HPCC STREAM and RA in Chapel Performance and Potential

Steve Deitz

Cray Inc.

What is Chapel?

- A new parallel language
  - Under development at Cray Inc.
  - Supported through the DARPA HPCS program
- Goals
  - **Improve programmer productivity**
  - Improve the programmability of parallel computers
  - Match or improve performance of MPI/UPC/CAF
  - Provide better portability than MPI/UPC/CAF
  - Improve robustness of parallel codes
  - Support multi-core and multi-node systems

Outline

- What is Chapel?
- Chapel's Parallel Programming Model
- HPCC STREAM Triad in Chapel
- HPCC RA in Chapel
- Summary and Future Work

Fragmented vs. Global-View: Definitions

- Programming model

  The mental model of a programmer

- Fragmented models

  Programmers take point-of-view of a single processor/thread

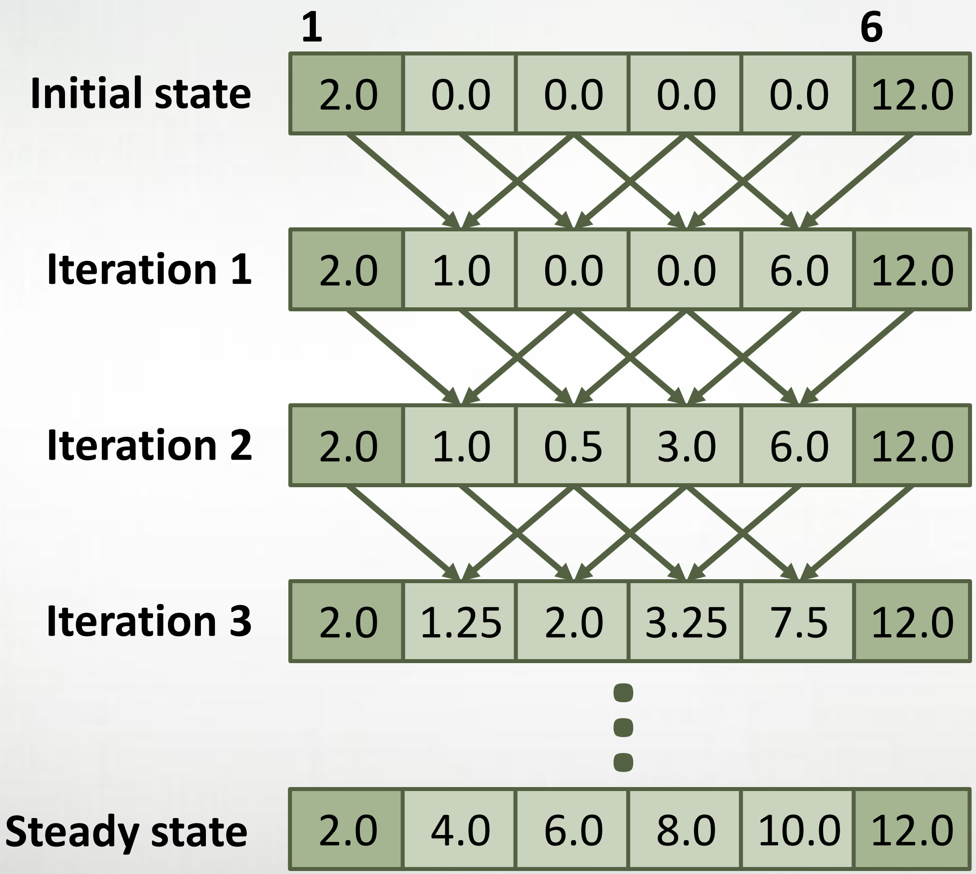
- SPMD models (Single Program, Multiple Data)

  Fragmented models with multiple copies of one program

- Global-view models

  Programmers write code to describe computation as a whole

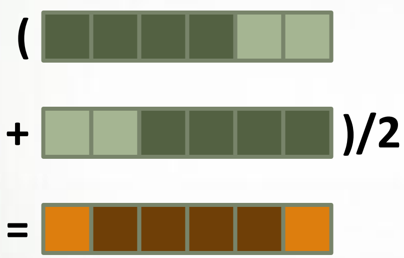
3-Point Stencil Example (n=6)



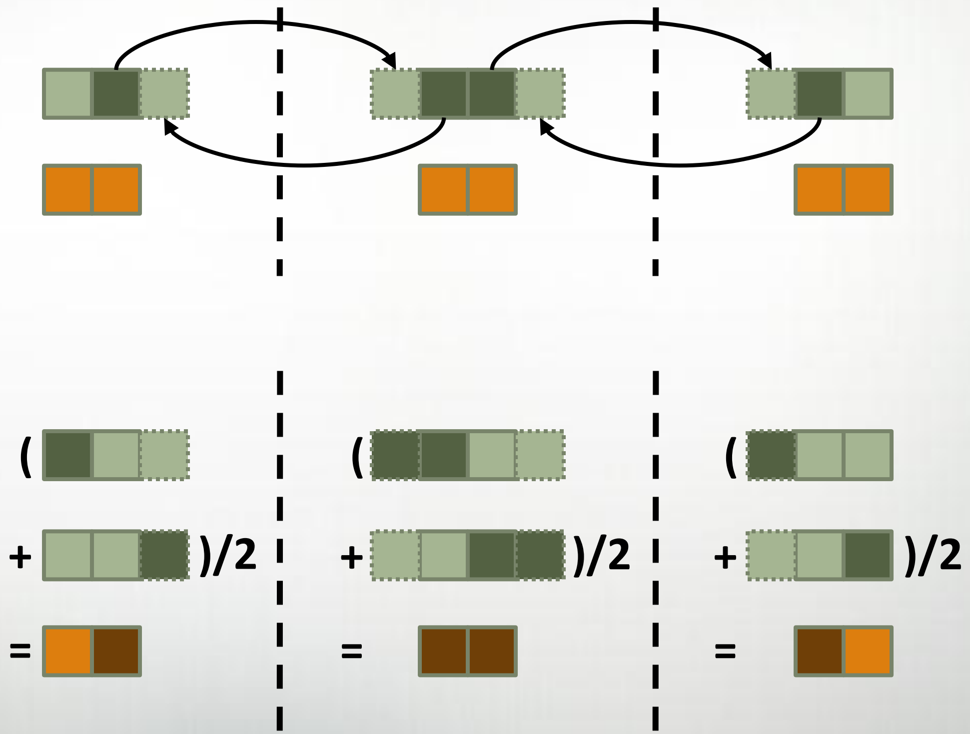
3-Point Stencil Example

Global-View vs. Fragmented Computation

Global-View



Fragmented

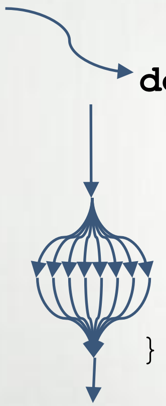


3-Point Stencil Example: Code

Global-View vs. Fragmented Code

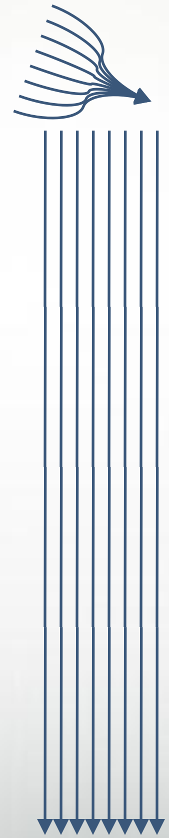
Assumes p divides n

Global-View



```
def main() {
  var n = 1000;
  var A, B: [1..n] real;

  forall i in 2..n-1 do
    B(i) = (A(i-1)+A(i+1))/2;
}
```

Fragmented



```
def main() {
  var n = 1000;
  var me = commID(), p = commProcs(),
    myN = n/p, myLo = 1, myHi = myN;
  var A, B: [0..myN+1] real;

  if me < p {
    send(me+1, A(myN));
    recv(me+1, A(myN+1));
  } else myHi = myN-1;
  if me > 1 {
    send(me-1, A(1));
    recv(me-1, A(0));
  } else myLo = 2;
  for i in myLo..myHi do
    B(i) = (A(i-1)+A(i+1))/2;
}
```

