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

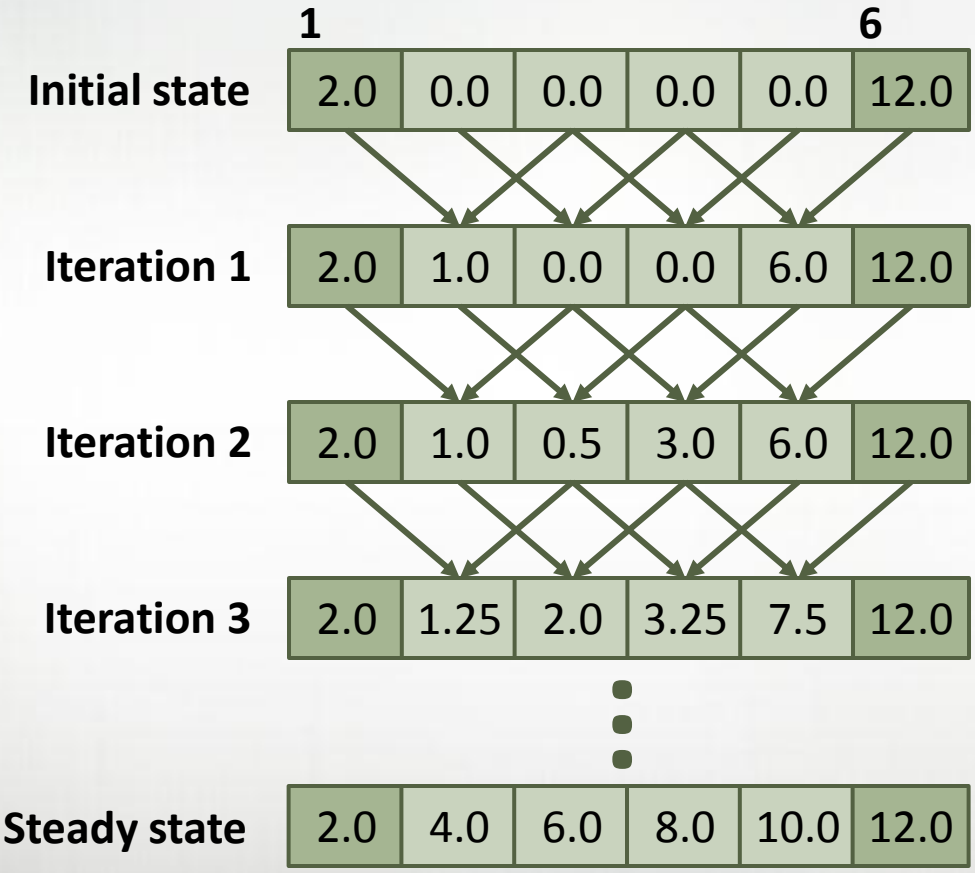
- SPMD models (Single Program, Multiple Data)

  Fragmented models with multiple copies of one program

- Global-view models

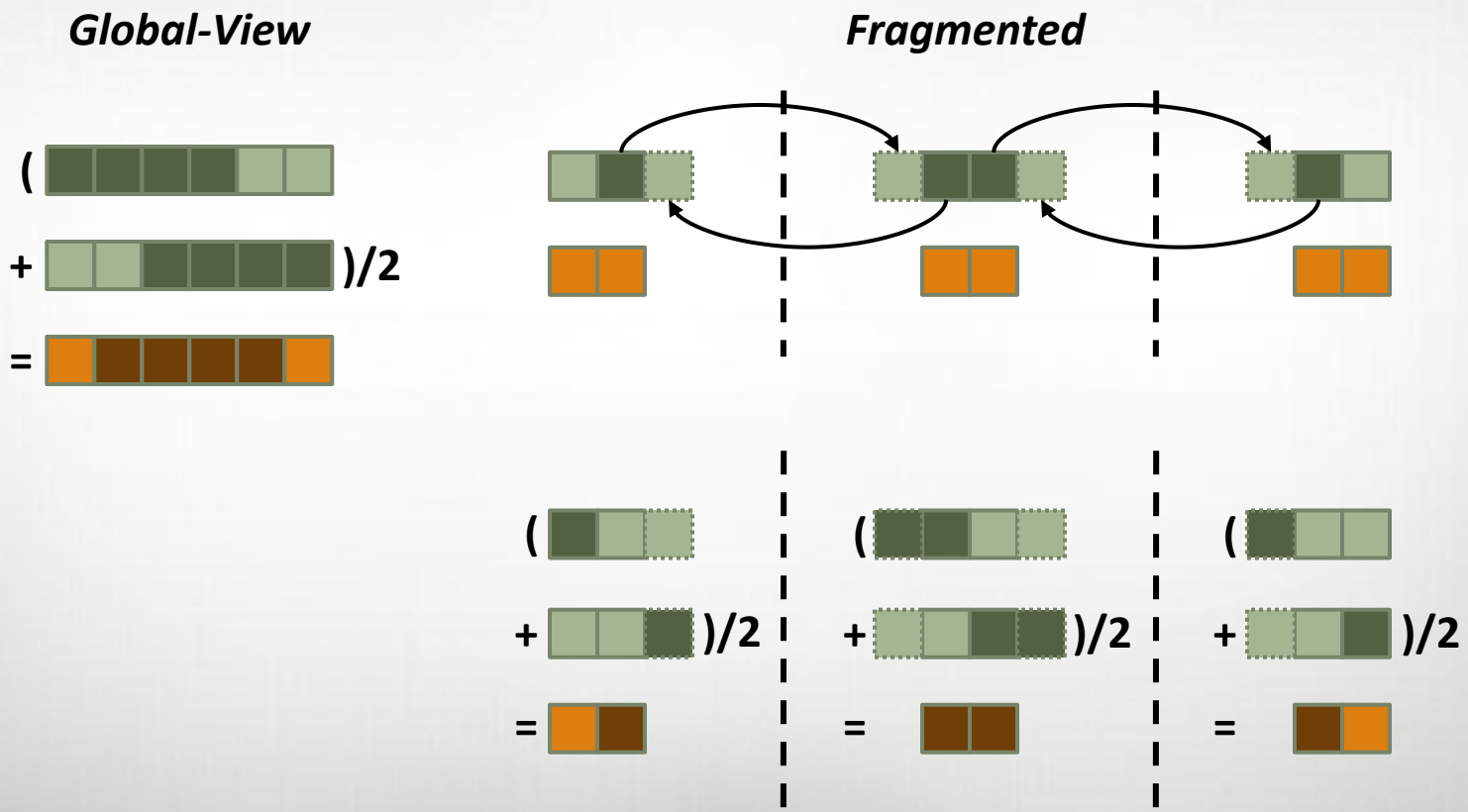
  Programmers write code to describe computation as a whole

