark
  - Developed at Intel, published at PGAS 2015
  - SPMD-style computation with barriers
  - Punctuated by all-to-all bucket-exchange pattern
    - buckets being exchanged are relatively large (100's of MBs)
  - References implemented in SHMEM and MPI

ISx: Background

- Chapel 1.17 scaled well, but raw performance was up to 30% behind SHMEM



ISx: Large Message Optimization

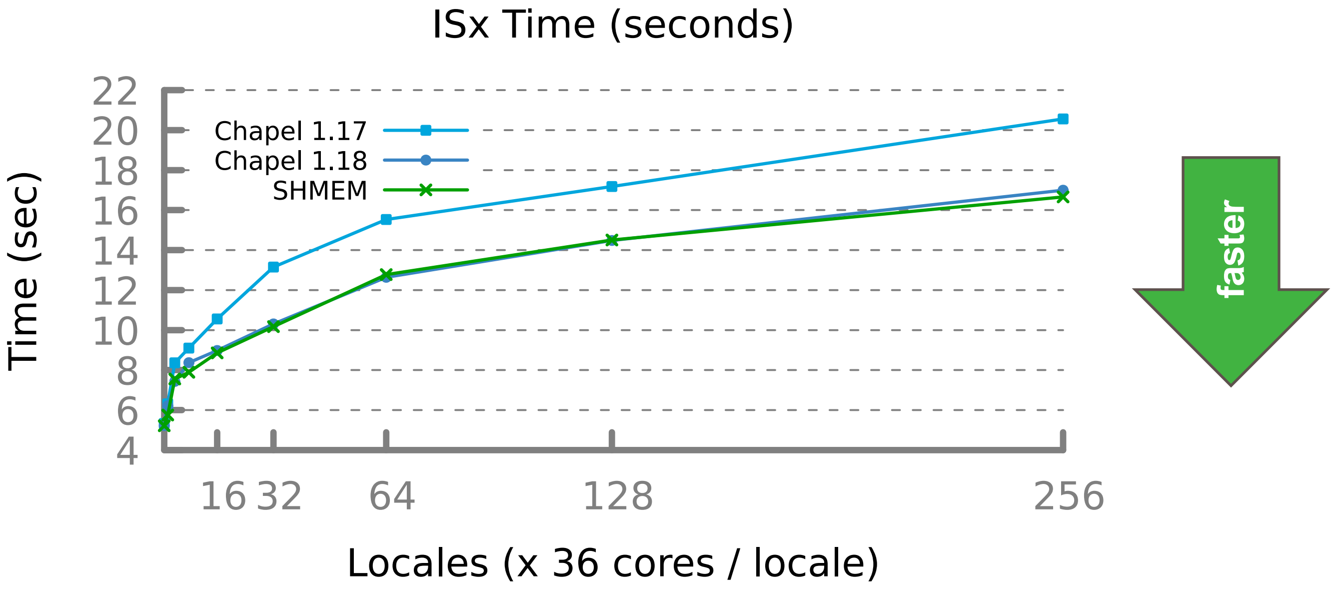
- On Crays Chapel uses the uGNI library to implement communication
  - uGNI provides 2 remote memory access mechanisms
    - Fast Memory Access (FMA)
    - Block Transfer Engine (BTE)

- Prior to 1.18, all communication was initiated with FMA
  - Discovered that BTE offers significantly better performance for large transfers

- In 1.18 we switched to initiating large transfers (4K or larger) with BTE
  - Significantly increased sustained bandwidth, can fully saturate network now