NAS MG Stencil in Fortran + MPI, in Chapel

```
use caf_intrinsics

 implicit none

 include 'cafnpb.h'
 include 'globals.h'

 integer n1, n2, n3, kk
 double precision u(n1,n2,n3)
 integer axis

 if( .not. dead(kk) )then
    do  axis = 1, 3
       if( nprocs .ne. 1) then
          call sync_all()
          call give3( axis, +1, u, n1, n2, n3,
        kk )
          call give3( axis, -1, u, n1, n2, n3,
        kk )
          call sync_all()
          call take3( axis, -1, u, n1, n2, n3 )
          call take3( axis, +1, u, n1, n2, n3 )
       else
          call comm1p( axis, u, n1, n2, n3, kk
        )
       endif
    enddo
 else
    do  axis = 1, 3
       call sync_all()
       call sync_all()
    enddo
    call zero3(u,n1,n2,n3)
 endif
 return
 end

 subroutine give3( axis, dir, u, n1, n2, n3, k
        )
 use caf_intrinsics


 implicit none

 include 'cafnpb.h'
 include 'globals.h'  subroutine
        comm3(u,n1,n2,n3,kk)

 integer axis, dir, n1, n2, n3, k, ierr
 double precision u( n1, n2, n3 )

 integer i3, i2, i1, buff_len,buff_id

 buff_id = 2 + dir
 buff_len = 0

 if( axis .eq.  1 )then
    if( dir .eq. -1 )then

       do  i3=2,n3-1
          do  i2=2,n2-1
             buff_len = buff_len + 1
             buff(buff_len,buff_id ) = u( 2,
        i2,i3)
          enddo
       enddo


        buff(1:buff_len,buff_id+1)[nbr(axis,dir
        ,k)] =
>      buff(1:buff_len,buff_id)

    else if( dir .eq. +1 ) then

       do  i3=2,n3-1
          do  i2=2,n2-1
             buff_len = buff_len + 1
             buff(buff_len, buff_id ) = u( n1-
        1, i2,i3)
          enddo
       enddo


        buff(1:buff_len,buff_id+1)[nbr(axis,dir
        ,k)] =
>      buff(1:buff_len,buff_id)

    endif
 endif

 if( axis .eq.  2 )then
    if( dir .eq. -1 )then
       do  i3=2,n3-1
          do  i1=1,n1
             buff_len = buff_len + 1
             buff(buff_len, buff_id ) = u( i1,
        2,i3)
          enddo
       enddo

     buff(1:buff_len,buff_id+1)[nbr(axis,dir,k)] =
>      buff(1:buff_len,buff_id)
else if( dir .eq. +1 ) then

       do  i3=2,n3-1
          do  i1=1,n1
             buff_len = buff_len + 1
             buff(buff_len,  buff_id )= u(
        i1,n2-1,i3)
          enddo
       enddo


        buff(1:buff_len,buff_id+1)[nbr(axis,dir
        ,k)] =
>      buff(1:buff_len,buff_id)

    endif
 endif

 if( axis .eq.  3 )then
    if( dir .eq. -1 )then

       do  i2=1,n2
          do  i1=1,n1
             buff_len = buff_len + 1
             buff(buff_len, buff_id ) = u(
        i1,i2,2)
          enddo
       enddo


        buff(1:buff_len,buff_id+1)[nbr(axis,dir
        ,k)] =
>      buff(1:buff_len,buff_id)

    else if( dir .eq. +1 ) then

       do  i2=1,n2
          do  i1=1,n1
             buff_len = buff_len + 1
             buff(buff_len, buff_id ) = u(
        i1,i2,n3-1)
          enddo
       enddo


        buff(1:buff_len,buff_id+1)[nbr(axis,dir
        ,k)] =
>      buff(1:buff_len,buff_id)

    endif
 endif

 return
 end


 subroutine take3( axis, dir, u, n1, n2, n3 )
use caf_intrinsics


implicit none

include 'cafnpb.h'
include 'globals.h'

integer axis, dir, n1, n2, n3
double precision u( n1, n2, n3 )

integer buff_id, indx

integer i3, i2, i1

buff_id = 3 + dir
indx = 0

if( axis .eq.  1 )then
   if( dir .eq. -1 )then

      do  i3=2,n3-1
         do  i2=2,n2-1
            indx = indx + 1
            u(n1,i2,i3) = buff(indx, buff_id )
         enddo
      enddo

   else if( dir .eq. +1 ) then

      do  i3=2,n3-1
         do  i2=2,n2-1
            indx = indx + 1
            u(1,i2,i3) = buff(indx, buff_id )
         enddo
      enddo

   endif
endif

if( axis .eq.  2 )then
   if( dir .eq. -1 )then

      do  i3=2,n3-1
         do  i1=1,n1
            indx = indx + 1
            u(i1,n2,i3) = buff(indx, buff_id )
         enddo
      enddo

   else if( dir .eq. +1 ) then

      do  i3=2,n3-1
         do  i1=1,n1
            indx = indx + 1
            u(i1,1,i3) = buff(indx, buff_id )
         enddo
      enddo

   endif
endif

if( axis .eq.  3 )then
   if( dir .eq. -1 )then

      do  i2=1,n2
         do  i1=1,n1
            indx = indx + 1
            u(i1,i2,n3) = buff(indx, buff_id )
         enddo
      enddo

   else if( dir .eq. +1 ) then

      do  i2=1,n2
         do  i1=1,n1
            indx = indx + 1
            u(i1,i2,1) = buff(indx, buff_id )
         enddo
      enddo

   endif
endif

return
end


subroutine comm1p( axis, u, n1, n2, n3, kk )
use caf_intrinsics


implicit none

include 'cafnpb.h'
include 'globals.h'

integer axis, dir, n1, n2, n3
double precision u( n1, n2, n3 )

integer i3, i2, i1, buff_len,buff_id
integer i, kk, indx

dir = -1

buff_id = 3 + dir
buff_len = nm2

do  i=1,nm2
   buff(i,buff_id) = 0.0D0
enddo


dir = +1

buff_id = 3 + dir
buff_len = nm2

do  i=1,nm2
   buff(i,buff_id) = 0.0D0
enddo

dir = +1

buff_id = 2 + dir
buff_len = 0

if( axis .eq.  1 )then
   do  i3=2,n3-1
      do  i2=2,n2-1
         buff_len = buff_len + 1
         buff(buff_len, buff_id ) = u( n1-1,
   i2,i3)
      enddo
   enddo
endif

if( axis .eq.  2 )then
   do  i3=2,n3-1
      do  i1=1,n1
         buff_len = buff_len + 1
         buff(buff_len,  buff_id )= u( i1,n2-
   1,i3)
      enddo
   enddo
endif

if( axis .eq.  3 )then
      do  i2=1,n2
         do  i1=1,n1
            buff_len = buff_len + 1
            buff(buff_len, buff_id ) = u(
   i1,i2,n3-1)
         enddo
      enddo
endif

dir = -1

buff_id = 2 + dir
buff_len = 0

if( axis .eq.  1 )then
   do  i3=2,n3-1
      do  i2=2,n2-1
         buff_len = buff_len + 1
         buff(buff_len,buff_id ) = u( 2,
   i2,i3)
      enddo
   enddo
endif

if( axis .eq.  2 )then
   do  i3=2,n3-1
      do  i1=1,n1
         buff_len = buff_len + 1
         buff(buff_len, buff_id ) = u( i1,
   2,i3)
      enddo
   enddo
endif

if( axis .eq.  3 )then
   do  i2=1,n2
      do  i1=1,n1
         buff_len = buff_len + 1
         buff(buff_len, buff_id ) = u(
   i1,i2,2)
      enddo
   enddo
endif

do  i=1,nm2
   buff(i,4) = buff(i,3)
   buff(i,2) = buff(i,1)
enddo

dir = -1

buff_id = 3 + dir
indx = 0

if( axis .eq.  1 )then
   do  i3=2,n3-1
      do  i2=2,n2-1
         indx = indx + 1
         u(n1,i2,i3) = buff(indx, buff_id )
      enddo
   enddo
endif

if( axis .eq.  2 )then
   do  i3=2,n3-1
      do  i1=1,n1
         indx = indx + 1
         u(i1,n2,i3) = buff(indx, buff_id )
      enddo
   enddo
endif

if( axis .eq.  3 )then
   do  i2=1,n2
      do  i1=1,n1
         indx = indx + 1
         u(i1,i2,n3) = buff(indx, buff_id )
      enddo
   enddo
endif

dir = +1

buff_id = 3 + dir
indx = 0

   if( axis .eq.  1 )then
      do  i3=2,n3-1
         do  i2=2,n2-1
            indx = indx + 1
            u(1,i2,i3) = buff(indx, buff_id )
         enddo
      enddo
   endif
if( axis .eq.  2 )then
   do  i3=2,n3-1
      do  i1=1,n1
         indx = indx + 1
         u(i1,1,i3) = buff(indx, buff_id )
      enddo
   enddo
endif

if( axis .eq.  3 )then
   do  i2=1,n2
      do  i1=1,n1
         indx = indx + 1
         u(i1,i2,1) = buff(indx, buff_id )
      enddo
   enddo
endif

return
end

subroutine
      rprj3(r,m1k,m2k,m3k,s,m1j,m2j,m3j,k)
implicit none
include 'cafnpb.h'
include 'globals.h'

integer m1k, m2k, m3k, m1j, m2j, m3j,k

double precision r(m1k,m2k,m3k),
      s(m1j,m2j,m3j)
integer j3, j2, j1, i3, i2, i1, d1, d2, d3, j
double precision x1(m), y1(m), x2,y2

if(m1k.eq.3)then
   d1 = 2
else
   d1 = 1
endif

if(m2k.eq.3)then
   d2 = 2
else
   d2 = 1
endif

if(m3k.eq.3)then
   d3 = 2
else
   d3 = 1
endif

do  j3=2,m3j-1
   i3 = 2*j3-d3
   do  j2=2,m2j-1
      i2 = 2*j2-d2
      do j1=2,m1j
         i1 = 2*j1-d1
         x1(i1-1) = r(i1-1,i2-1,i3  ) + r(i1-
   1,i2+1,i3  )
>              + r(i1-1,i2,  i3-1) + r(i1-
   1,i2,  i3+1)
         y1(i1-1) = r(i1-1,i2-1,i3-1) + r(i1-
   1,i2-1,i3+1)
>              + r(i1-1,i2+1,i3-1) + r(i1-
   1,i2+1,i3+1)
      enddo
      do  j1=2,m1j-1
         i1 = 2*j1-d1
         y2 = r(i1,  i2-1,i3-1) + r(i1,  i2-
   1,i3+1)
>           + r(i1,  i2+1,i3-1) + r(i1,
   i2+1,i3+1)
         x2 = r(i1,  i2-1,i3  ) + r(i1,  i2+1,i3
   )
>           + r(i1,  i2,  i3-1) + r(i1,  i2,
   i3+1)
         s(j1,j2,j3) =
>              0.5D0 * r(i1,i2,i3)
>            + 0.25D0 * (r(i1-1,i2,i3) +
   r(i1+1,i2,i3) + x2)
>            + 0.125D0 * ( x1(i1-1) + x1(i1+1) +
   y2)
>            + 0.0625D0 * ( y1(i1-1) + y1(i1+1) )
      enddo
   enddo
enddo
j = k-1
call comm3(s,m1j,m2j,m3j,j)
return
end
```

```
def rprj3(S, R) {
  const Stencil = [-1..1, -1..1, -1..1],
        W: [0..3] real = (0.5, 0.25, 0.125, 0.0625),
        W3D = [(i,j,k) in Stencil] W((i!=0)+(j!=0)+(k!=0));

  forall inds in S.domain do
    S(inds) =
      + reduce [offset in Stencil] (W3D(offset) *
                                    R(inds + offset*R.stride));
}
```