3-Point Stencil Example (n=6)



3-Point Stencil Example

Global-View vs. Fragmented Computation

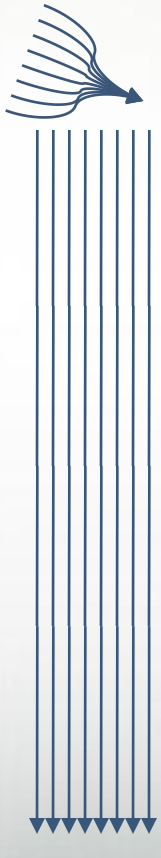


3-Point Stencil Example: Code

Global-View vs. Fragmented Code

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

Assumes p divides n

```
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

```
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));
}
```

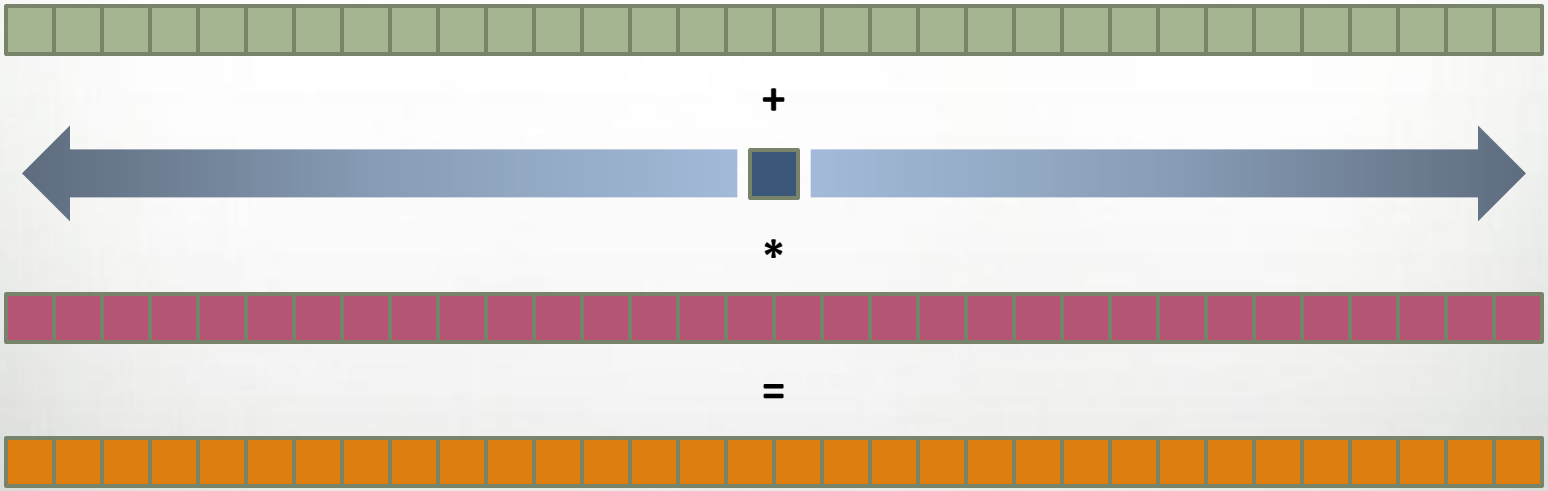
Outline

- 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*