ISx: Performance Impact

- Chapel 1.18 performs on par with reference SHMEM version



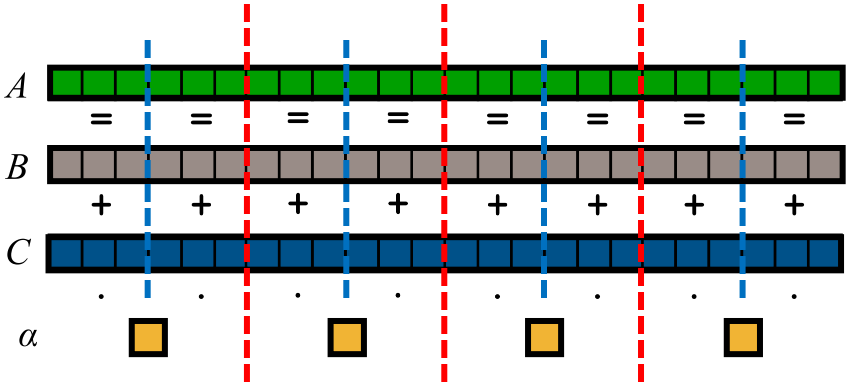
Stream Optimization

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: Background

- Multiple variants of Stream benchmark exist, e.g.:
  - **EP:** Explicit SPMD, uses local arrays, task spawning not included in time
  - **Global:** Elegant, uses block distributed arrays, task spawning included in time

Stream EP

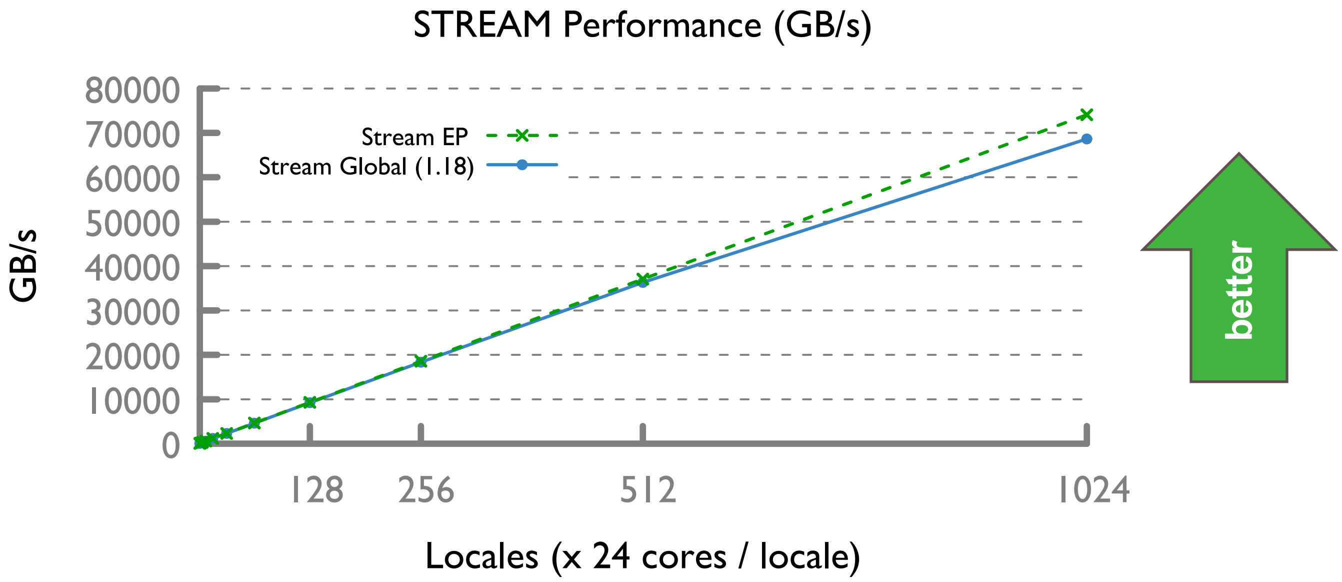
```
coforall loc in Locales do on loc {
  var A, B, C: [1..m] real;
  initVectors(B, C);

  startTimer();

  forall (a, b, c) in zip(A, B, C) do
    a = b + alpha * c;

  stopTimer();
}
```

Global Stream

```
const Space = {1..m} dmapped Block({1..m});
var A, B, C: [Space] real;
initVectors(B, C);

startTimer();

forall (a, b, c) in zip(A, B, C) do
  a = b + alpha * c;

stopTimer();
```