Outline

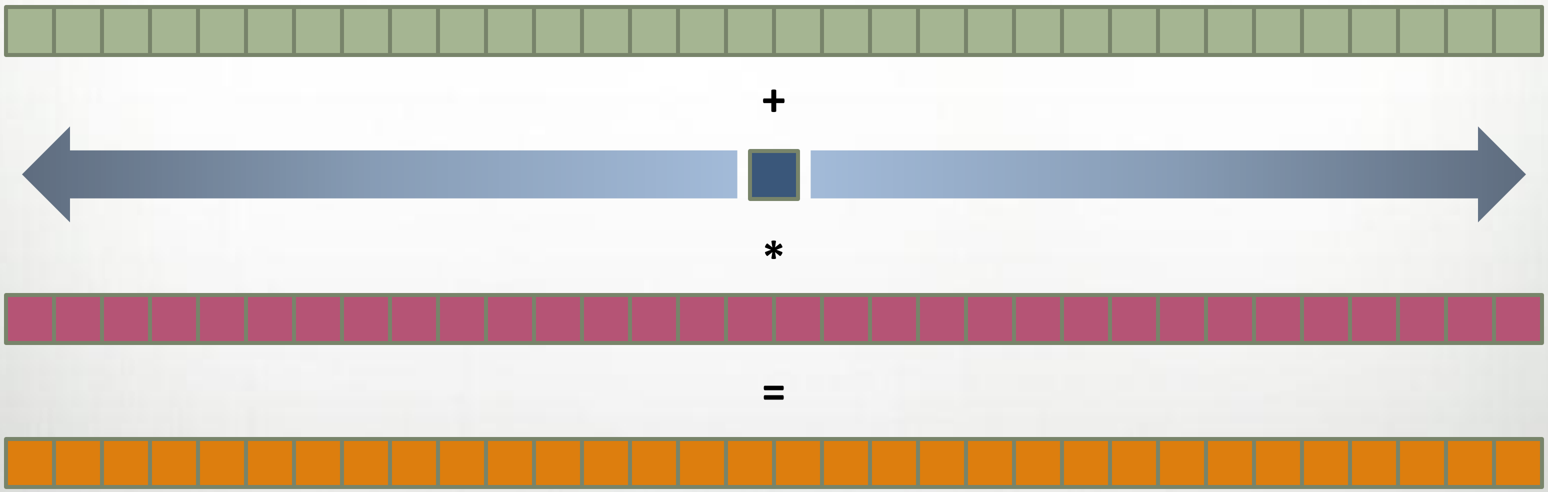
- What is Chapel?
- Chapel's Parallel Programming Model
- HPCC STREAM Triad in Chapel
- HPCC RA in Chapel
- Summary and Future Work

Introduction to STREAM Triad

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

Compute:

```
forall i in 1..m do
  A(i) = B(i) + α * C(i);
```

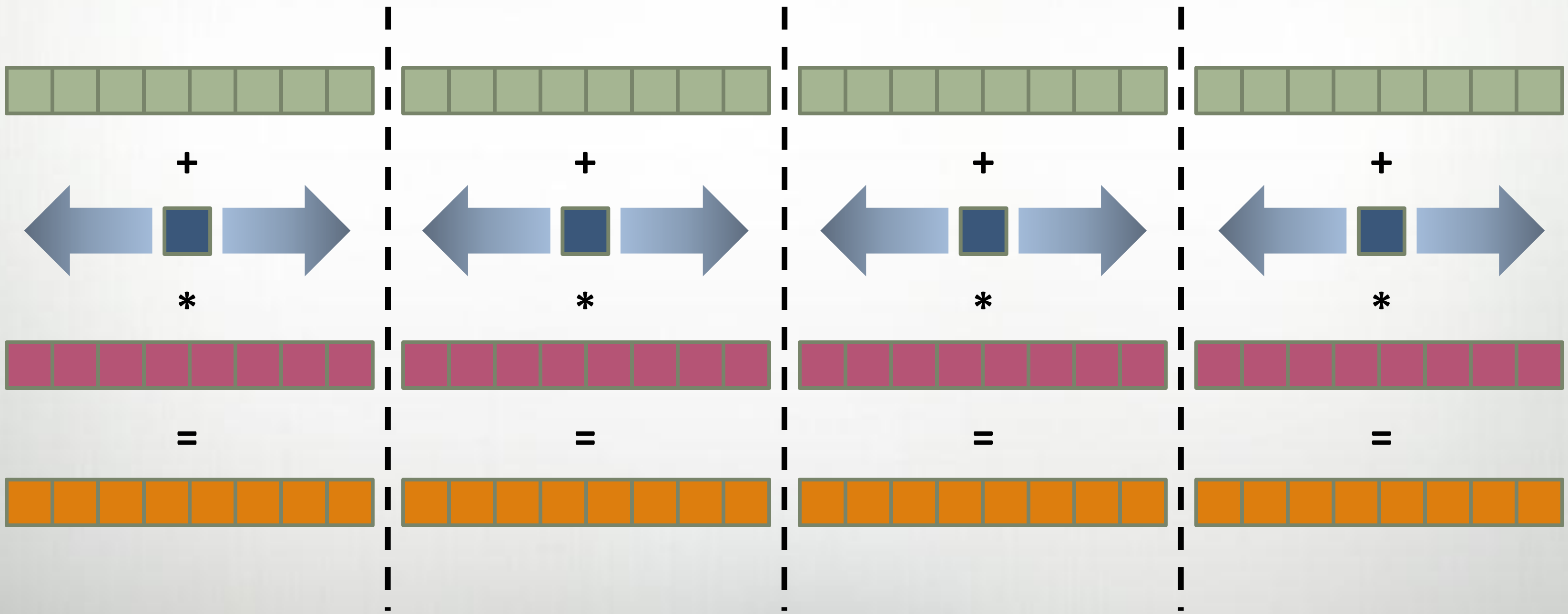


Distributed Parallelization of STREAM Triad

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

Compute:

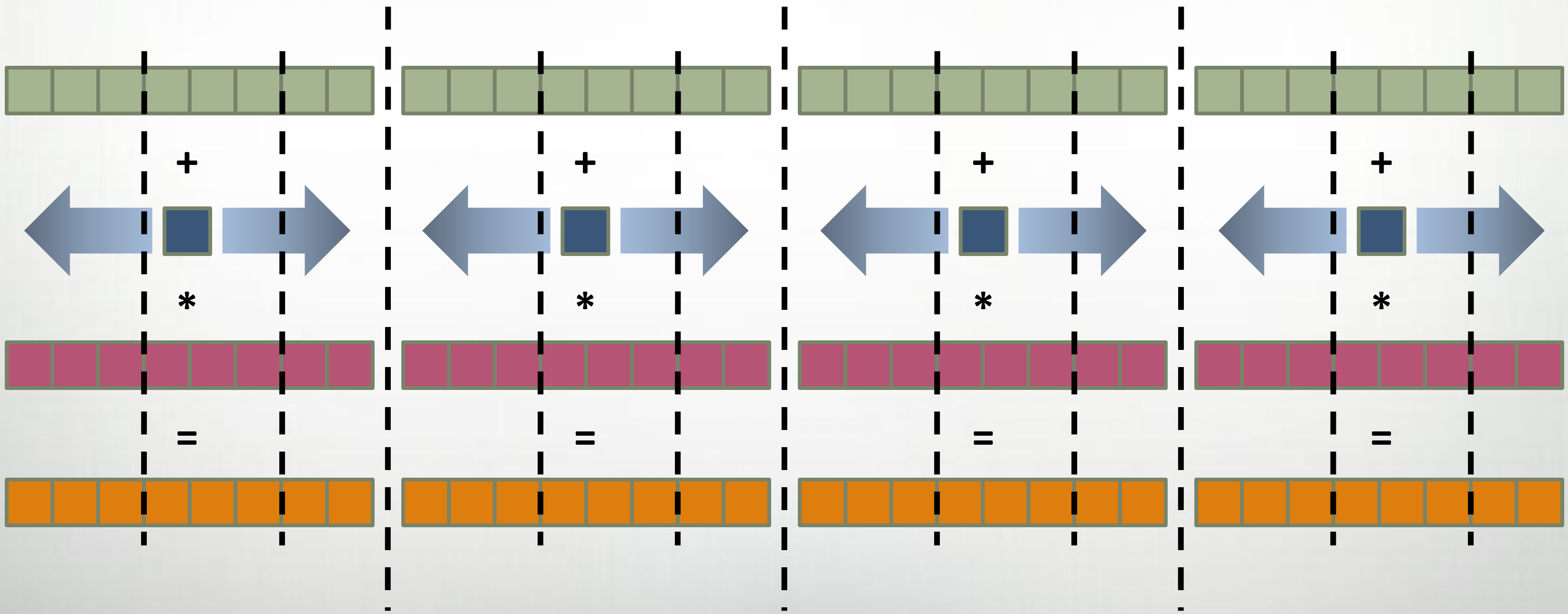
```
forall i in 1..m do
  A(i) = B(i) + α * C(i);
```



Further Parallelization of STREAM Triad

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

Compute:
```
forall i in 1..m do
  A(i) = B(i) + α * C(i);
```



STREAM Triad in Chapel: Single Locale

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

Compute:
```
forall i in 1..m do
  A(i) = B(i) + α * C(i);
```

```
config const m: int(64) = ...;
const alpha: real = 3.0;
const ProblemSpace: domain(1,int(64)) = [1..m];
var A, B, C: [ProblemSpace] real;

forall i in ProblemSpace do
  A(i) = B(i) + alpha * C(i);
```

STREAM Triad in Chapel: Single Locale

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

Compute:
```
forall i in 1..m do
  A(i) = B(i) + α * C(i);
```

```
config const m: int(64) = ...;
const alpha: real = 3.0;
const ProblemSpace: domain(1,int(64)) = [1..m];
var A, B, C: [ProblemSpace] real;

A = B + alpha * C;
```

More concise variation using whole array operations

STREAM Triad in Chapel: Single Locale

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

Compute:
```
forall i in 1..m do
  A(i) = B(i) + α * C(i);
```

```
config const m: int(64) = ...;
const alpha: real = 3.0;
const ProblemSpace: domain(1,int(64)) = [1..m];
var A, B, C: [ProblemSpace] real;

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

Variation that iterates directly over the arrays

STREAM Triad in Chapel: Multi-Locale

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

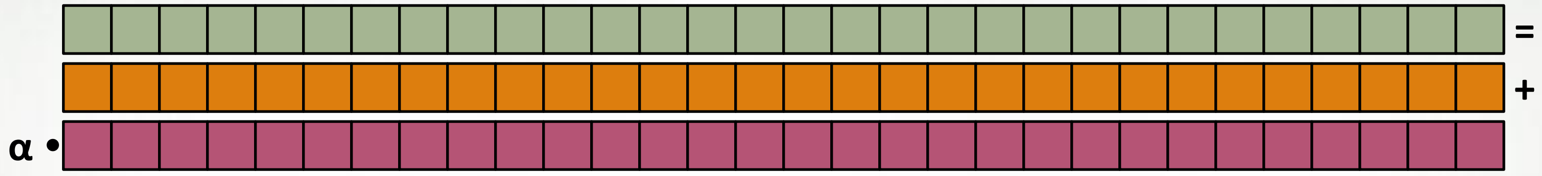
Compute:
```
forall i in 1..m do
  A(i) = B(i) + α * C(i);
```

```
config const m: int(64) = ..., tpl = ...;
const alpha: real = 3.0;
const BlockDist = new Block(1,int(64),[1..m],tpl);
const ProblemSpace: domain(1, int(64))
                          distributed BlockDist = [1..m];
var A, B, C: [ProblemSpace] real;

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

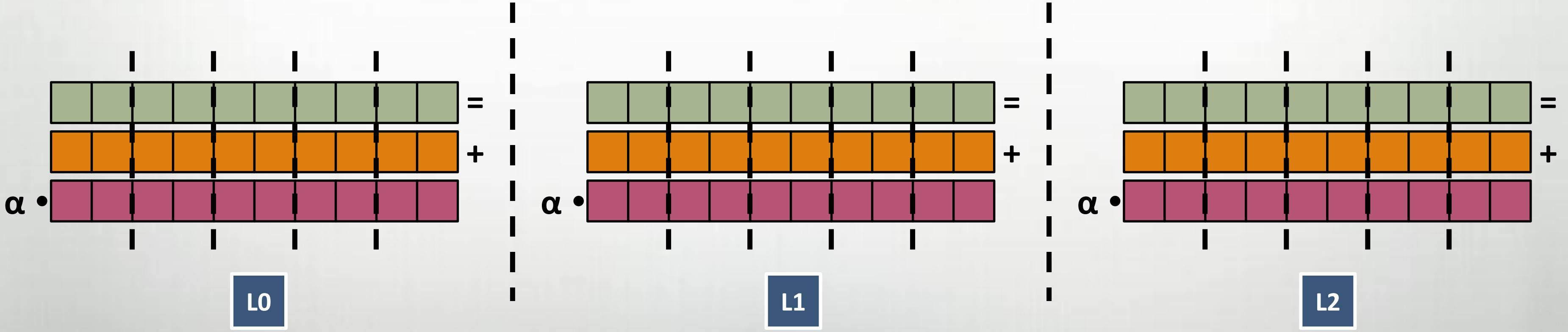
What is a Distribution?

A "recipe" for distributed arrays that...