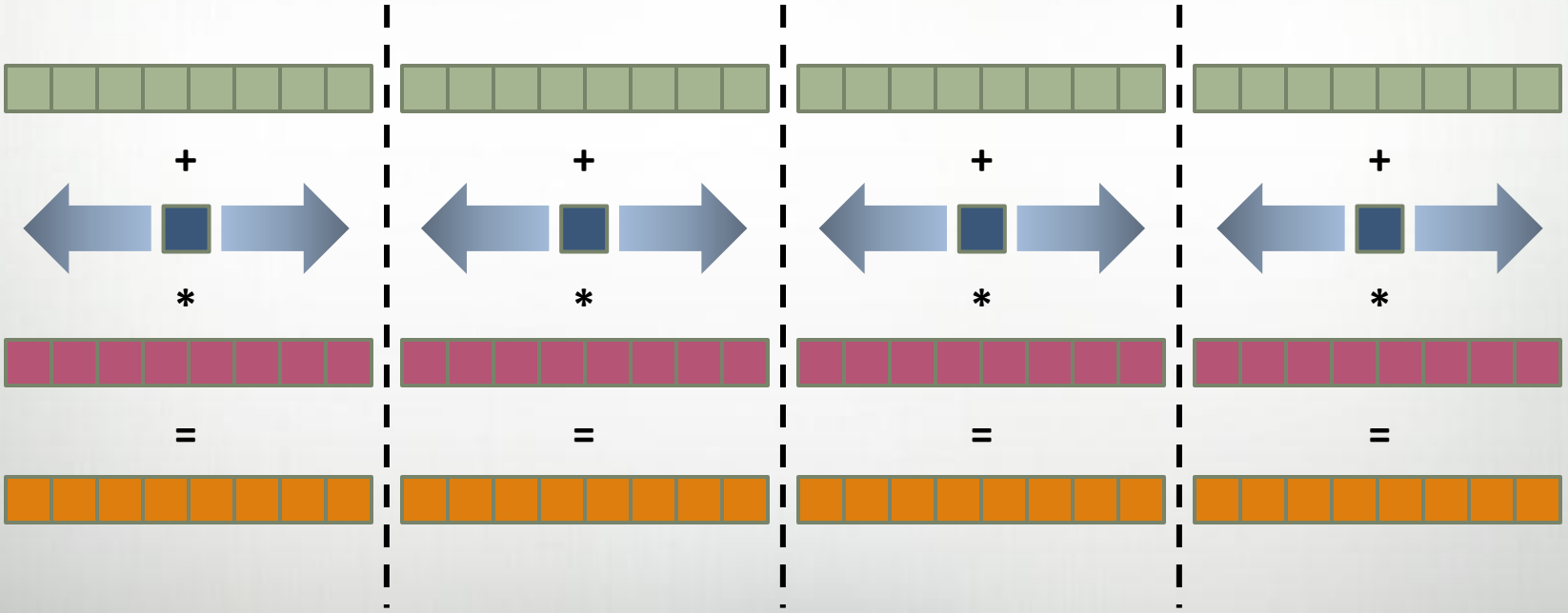
Compute:
```
forall i in 1..m do
  A(i) = B(i) + α * C(i);
```



Distributed Parallelization of STREAM Triad

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

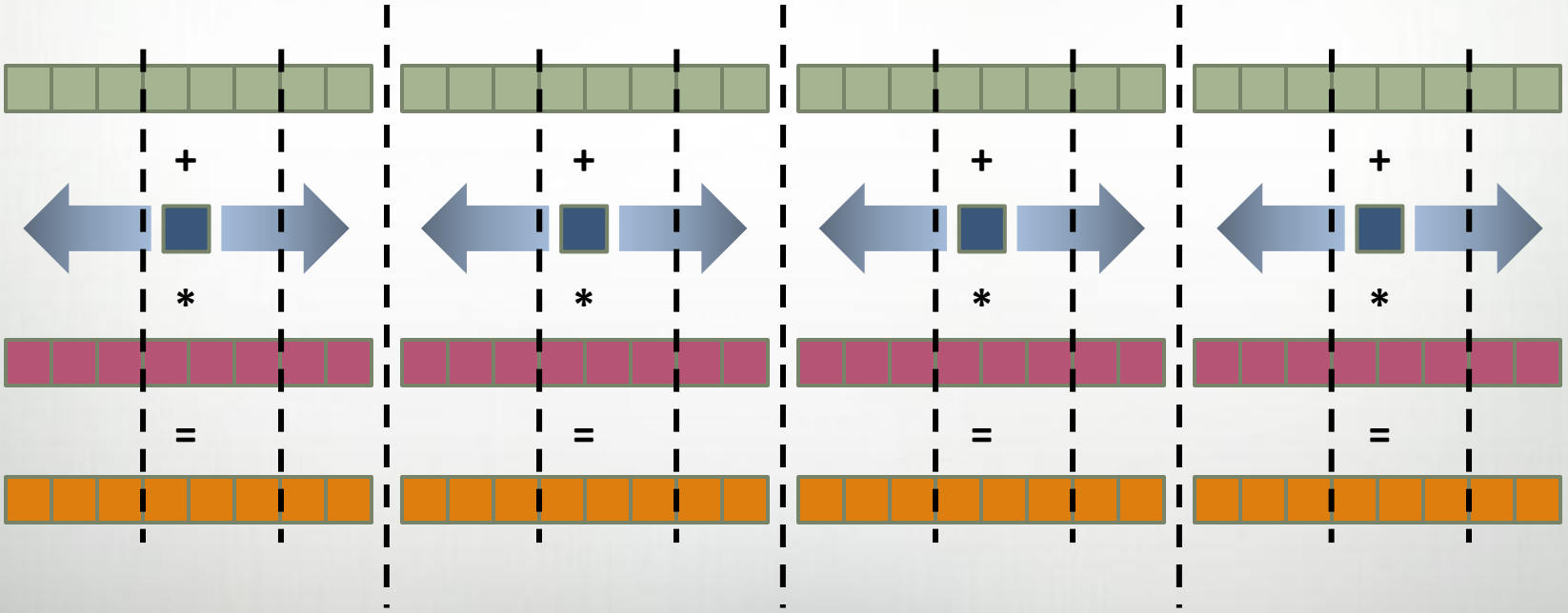
Compute:
```
forall i in 1..m do
  A(i) = B(i) + α * C(i);
```



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*

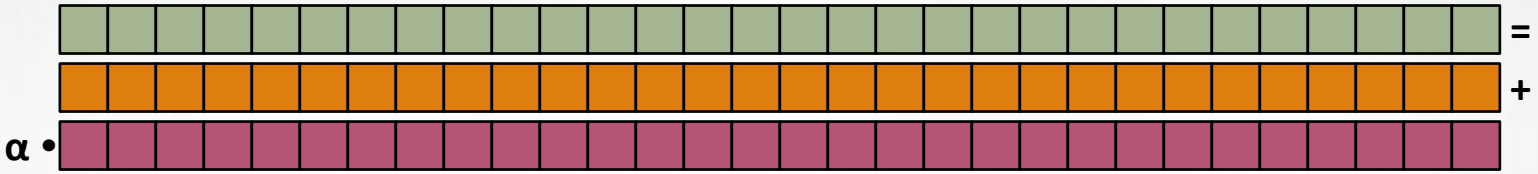
Compute: `forall i in 1..m do A(i) = B(i) + α * C(i);`