Stream: Background

- In 1.18, Global Stream performance lagged at higher locale counts



Stream: Task Spawning Optimization

- Task creation and on-statements are used to create remote tasks
- A common idiom is to create a task on each locale

```
coforall loc in Locales do on loc { body(args); }
```

Stream: Task Spawning Optimization

- Under 'ugni' in 1.18, remote-coforalls were translated into something like:

```
var endCount: atomic int = Locales.size;
for loc in Locales {
  var ACK = startRemoteTask(loc, bodyWrap, args, endCount);
  while (!received(ACK)) {}



}
endCount.waitFor(0);

proc bodyWrap(args, endCount) { body(args); endCount.sub(1); }
```

Stream: Task Spawning Optimization

- Under 'ugni' in 1.18, remote-coforalls were translated into something like:

```
var endCount: atomic int = Locales.size;
for loc in Locales {
  var ACK = startRemoteTask(loc, bodyWrap, args, endCount);
  while (!received(ACK)) {} // problem, network round trip wait



}
endCount.waitFor(0);


proc bodyWrap(args, endCount) { body(args); endCount.sub(1); }
```

Stream: Task Spawning Optimization

- They are now translated into something like:

```
var endCount: atomic int = Locales.size;
for loc in Locales {
  var ACK = startRemoteTask(loc, bodyWrap, args, endCount);
  ackBuff[ackIndex()] = ACK;
  if ackBuff.full() then          // normally not full, so no waiting
    retireAtLeastOneTX();         // fast, usually a few ready to retire
}
endCount.waitFor(0);

proc bodyWrap(args, endCount) { body(args); endCount.sub(1); }
```

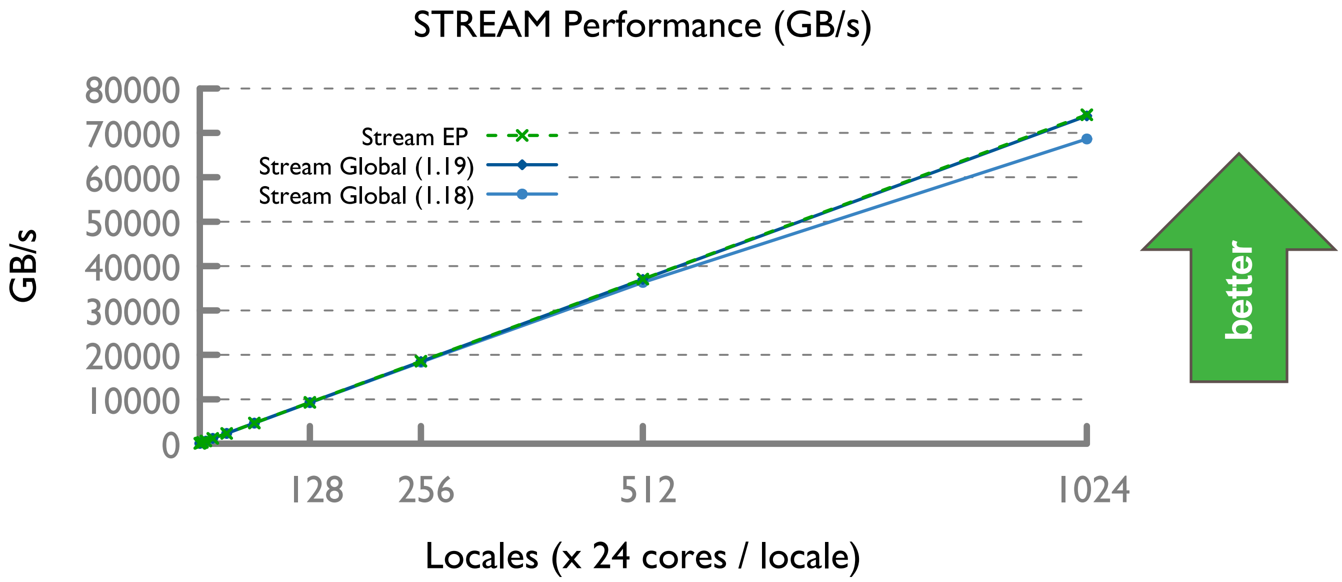
Stream: Task Spawning Optimization

- Other optimizations reduced the amount of communication required
  - Most remote tasks can be initiated with a single non-blocking transaction