Instructs the compiler how to Map the global view...



...to a fragmented, per-processor implementation



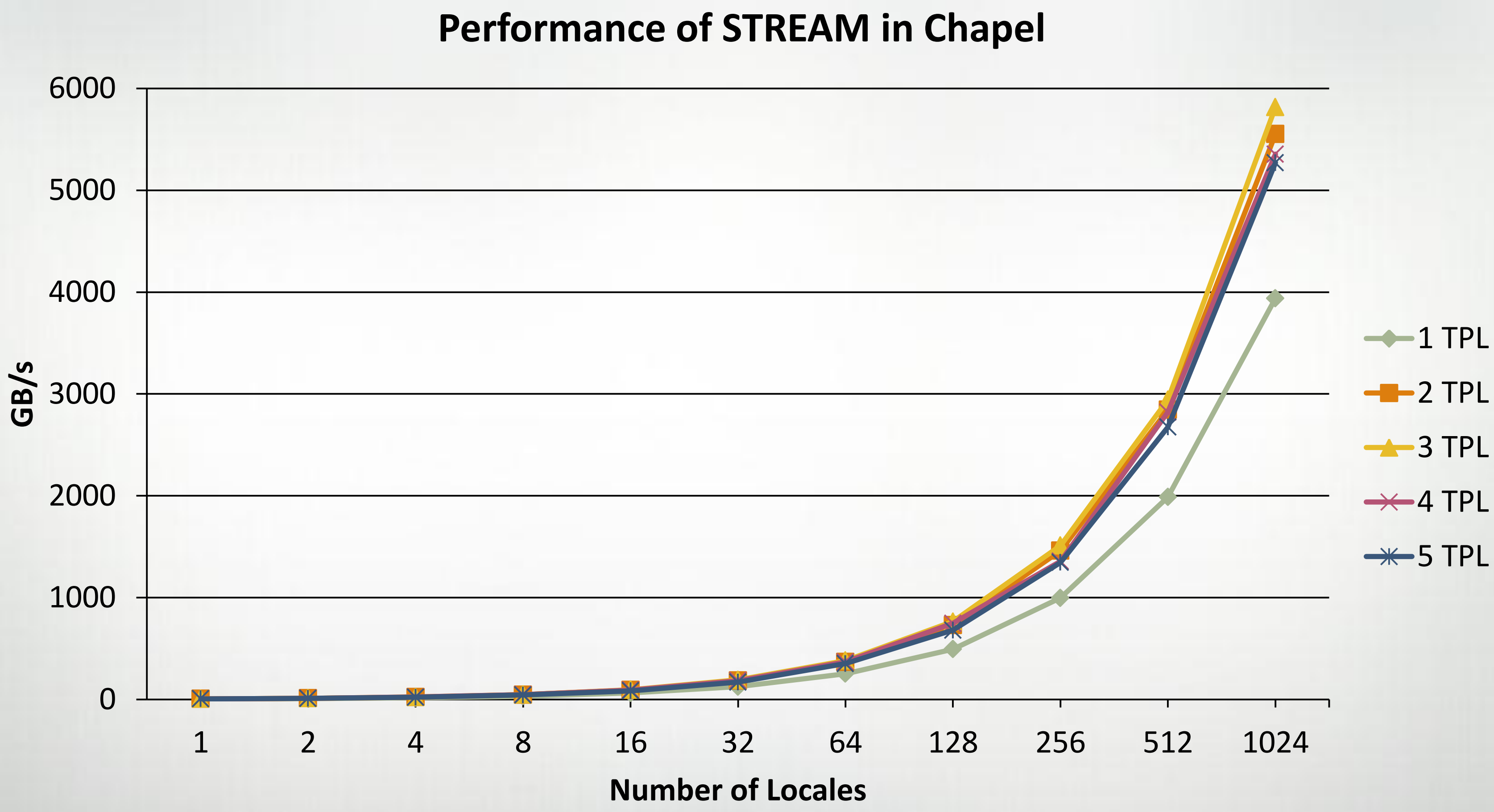
STREAM Triad in Chapel: Multi-Locale

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

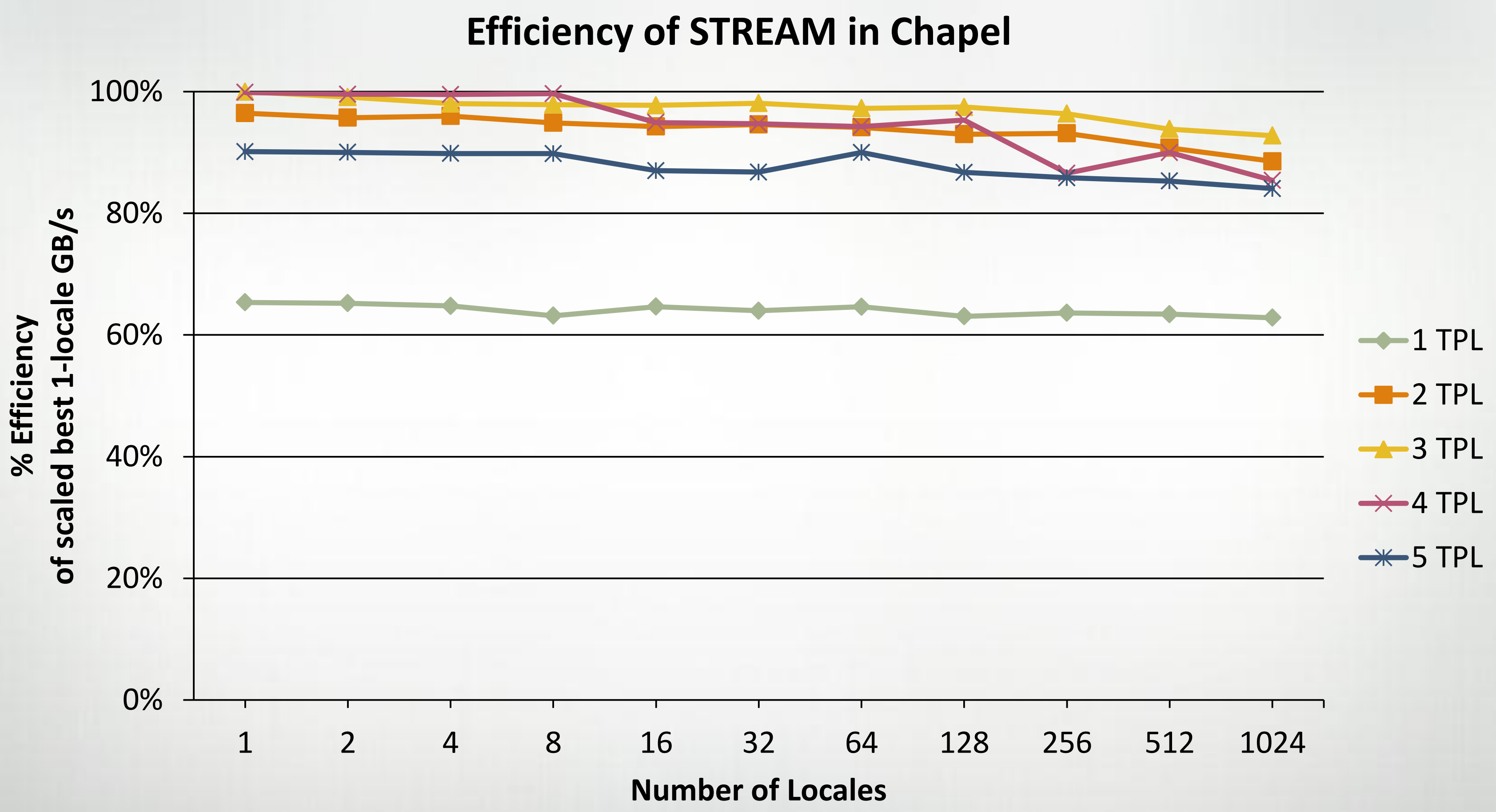
Compute:
```
forall i in 1..m do
  A(i) = B(i) + α * C(i);
```

```
config const m: int(64) = ..., tpl = ...;
const alpha: real = 3.0;
const BlockDist = new Block(1,int(64),[1..m],tpl);
const ProblemSpace: domain(1, int(64))
                    distributed BlockDist = [1..m];
var A, B, C: [ProblemSpace] real;

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

HPCC STREAM Performance



HPCC STREAM Efficiency



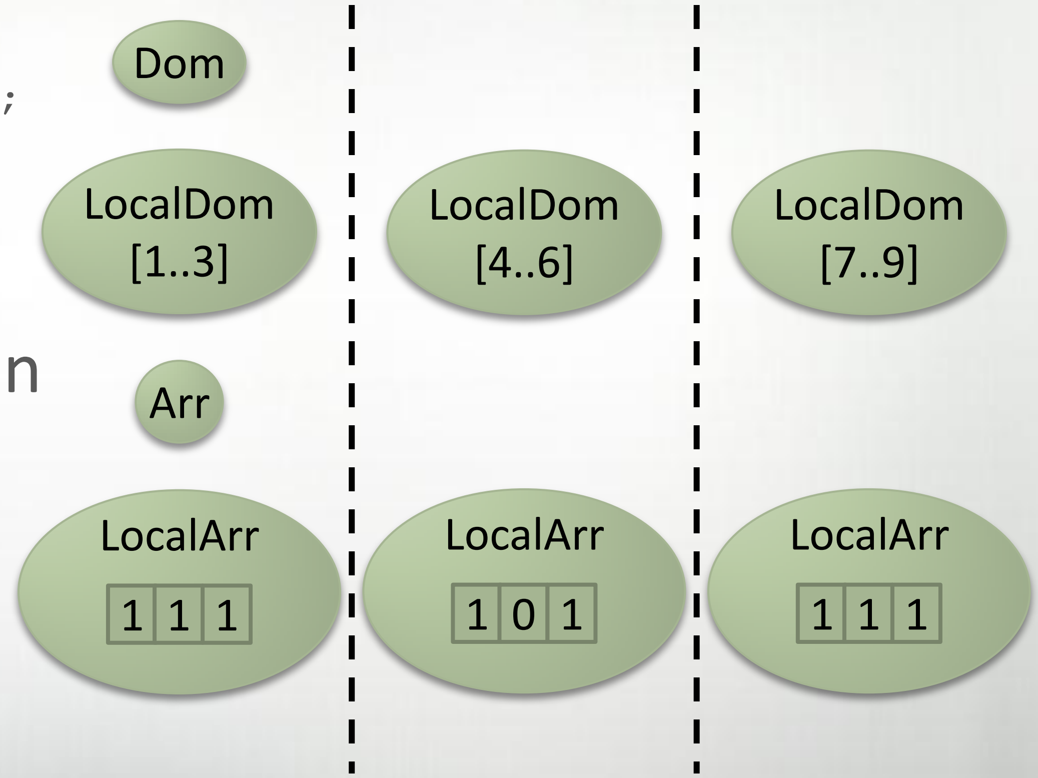
Optimization: Privatization

Simple example