```
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;
```

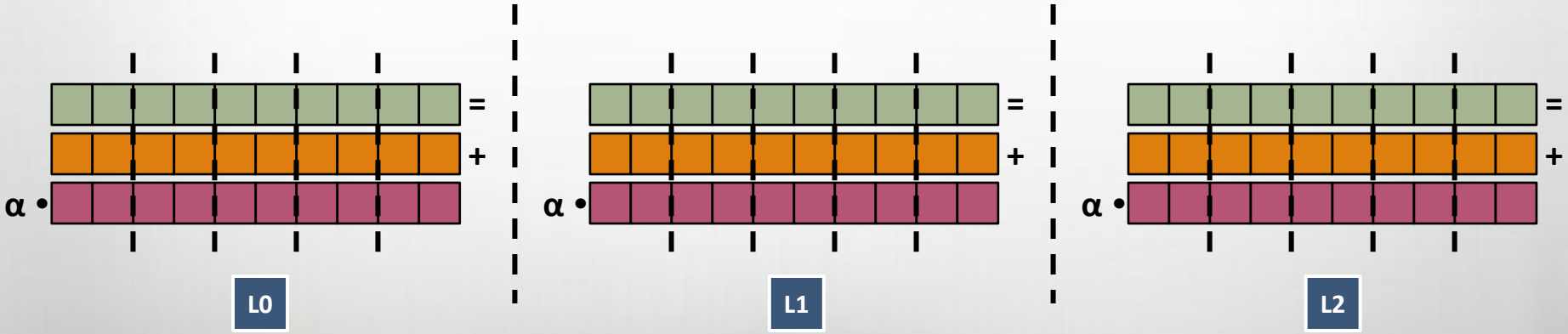
What is a Distribution?

A “recipe” for distributed arrays that...

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



...to a fragmented, per-processor implementation