- Combined, these optimizations resulted in 9x faster task creation at 1,024 locales

Stream: Performance Impact

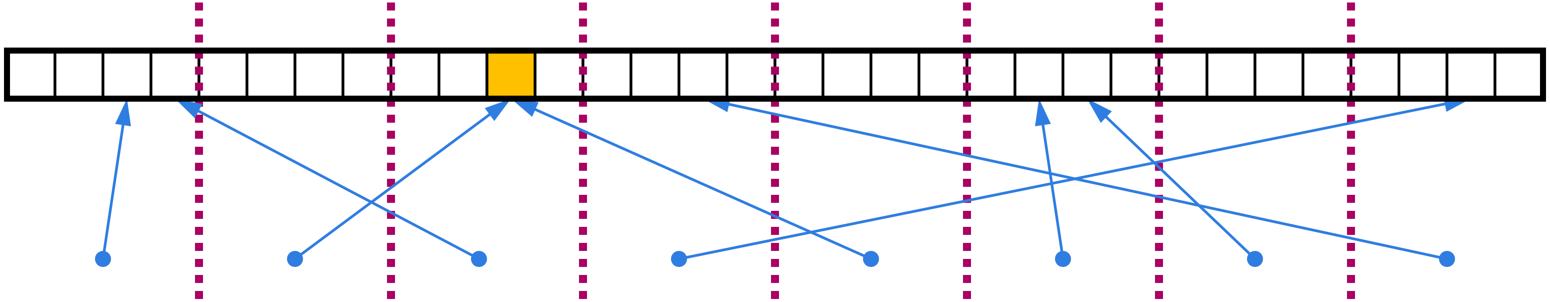
- Stream Global performance now on par with EP at 1,024 locales



Random Access Improvements

HPCC Random Access (RA)

Data Structure: distributed table



Computation: update random table locations in parallel

HPCC RA: MPI kernel

```
/* Perform updates to main table.  The scalar equivalent is:
 *
 *   for (i=0; i<Updates; i++) {
 *     r = (r << 1) ^ ((r < 0) ? POLY : 0);
 *     T[r & indexMask] ^= r;
 *   }
 */

MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
          MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
 while (i < SendCnt) {
   /* receive messages */
   do {
     MPI_Test(&inreq, &have_done, &status);
     if (have_done) {
       if (status.MPI_TAG == UPDATE_TAG) {
         MPI_Get_count(&status, tparams.dtype64, &recvUpdates);
         bufferBase = 0;
         for (j=0; j < recvUpdates; j ++) {
           inmsg = LocalRecvBuffer[bufferBase+j];
           LocalOffset = (inmsg & (tparams.TableSize - 1)) -
                         tparams.GlobalStartMyProc;
           HPCC_Table[LocalOffset] ^= inmsg;
         }
       } else if (status.MPI_TAG == FINISHED_TAG) {
         NumberReceiving--;
       } else
         MPI_Abort( MPI_COMM_WORLD, -1 );
       MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
                 MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
     }
   } while (have_done && NumberReceiving > 0);
   if (pendingUpdates < maxPendingUpdates) {
     Ran = (Ran << 1) ^ ((s64Int) Ran < ZERO64B ? POLY : ZERO64B);
     GlobalOffset = Ran & (tparams.TableSize-1);
     if ( GlobalOffset < tparams.Top)
       WhichPe = ( GlobalOffset / (tparams.MinLocalTableSize + 1) );
     else
       WhichPe = ( (GlobalOffset - tparams.Remainder) /
                   tparams.MinLocalTableSize );
     if (WhichPe == tparams.MyProc) {
       LocalOffset = (Ran & (tparams.TableSize - 1)) -
                     tparams.GlobalStartMyProc;
       HPCC_Table[LocalOffset] ^= Ran;
     }  else {
       HPCC_InsertUpdate(Ran, WhichPe, Buckets);
        pendingUpdates++;
     }
     i++;
   }
   else {
     MPI_Test(&outreq, &have_done, MPI_STATUS_IGNORE);
     if (have_done) {
       outreq = MPI_REQUEST_NULL;
       pe = HPCC_GetUpdates(Buckets, LocalSendBuffer, localBufferSize,
                            &peUpdates);
       MPI_Isend(&LocalSendBuffer, peUpdates, tparams.dtype64, (int)pe,
                 UPDATE_TAG, MPI_COMM_WORLD, &outreq);
       pendingUpdates -= peUpdates;
     }
   }
 }
/* send remaining updates in buckets */
while (pendingUpdates > 0) {
  /* receive messages */
  do {
    MPI_Test(&inreq, &have_done, &status);
    if (have_done) {
      if (status.MPI_TAG == UPDATE_TAG) {
        MPI_Get_count(&status, tparams.dtype64, &recvUpdates);
        bufferBase = 0;
        for (j=0; j < recvUpdates; j ++) {
          inmsg = LocalRecvBuffer[bufferBase+j];
          LocalOffset = (inmsg & (tparams.TableSize - 1)) -
                        tparams.GlobalStartMyProc;
          HPCC_Table[LocalOffset] ^= inmsg;
        }
      } else if (status.MPI_TAG == FINISHED_TAG) {
        /* we got a done message.  Thanks for playing... */
        NumberReceiving--;
      } else {
        MPI_Abort( MPI_COMM_WORLD, -1 );
      }
      MPI_Irecv(&LocalRecvBuffer, localBufferSize, tparams.dtype64,
                MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &inreq);
    }
  } while (have_done && NumberReceiving > 0);

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

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

HPCC RA: MPI kernel comment vs. Chapel


