Global-View Abstractions for User-Defined Reductions and Scans

PPoPP
March 29, 2006

Steve Deitz
Brad Chamberlain
David Callahan
Larry Snyder

A Bit of Context

- DARPA **HPCS**: **H**igh **P**roductivity **C**omputing **S**ystems
  - HPLS: High Productivity Language Systems
- Increase productivity for HEC community by 2010

Productivity = Programmability +
Performance +
Portability +
Robustness

- Revolutionary results (not evolutionary)
- Marketable to people other than program sponsors
- Phase II competition: Cray, IBM,and Sun

What is Chapel?

- *Chapel:* Cascade High-Productivity Language
- Overall goal:
  - Simplify the creation of parallel programs
  - Support evolution to production-grade codes
  - Emphasize generality
- Motivating Language Technologies:
  1) Global-view multithreaded parallel programming
  2) Locality-aware programming
  3) Object-oriented programming
  4) Generic programming and type inference

A Bit of Background (Reduce)

- Definition

  The **reduce operator** takes a binary operation $\oplus$ and a sequence of values $(/a_1, a_2, \ldots, a_n/)$ and returns the value

$$a_1 \oplus a_2 \oplus \ldots \oplus a_n.$$

- Example

  + reduce (/1, 2, 3, 4, 5, 6, 7, 8, 9, 10/)

  $\rightarrow$ 55

A Bit of Background (Scan)

- Definition

  The **scan operator** takes a binary operation $\oplus$ and a sequence of values $(/a_1, a_2, \ldots, a_n/)$ and returns the sequence of values

$$(/a_1,\ a_1 \oplus a_2,\ \ldots,\ a_1 \oplus a_2 \oplus \ldots \oplus a_n/).$$

- Example

  + scan (/1, 2, 3, 4, 5, 6, 7, 8, 9, 10/)

  → (/1, 3, 6, 10, 15, 21, 28, 36, 45, 55/)

Parallel Performance

- Performance of reduce and scan
  - Highly efficient implementation
    - Log-tree for reduce
    - Parallel-prefix for scan
  - Associative operator → parallel performance
  - Commutative operator → more efficient
- Application of reduce and scan
  - Common in scientific computing
    - Used to test for convergence in iterative algorithms
    - Used in matrix multiplication and sorting kernels
    - ~9% of MPI calls in NPB are reductions (static count)
  - Supported on today's systems

Outline

- Reduce and Scan Overview
- Local-View vs. Global-View Overview
- MPI and Chapel Abstraction for Reduce and Scan
- Application to MPI and Quantitative Results

Local vs. Global View

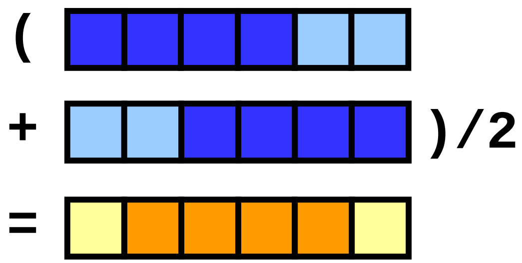
- ***Local-View Programming:***
  - Programmer codes on a per-processor basis
    - Breaks distributed data structures into per-process chunks
    - Breaks work into per-process iterations/control flow

- ***Global-View Programming:***
  - Programmer codes independent of processors (mostly)
    - Relies on compiler to manage decomposition details
    - Provides guide to decomposition at a high level