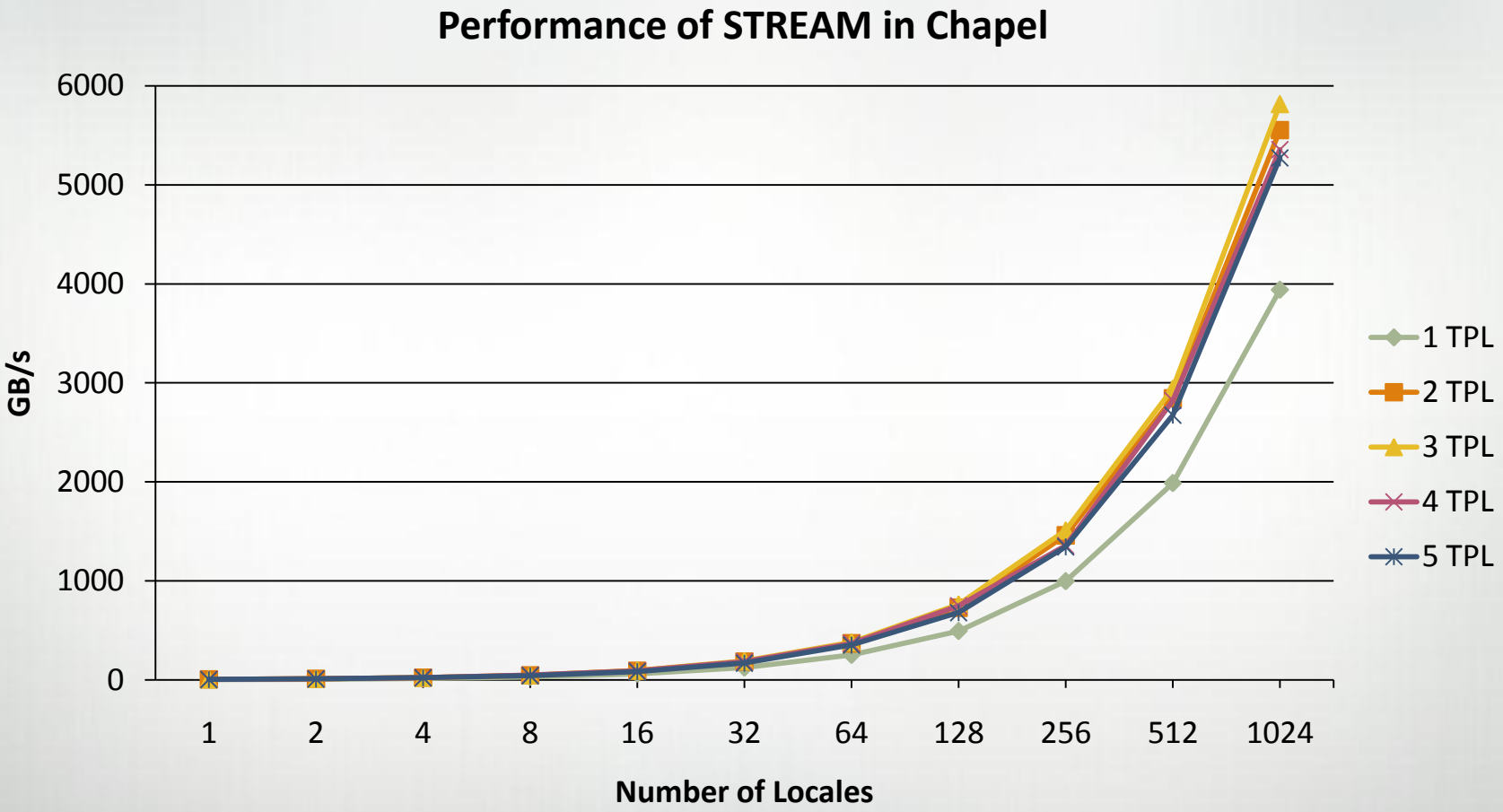
STREAM Triad in Chapel: Multi-Locale

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

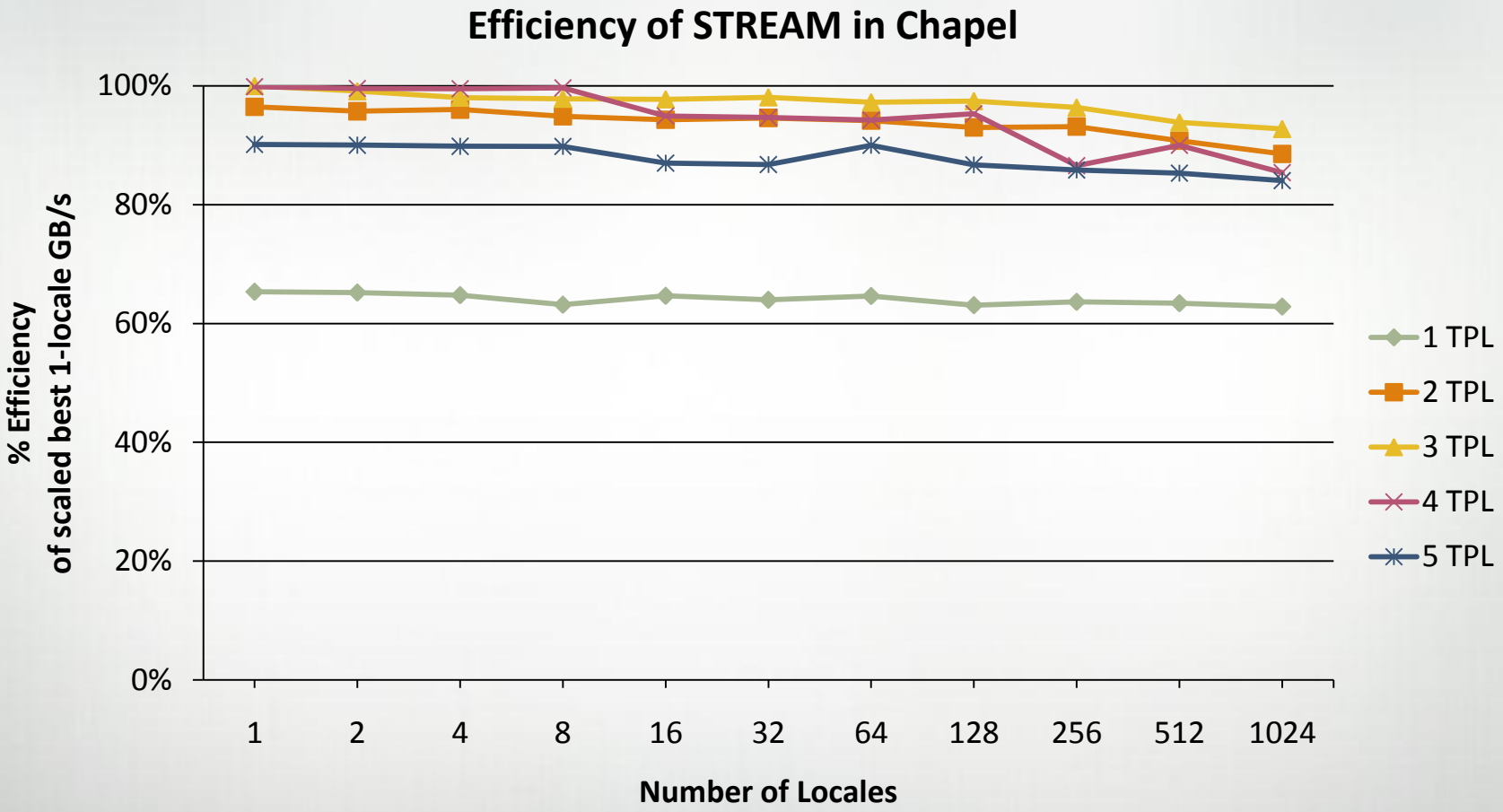
Compute: **forall** i **in** 1..m **do**
A(i) = **B**(i) + **α** * **C**(i);