```
/* Perform updates to main table.  The scalar equivalent is:
 *
 *     for (i=0; i<Updates; i++) {
 *       r = (r << 1) ^ ((r < 0) ? POLY : 0);
 *       T[r & indexMask] ^= r;
 *     }
 */
```


Chapel Kernel

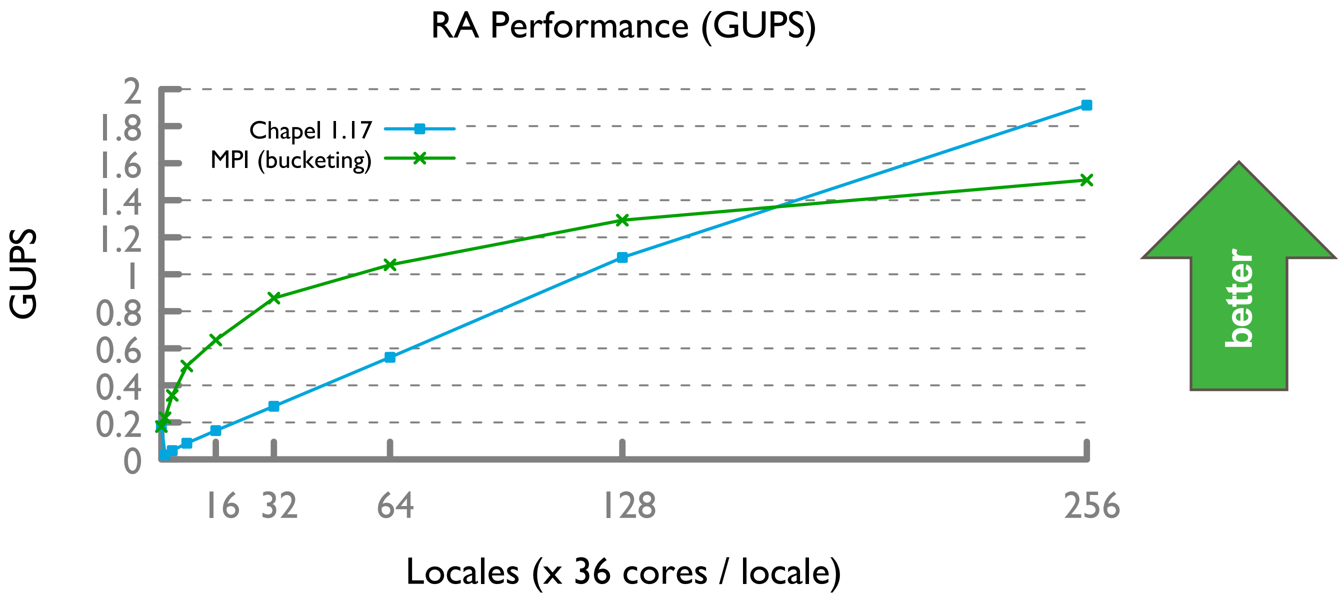
```
forall (_, r) in zip(Updates, RAStream()) do
  T[r & indexMask].xor(r);
```

MPI Comment