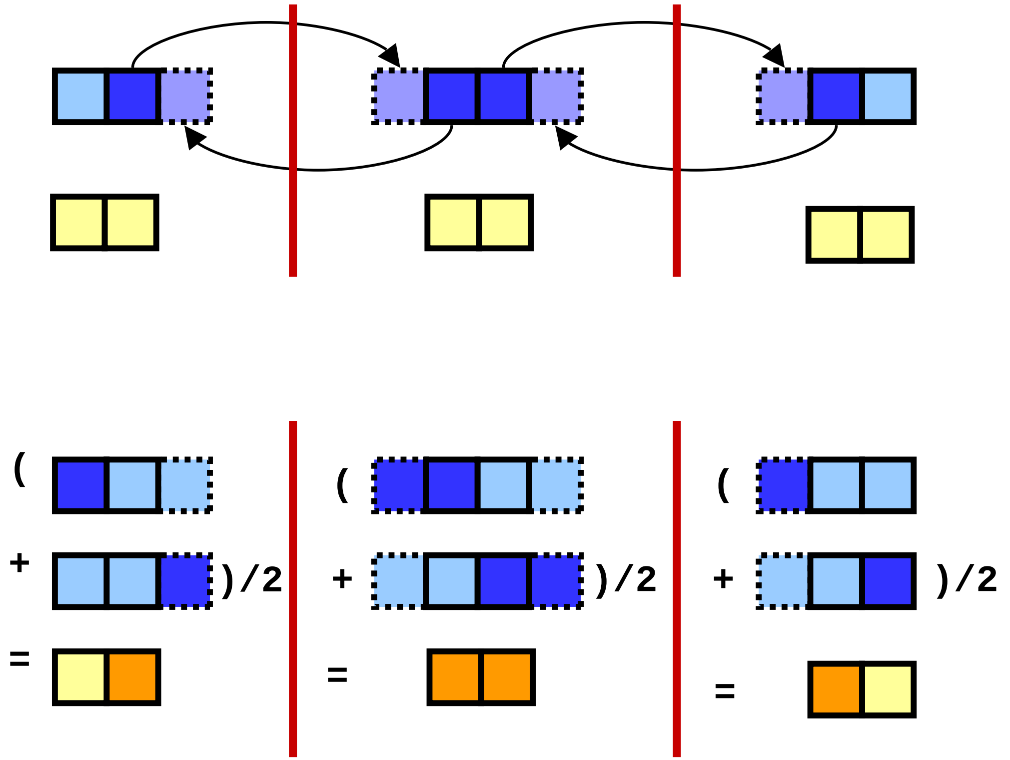
Local vs. Global Example

- **Example:** "Apply a stencil to a vector"

global-view



local-view



Local vs. Global Example

- **Example:** “Apply a stencil to a vector”

global-view

```
var n: int = 1000;
var a, b: [1..n] float;

forall i in 2..n-1 do
  b(i) = (a(i-1) + a(i+1))/2;
```

local-view

```
var n: int = 1000;
var locN: int = n/numProcs;
var a, b: [0..locN+1] float;
var innerLo: int = 1;
var innerHi: int = locN;

if (iHaveRightNeighbor) {
  send(right, a(locN));
  recv(right, a(locN+1));
} else
  innerHi = locN-1;
if (iHaveLeftNeighbor) {
  send(left, a(1));
  recv(left, a(0));
} else
  innerLo = 2;
forall i in innerLo..innerHi do
  b(i) = (a(i-1) + a(i+1))/2;
```

3D Stencil in NAS MG

```
param coeff: domain(1) = [0..3];
param Stencil: domain(3) = [-1..1, -1..1, -1..1];

function rprj3(S, R) {
  param w: [coeff] float
      = (/0.5, 0.25, 0.125, 0.0625/);
  param w3d: [(i,j,k) in Stencil] float
      = w((i!=0) + (j!=0) + (k!=0));
  const SD = S.Domain,
        Rstr = R.stride;

  S = [ijk in SD] sum reduce
        [off in Stencil]
          (w3d(off) * R(ijk + Rstr*off));
}
```

```
subroutine comm3(u,n1,n2,n3,kk)
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'

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
```

Outline

- Reduce and Scan Overview
- Local-View vs. Global-View Overview
- MPI and Chapel Abstraction for Reduce and Scan
- Application to MPI and Quantitative Results

User Defined Reduction MinK

Compute the k minimum values in a sequence of values

E.g. mink(2) reduce (/8, 3, 5, 2, 7, 9, 10, 1, 4, 6/)