```
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



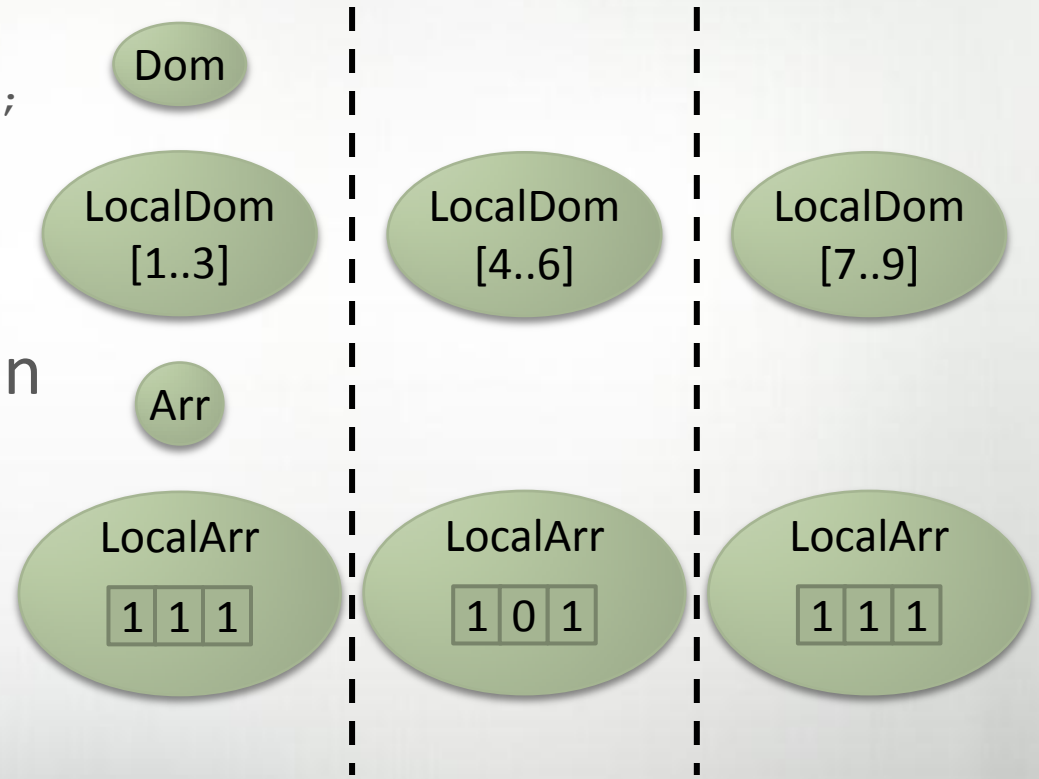
Optimization: Privatization

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;
}
```



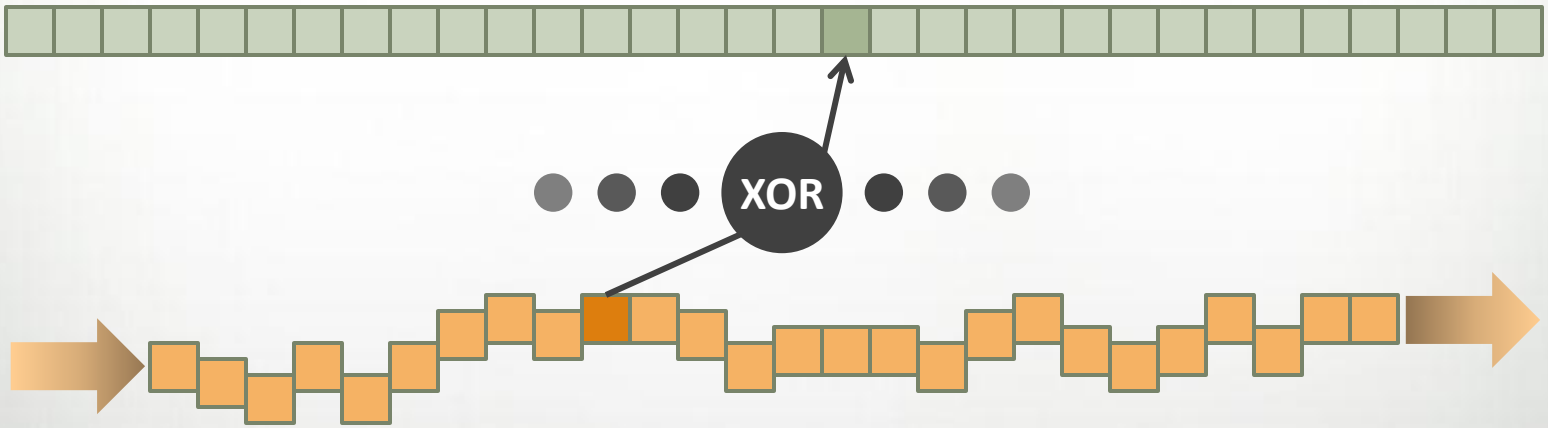
Outline

- 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 $m = 2^n$)

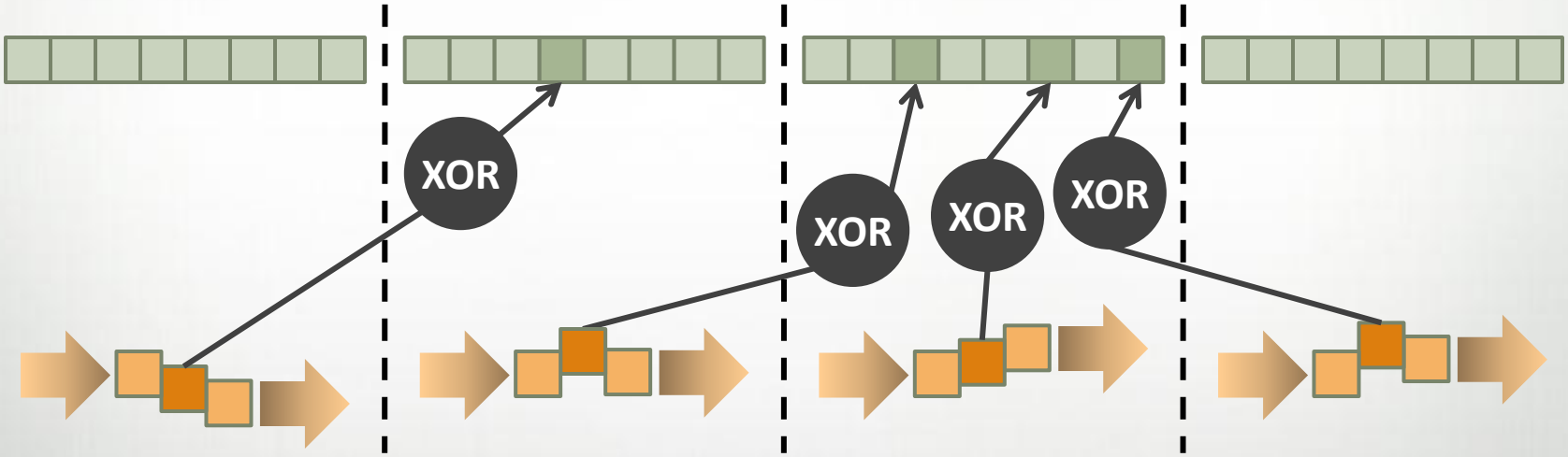
Compute:
```
forall r in RandomUpdates do
  T(r & (m-1)) ^= r;
```



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

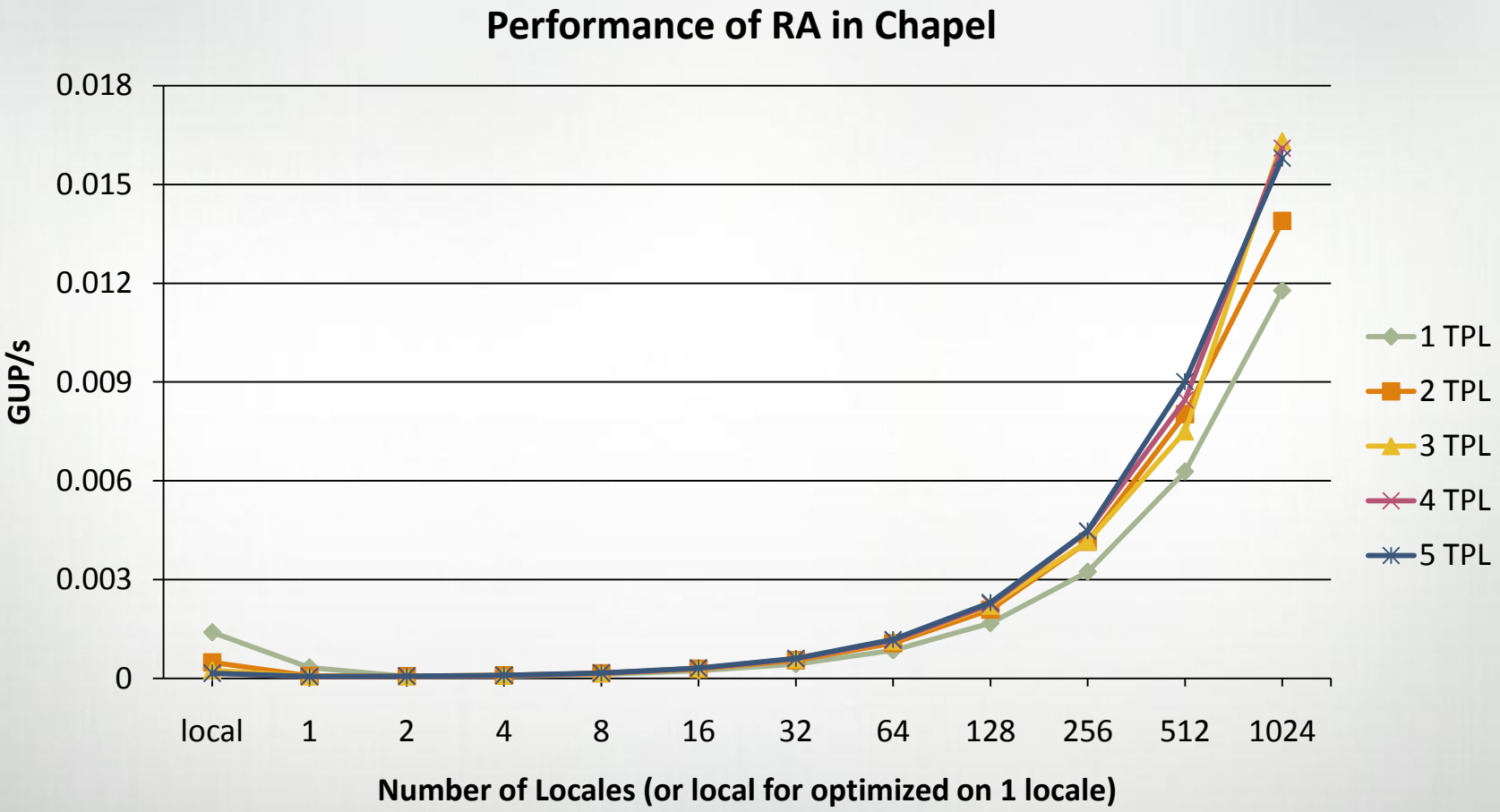