```
/* Perform updates to main table. The scalar equivalent is:
 *
 *     for (i=0; i<Updates; i++) {
 *       r = (r << 1) ^ ((r < 0) ? POLY : 0);
 *       T[r & indexMask] ^= r;
 *     }
 */
```

RA Performance

- In 1.17 Chapel already outperformed reference MPI
  - We have made significant improvements since then



Blocking Communication

- By default, remote operations in Chapel are blocking/ordered
  - Supports Memory Consistency Model (MCM)
    - “sequential consistency for data-race-free programs”

```
var a: atomic int;
a.add(1);
writeln(a); // must print 1
```

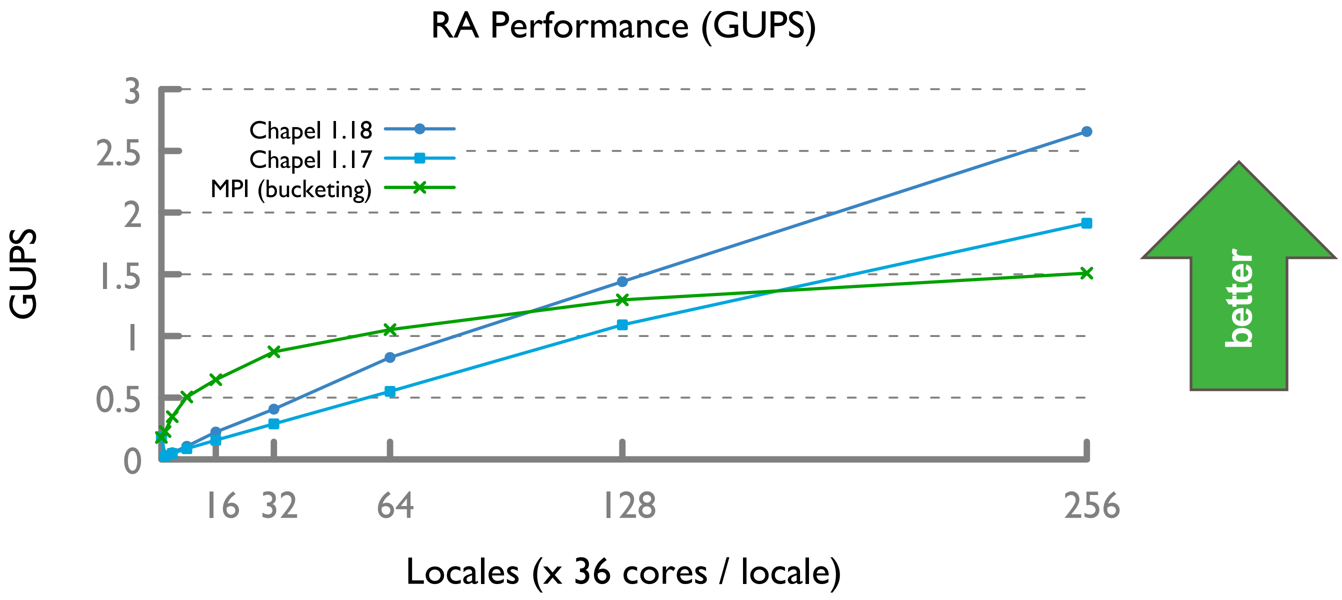
Blocking Communication

- Blocking is implemented with initiation, then task yields until ACK is received
  - Yielding allows for comm/compute overlap