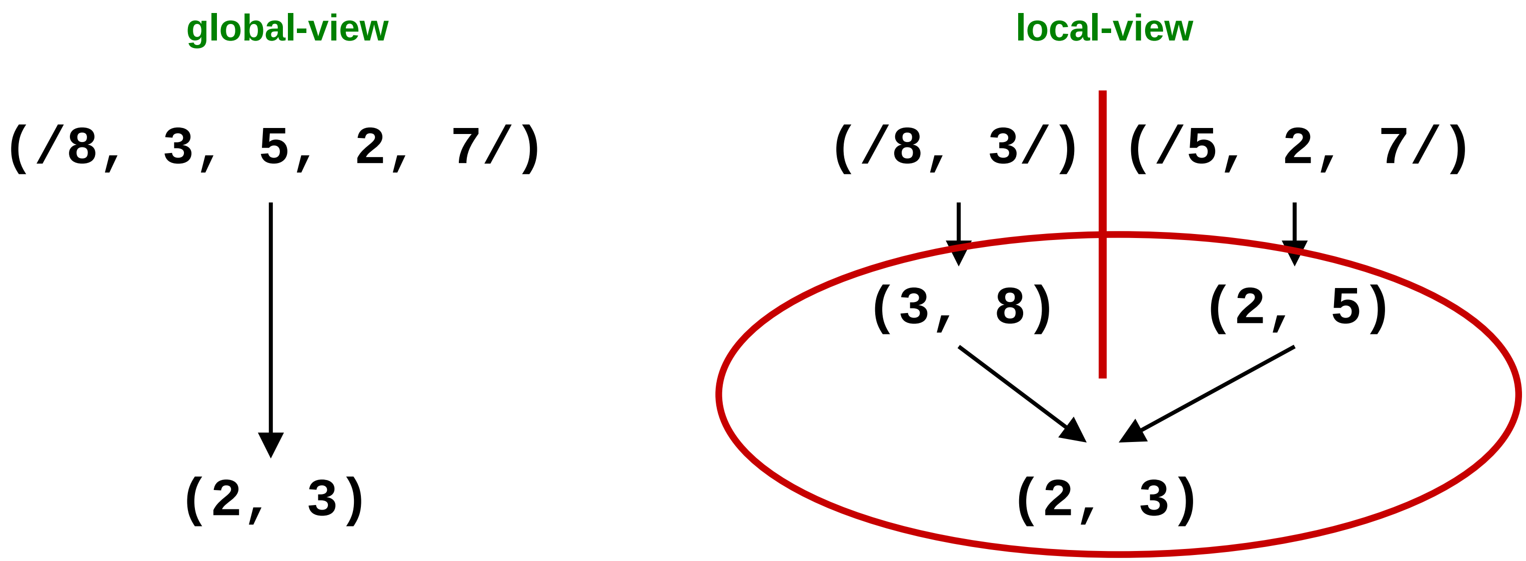
→ (1, 2)

E.g. mink(2) reduce (/Cray, Gonzaga, IBM, Sun, Duke/)

→ (Duke, Gonzaga)

Local vs. Global View

- **Example:** “Reduce a sequence with Min2”



Local-view abstraction applies only to this part of the reduction

MPI Reductions

- Has two varieties
  - Reduce returns reduction on root process
  - Allreduce returns result back to all processes