Given: *m*-element table *T* (where $m = 2^n$)

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

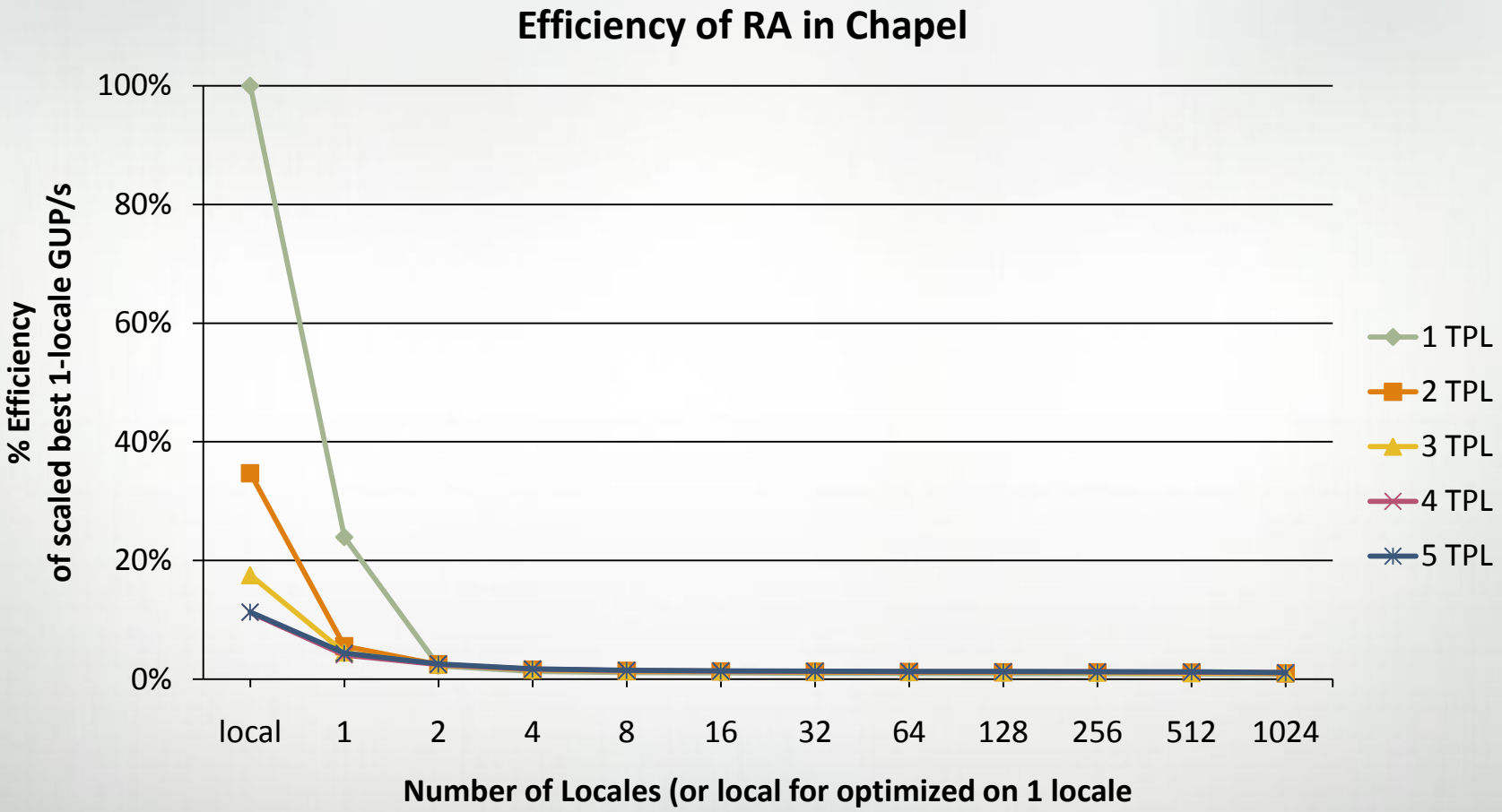
```
config const m = ..., N_U = ..., tpl = ...;
const TableDist = new Block(1,uint(64),[0..m-1],tpl),
      UpdateDist = new Block(1,uint(64),[0..N_U-1],tpl),
      TableSpace: domain(1,uint(64))
                  distributed TableDist = [0..m-1],
      Updates: domain(1,uint(64))
               distributed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