```
var Dist: Block(1,int(64));
var Dom: domain(1,int(64))
             distributed Dist;
var Arr: [Dom] int;
```

Reference to local data requires communication

```
on Locales(1) {
  Arr(5) = 0;
}
```



Outline

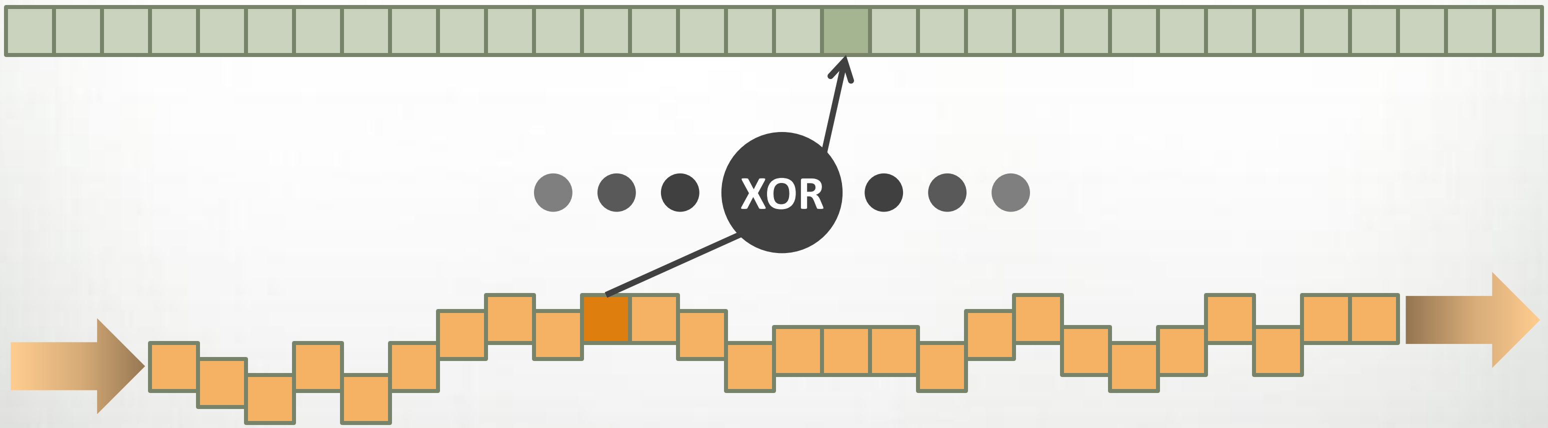
- What is Chapel?
- Chapel's Parallel Programming Model
- HPCC STREAM Triad in Chapel
- HPCC RA in Chapel
- Summary and Future Work

Introduction to RA

Given: *m*-element table *T* (where $m = 2^n$)

Compute:

```
forall r in RandomUpdates do
  T(r & (m-1)) ^= r;
```

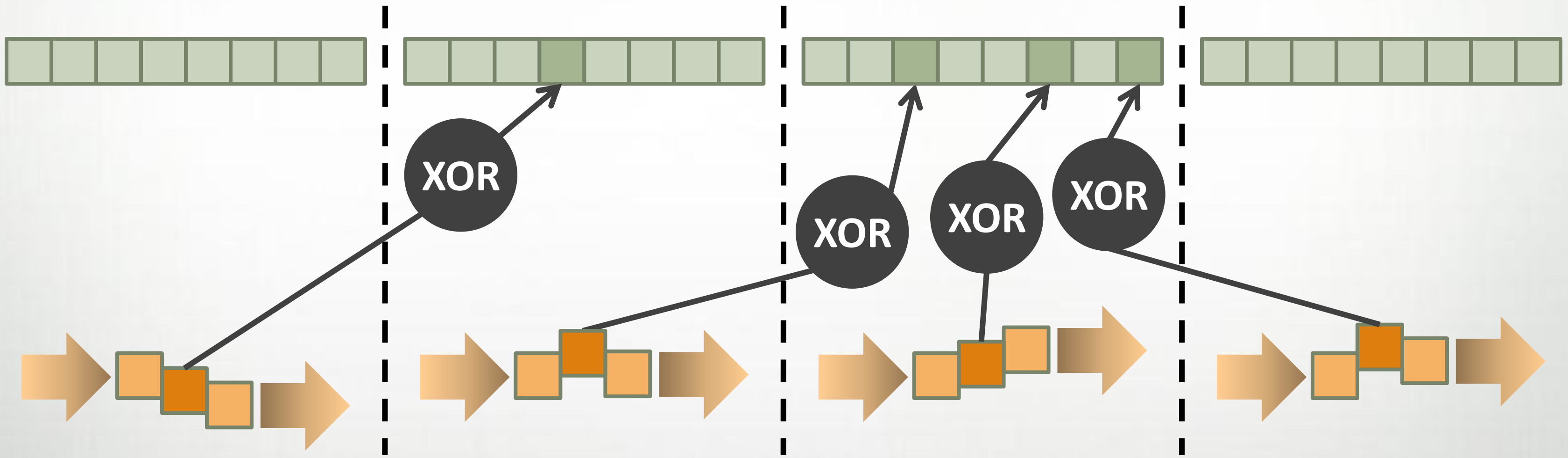


Distributed Parallelization of RA

Given: *m*-element table *T* (where $m = 2^n$)

Compute:

```
forall r in RandomUpdates do
  T(r & (m-1)) ^= r;
```



RA in Chapel: Single Locale

Given: *m*-element table *T* (where $m = 2^n$)

Compute:
```
forall r in RandomUpdates do
  T(r & (m-1)) ^= r;
```