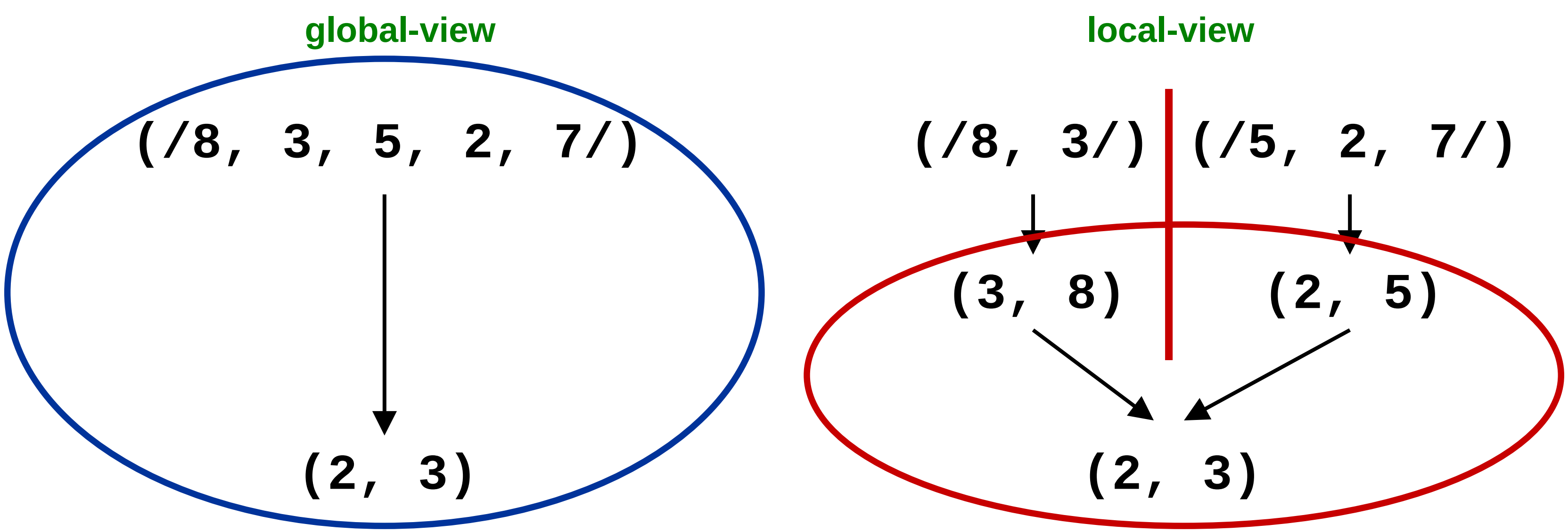
```
int MPI_Reduce(
  void* sendbuf,
  void* recvbuf,
  int count,
  MPI_Datatype datatype,
  MPI_Op op,
  int root,
  MPI_Comm comm)
```

```
int MPI_Allreduce(
  void* sendbuf,
  void* recvbuf,
  int count,
  MPI_Datatype datatype,
  MPI_Op op,
  MPI_Comm comm)
```

- Supports nine built-in reduction operators
- Accepts arbitrary combine function
- MPI_Op assumes associativity, can be commutative

Local vs. Global View

- **Example:** “Reduce a sequence with Min2”



Global-view abstraction needs to hide the details of communication

Local-view abstraction applies only to this part of the reduction

Brief Intro to Chapel Classes

- Similar to classes in other OO languages
  - Fields are accessed via "." syntax
  - Methods are invoked "." syntax
  - Support multiple inheritance, dynamic dispatch
- Automatic default constructor

```
class Point {
  var x : int, y : int;
}
class ColorfulPoint : Point {
  var c : color;
}
var cp = ColorfulPoint(x=2,y=3,c=red);
```

Default constructor provides named argument passing

Brief Intro to Chapel Classes

- Classes can be generic

```
class Triple {
  type elt_type;
  var x, y, z : elt_type;
}
def Triple.add(e : elt_type) {
  x += e; y += e; z += e;
}
var t = Triple(float);
t.add(1.0);
```

Class can be instantiated on different types

Chapel Reduction Class

```
class reduction {
  type elt_type;
  param commutative = true;
  param associative = true;
  def accumulate(x : elt_type);
  def combine(s : reduction);
  def gen();
  def scan_gen(x : elt_type) do
    return gen();
  def red_gen() do
    return gen();
}
```

All reductions are generic to the sequence element type and two parameters that can be overridden

Default scan and reduce generate functions call a common generate function

```
def reduce(r, s) {
  for e in s do
    r.accumulate(e);
  return r.red_gen();
}
```

Serial reduce is implemented as is any overloaded operator

MinK in Chapel

```
class mink : reduction {
  var k : int;
  var v : [1..k] elt_type = max(elt_type);
  def accumulate(x : elt_type) do
    if x < v(1) {
      v(1) = x;
      for i in 2..k do
        if v(i-1) < v(i) then
          (v(i-1),v(i)) = (v(i),(v(i-1));
    }
  def combine(s: mink(elt_type)) do
    for i in 1..k do
      accumulate(s.v(i));
  def gen() do
    return v;
}

... mink(2) reduce (/8, 3, 5, 2, 7/) ...
```

Default constructor stores maxes in k-element array. Operator class stores state.