```
var ACK = initiateAtomic(locale, …);
while (!received(ACK)) {
  chpl_task_yield();
}
```

Blocking Operations

- In 1.18 we optimized how we wait for blocking operations to complete
  - Yield less frequently to allow for faster ACK processing



Unordered Operations

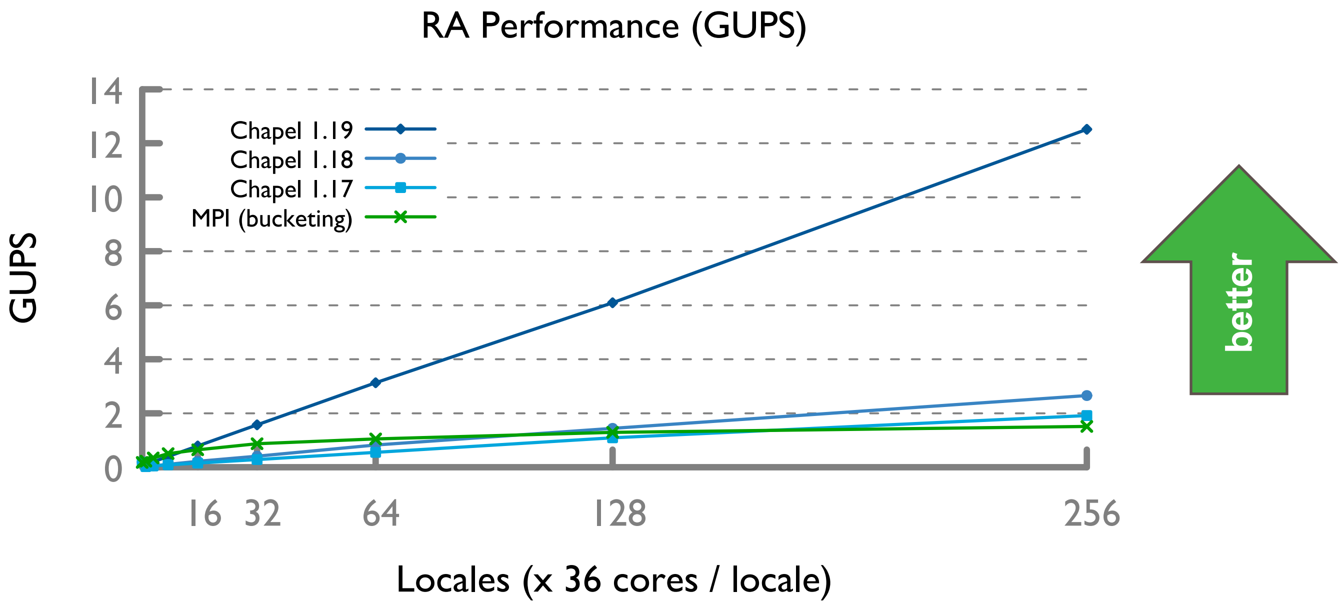
- 1.19 introduced unordered operations (including unordered atomics)
  - Unordered operations are not consistent with normal operations
  - Results are only visible at task/forall termination or with an explicit fence