```
config const m = ..., N_U = ...;
const TableSpace: domain(1,uint(64)) = [0..m-1],
      Updates: domain(1,uint(64)) = [0..N_U-1];
var T: [TableSpace] uint(64);

forall (i,r) in (Updates,RAStream()) do
  T(r & (m-1)) ^= r;
```

RA in Chapel: Multi-Locale

Given: *m*-element table *T* (where $m = 2^n$)

Compute:
```
forall r in RandomUpdates do
  T(r & (m-1)) ^= r;
```

```
config const m = ..., N_U = ..., tpl = ...;
const TableDist = new Block(1,uint(64),[0..m-1],tpl),
      UpdateDist = new Block(1,uint(64),[0..N_U-1],tpl),
      TableSpace: domain(1,uint(64))
                    distributed TableDist = [0..m-1],
      Updates: domain(1,uint(64))
                 distributed UpdateDist = [0..N_U-1];
var T: [TableSpace] uint(64);

forall (i,r) in (Updates,RAStream()) do
  on T(r & (m-1)) do
    T(r & (m-1)) ^= r;
```

RA in Chapel: Multi-Locale

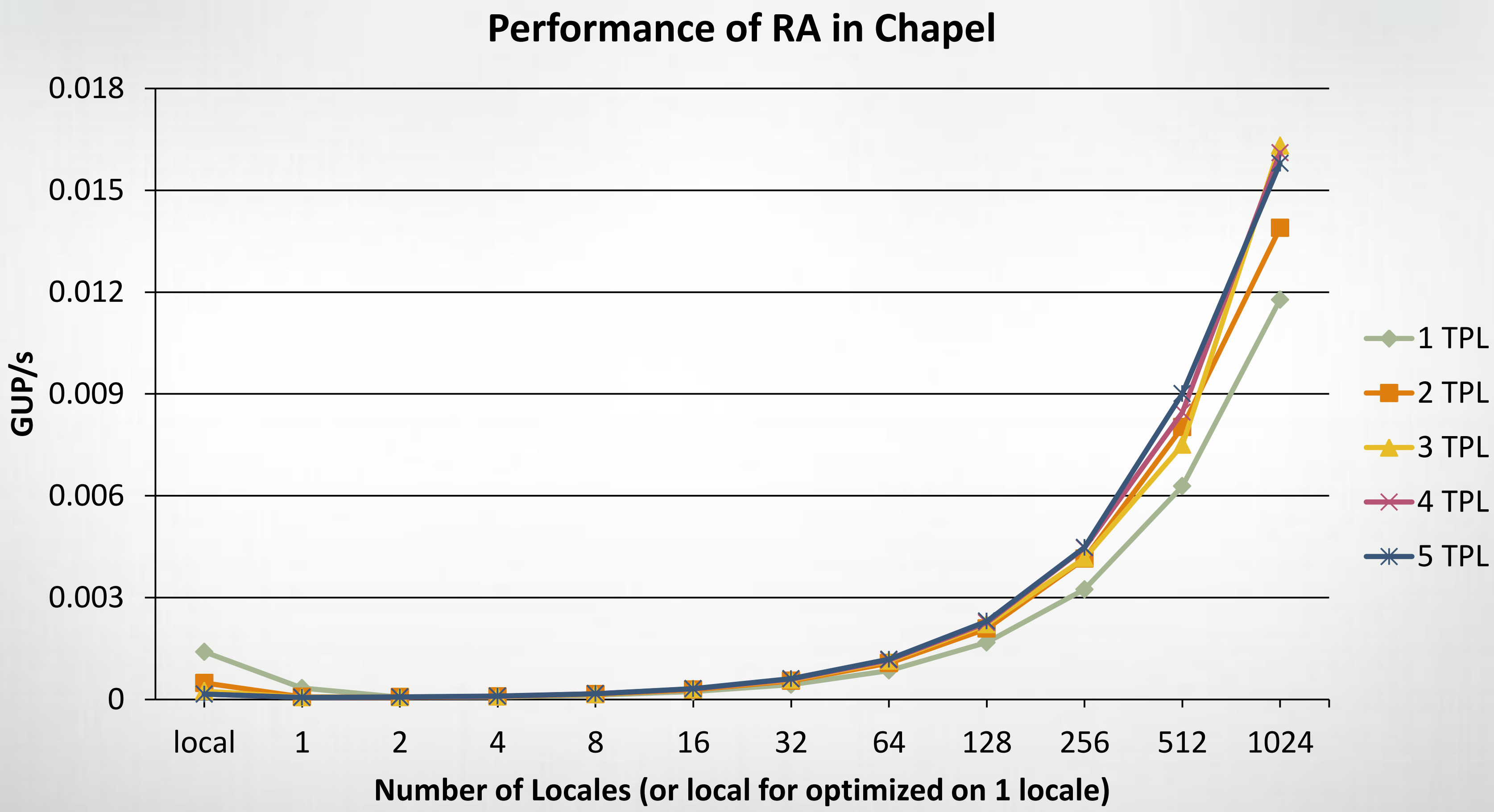
Given: *m*-element table *T* (where $m = 2^n$)

Compute:
```
forall r in RandomUpdates do
  T(r & (m-1)) ^= r;
```

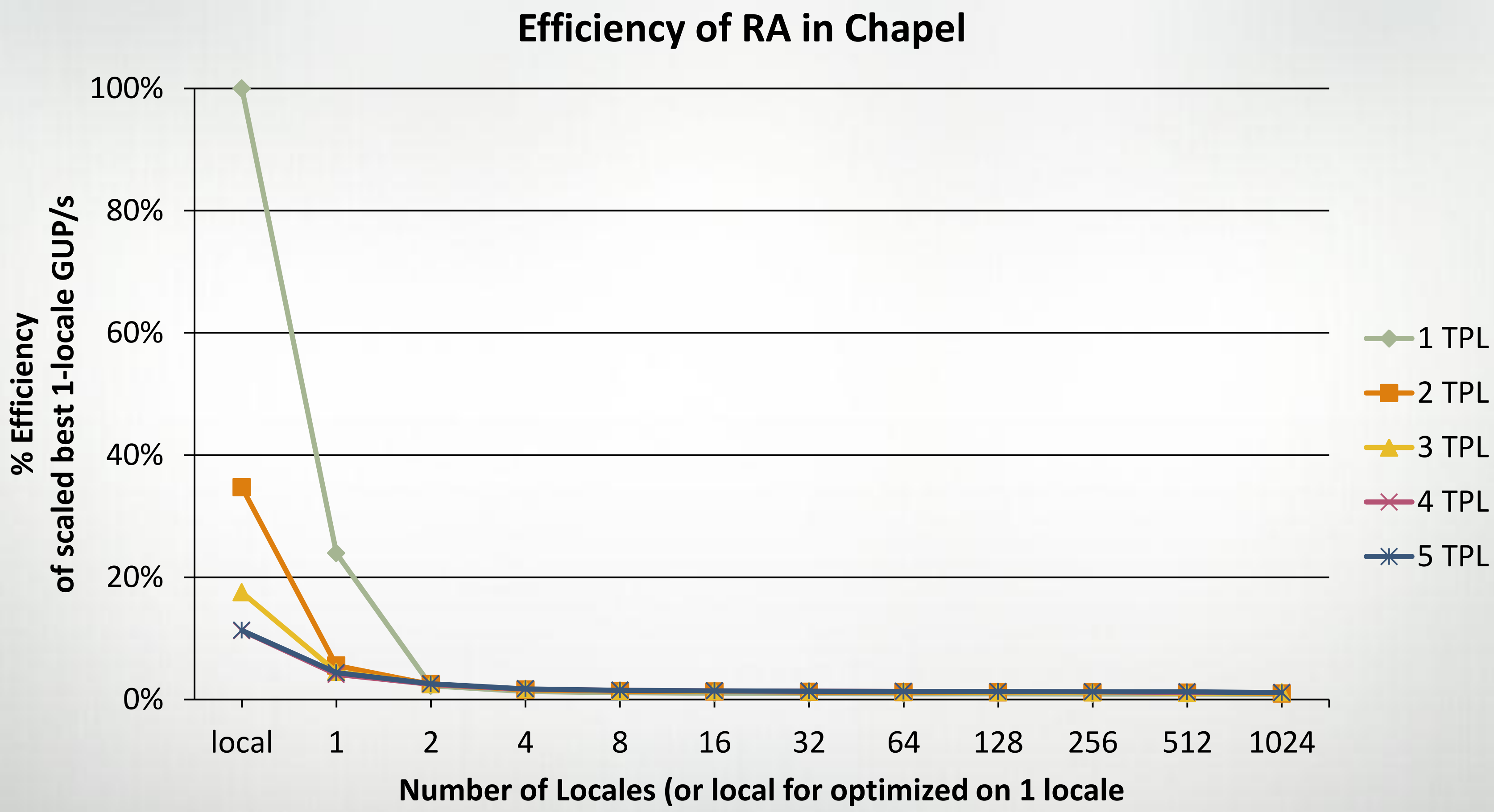
```
config const m = ..., N_U = ..., tpl = ...;
const TableDist = new Block(1,uint(64),[0..m-1],tpl),
      UpdateDist = new Block(1,uint(64),[0..N_U-1],tpl),
      TableSpace: domain(1,uint(64))
                    distributed TableDist = [0..m-1],
      Updates: domain(1,uint(64))
                 distributed UpdateDist = [0..N_U-1];
var T: [TableSpace] uint(64);