Combine function takes other thread's locally accumulated state.

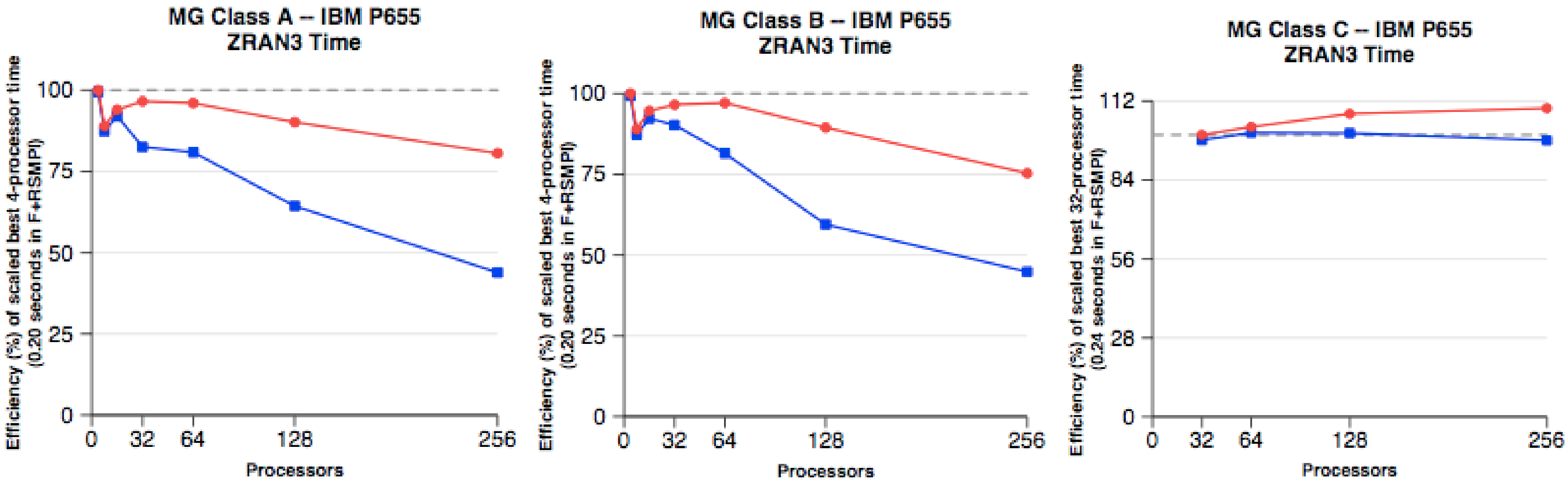
Outline

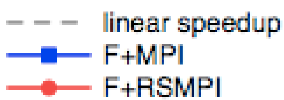
- Reduce and Scan Overview
- Local-View vs. Global-View Overview
- MPI and Chapel Abstraction for Reduce and Scan
- Application to MPI and Quantitative Results

Global-View in MPI

- RSMPI (Reduce and Scan in MPI)
  - Proof of concept of global view
  - Potential use for MPI programmers
- Minimal changes to MPI to support global view
  - Adds a construct to iterate over the local part of an ordered set
  - Uses a preprocessor to generate the accumulate loop based on the iterator construct.
  - Adds an operator as a collection of functions
    - Identity function
    - Accumulate function
    - Combine function
    - Generate function
  - Requires a single RSMPI_Reduce call