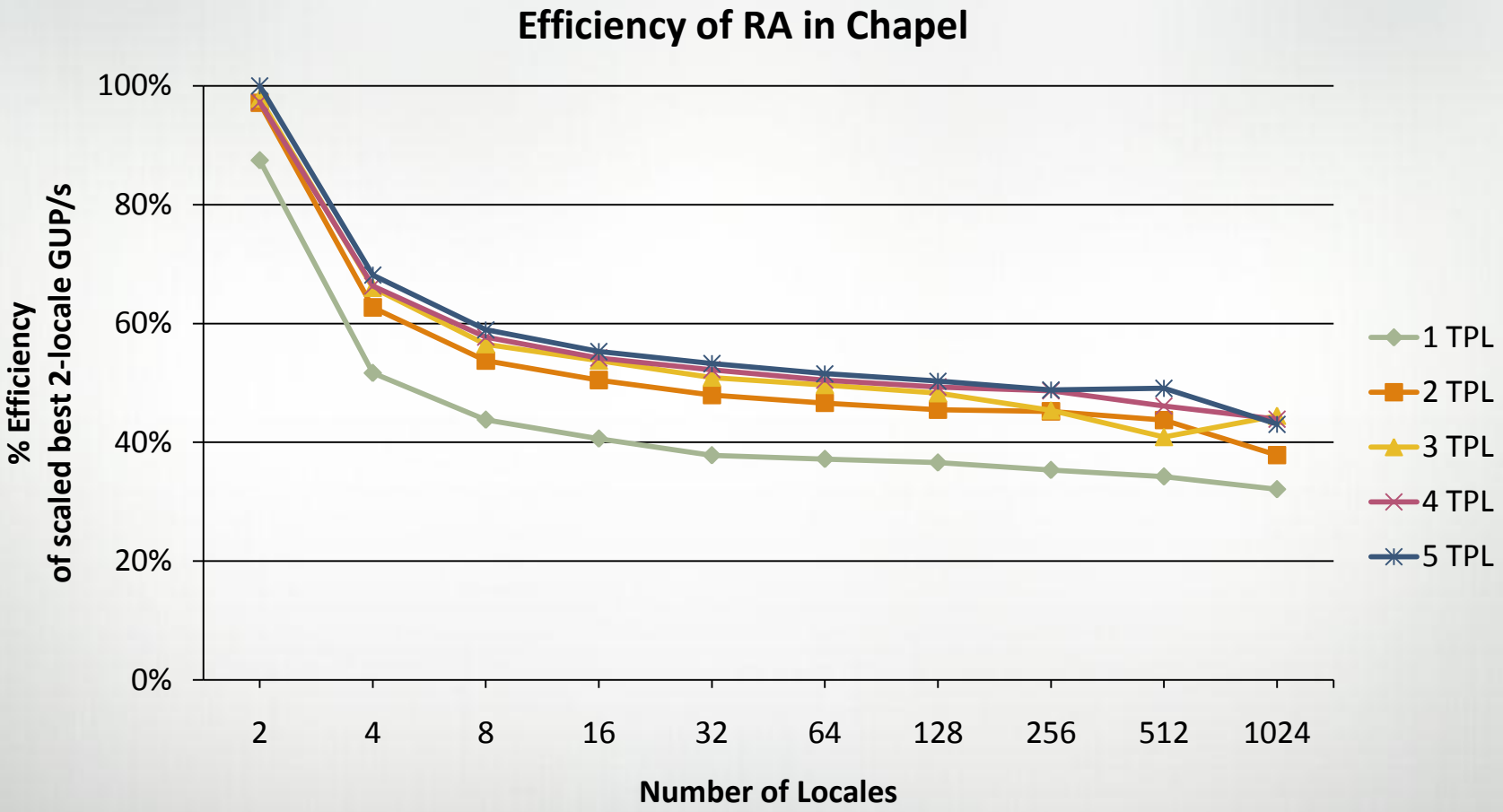
HPCC RA Performance



HPCC RA Efficiency I



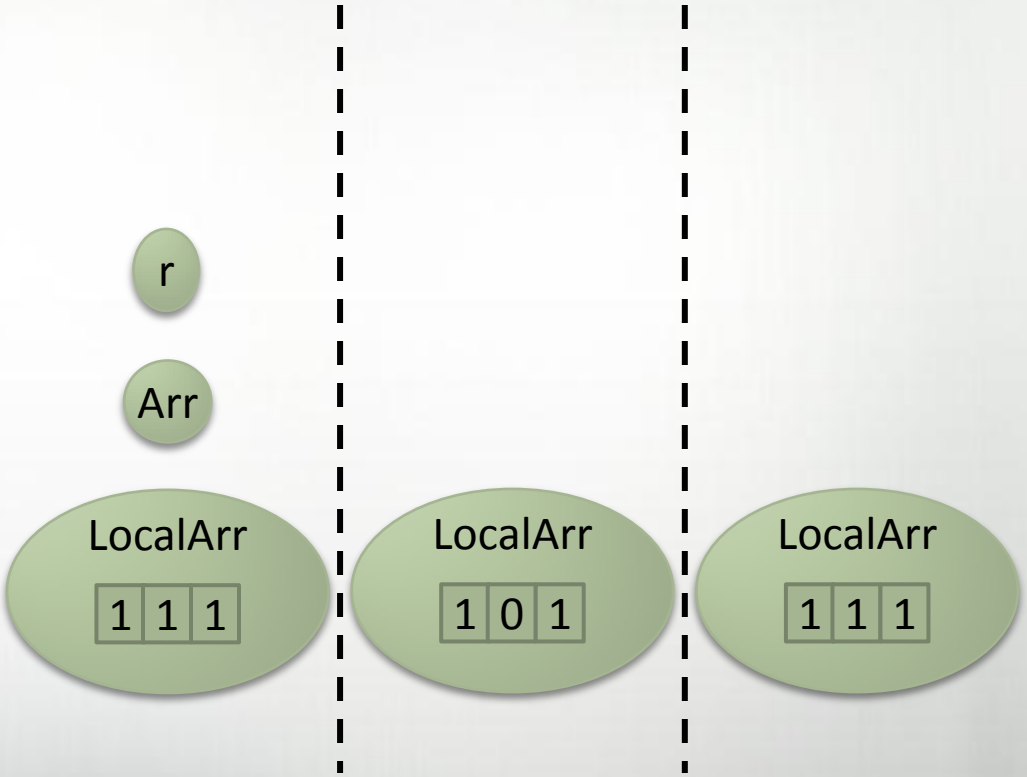
HPCC RA Efficiency II



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

CRAY
THE SUPERCOMPUTER COMPANY

Questions?