forall (i,r) in (Updates,RAStream()) do
  on T.domain.dist.ind2loc(r & (m-1)) do
    T(r & (m-1)) ^= r;
```

Call ind2loc method directly

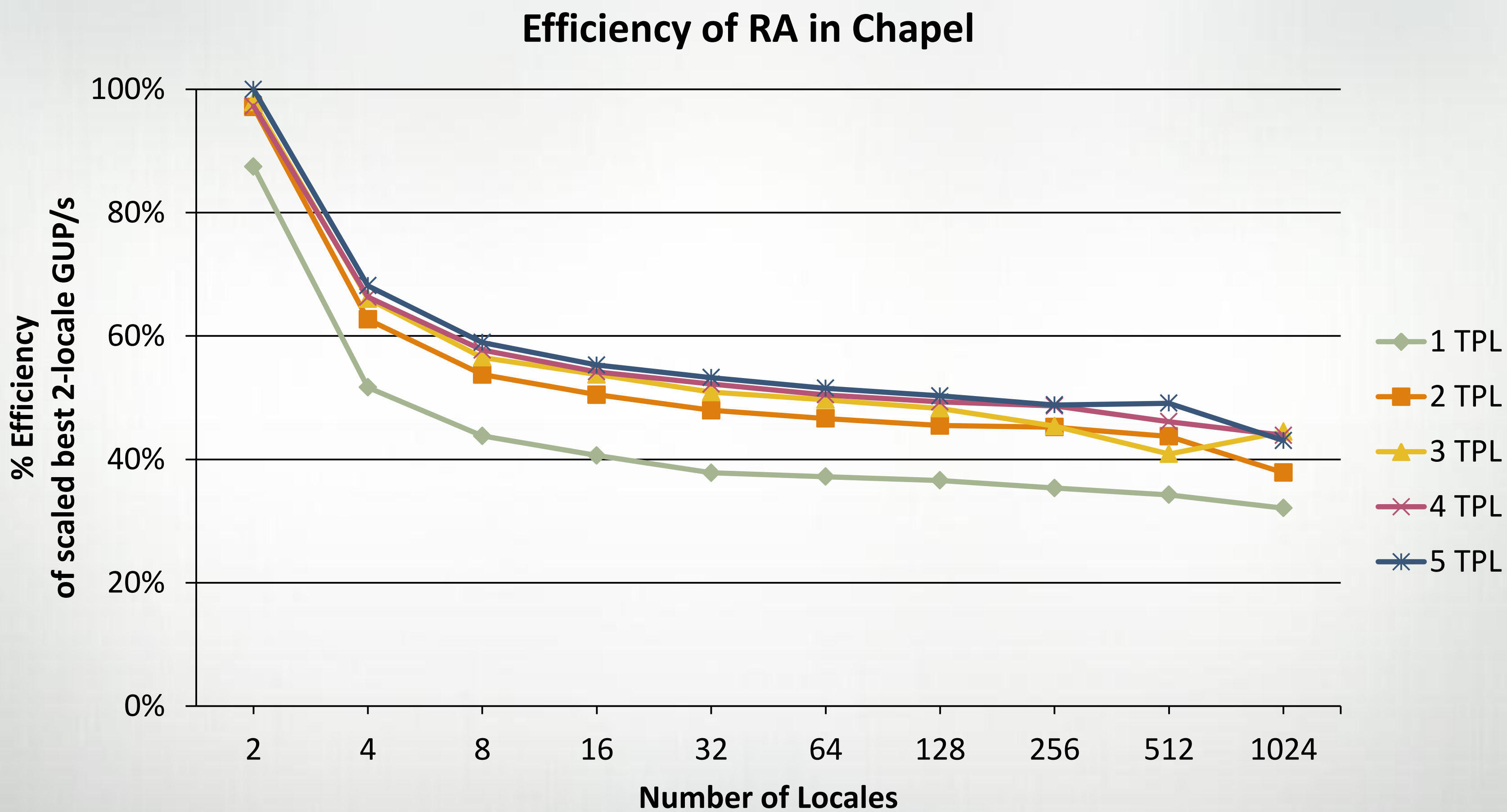
HPCC RA Performance



HPCC RA Efficiency I



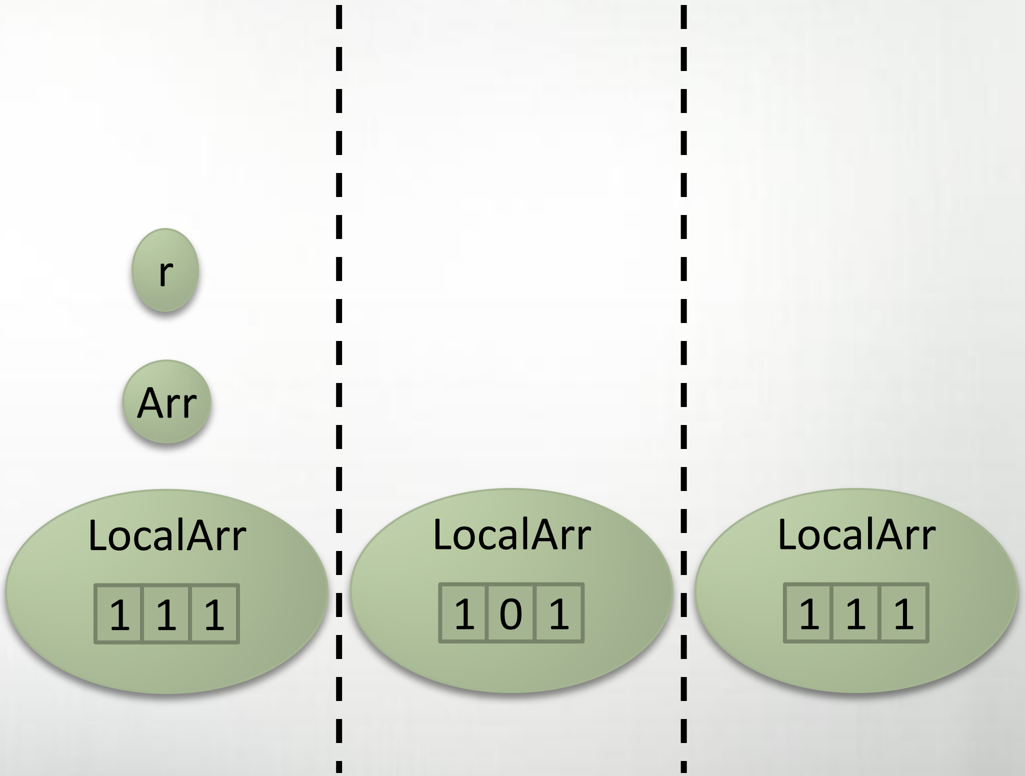
HPCC RA Efficiency II



Optimization: Remote Value Forwarding

Simple example

```
var Arr: [Dom] int;
var r: int;
on Locales(1) {
  Arr(r) ^= r;
}
```



Outline

- What is Chapel?
- Chapel's Parallel Programming Model
- HPCC STREAM Triad in Chapel
- HPCC RA in Chapel
- Summary and Future Work

Summary

The global-view programming model is easy to use.

- Shorter, more concise code
- Separation of concerns (partitioning)
- Easy to change data distributions

Distributions implement the global-view model.

- Flexible mechanism for experimentation
- Implementation of distributions is in Chapel

Future Work

- Optimizations
  - Within the compiler
  - Within the runtime
  - Within the distributions
- Complete implementation of Block distribution
- Implement new distributions
  - Cyclic, BlockCyclic, RecursiveBisection
- Experiment with variations of STREAM and RA

Questions?