MG Initialization Results



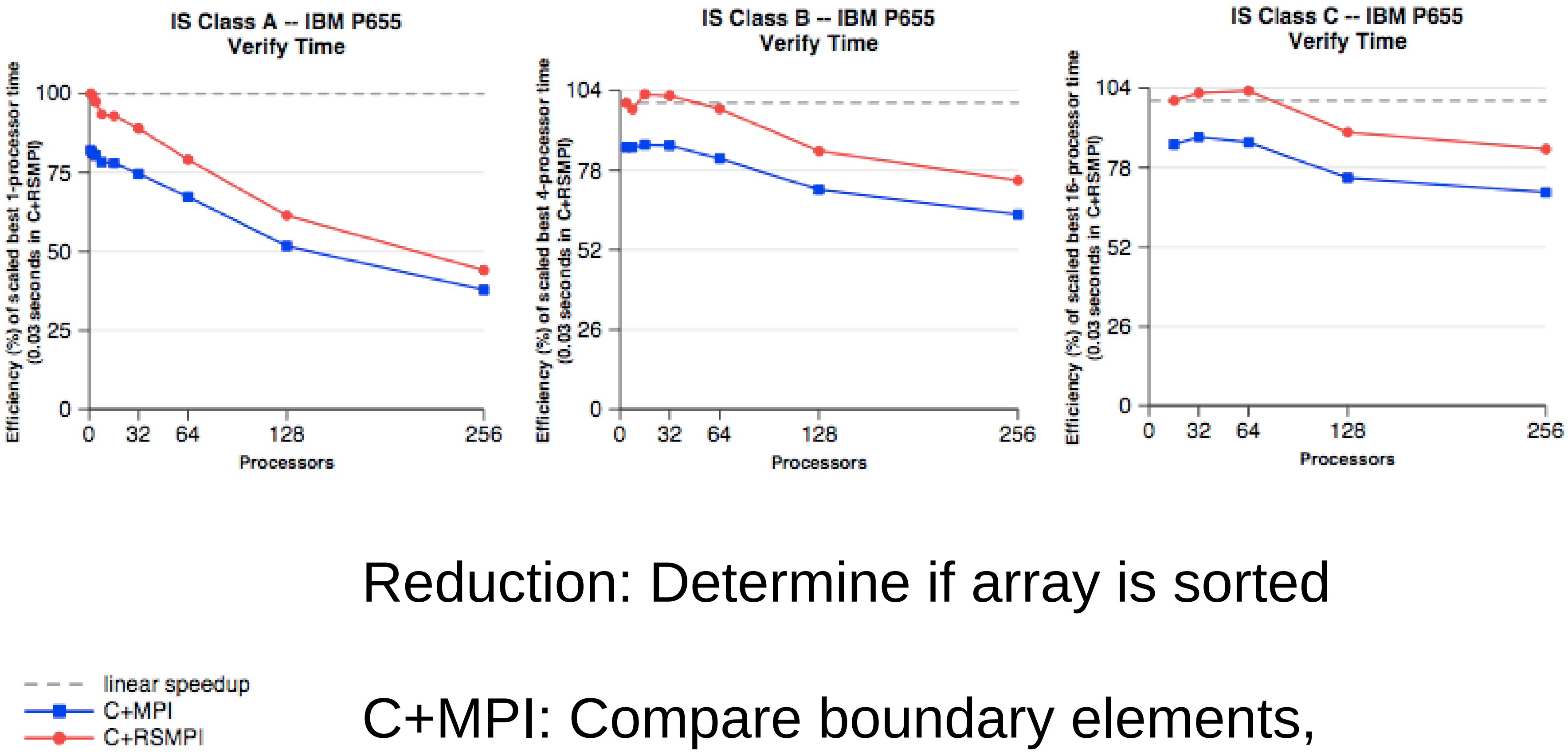


Reduction: Find 10 minimum and 10 maximum values and locations

F+MPI: Uses 10 min and 10 max reductions

F+RSMPI: Uses one big reduction

IS Verification Results



Reduction: Determine if array is sorted

C+MPI: Compare boundary elements, determine if local portion is sorted, and 'sum' reduce local determinations

C+RSMPI: Use one big reduction

Summary

- High-level Chapel abstraction
  - Takes advantage of generic OOP capabilities
  - Contains both accumulate and combine function
  - Uses default constructor to produce identity
  - Supports generate function to return result that is different from the combining state
  - Supports a scan generate function that can return different result from a reduction
- RSMPI
  - Shows performance does not suffer at global view
  - Suggests path for incremental improvements
- Global-View
  - Orthogonal to changes in the distribution/layout

Questions?

For more information:

http://chapel.cs.washington.edu

email: chapel_info@cray.com