```
var a: atomic int;
a.unorderedAdd(1);
writeln(a); // can print 0 or 1
unorderedAtomicTaskFence();
writeln(a); // must print 1
```

  - Allows for significant optimization leeway

Unordered Operations

- Unordered operations have significant performance advantages
  - 4.5x speedup over already optimized blocking/ordered performance



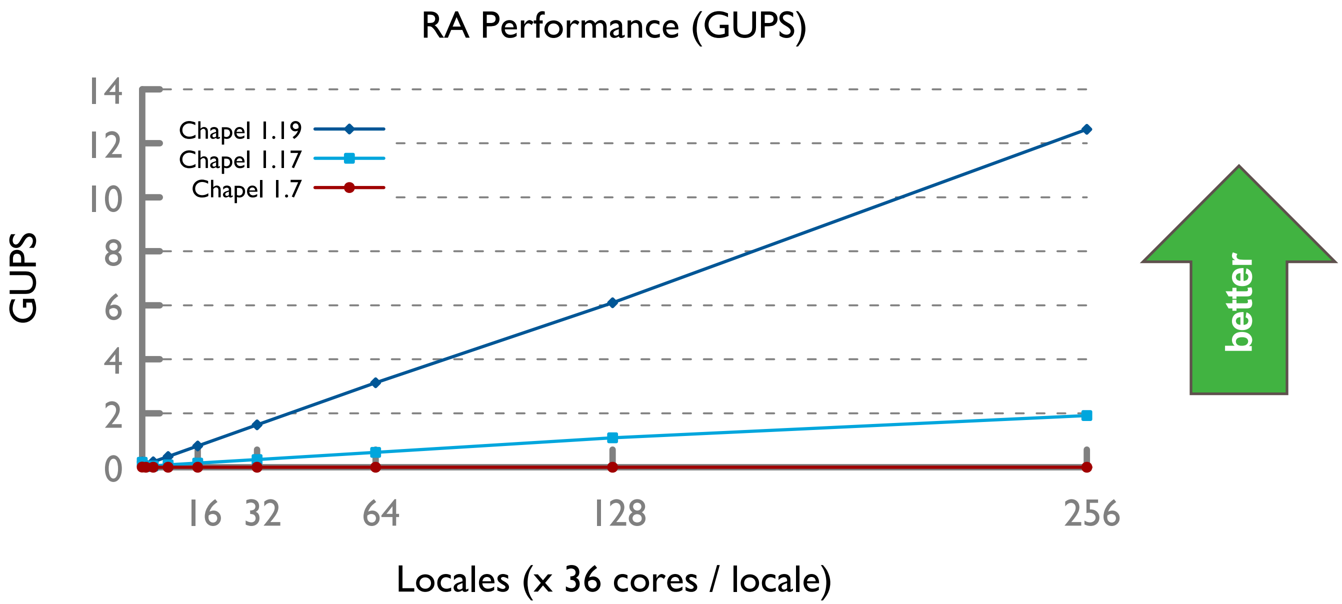
Unordered Compiler Optimization

- Since 1.19 we have enabled an unordered compiler optimization
  - Automatically transforms ordered communication into unordered when legal
  - Compiler is able to automatically optimize when …
    - Inside a forall loop (no ordering requirements across iterations)
    - Lifetime of operands is longer than forall loop scope
    - Operations are not used for synchronization
    - Result of operation is not used within the same iteration

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

RA Summary

- RA performance has improved significantly with no changes to the benchmark
  - Now achieves network injection rate for small messages



Performance Summary

- These communication optimizations have had significant performance impacts
  - 30% improvement for ISx at 256 locales (~10K cores)
  - 10% improvement for Stream Global at 1,024 locales (~25K cores)
  - 6x improvement for Random Access at 256 locales (~10K cores)

Performance Summary

- There have been dozens of other performance optimizations over the last year
  - Optimized Sync variables
  - Reduced Communication
  - Optimized Distributed Array Iteration
  - Optimized Sorting
  - Optimized Large Transfers
  - Optimized Network Atomics
  - Improved on-stmt Performance
  - Optimized Barriers
  - Improved Task Placement/Affinity
  - Optimized Linear Algebra Routines
  - Optimized Scan Performance
  - Improved String Performance
  - Optimized Locks
  - Defaulted to cstdlib Atomics
  - Improved Vectorization
  - Optimized Fine-Grained Comm
  - Added Unordered Operations
  - Improved Comm/Compute Overlap

Next Steps

- Continue benchmark driven optimizations
  - User Applications
  - Bale
  - DOE Proxy Apps
  - Intel Parallel Research Kernels
- Optimize for non-Cray networks
  - In particular optimize for InfiniBand

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?

chapel_info@cray.com
@ChapelLanguage
chapel-lang.org

cray.com
@cray_inc
linkedin.com/company/cray-inc-