Chapel: High-Productivity Parallel Computing

Brad Chamberlain

Chapel Team

Google

September 19, 2007

Chapel

Chapel: a new parallel language being developed by Cray Inc.

Themes:

- general parallelism
  - data-, task-, nested parallelism using *global-view* abstractions
  - general parallel architectures
- locality control
  - data distribution
  - task placement (typically data-driven)
- reduce gap between mainstream and parallel languages
  - object-oriented programming (OOP)
  - type inference and generic programming

Chapel's Setting: HPCS

HPCS: High *Productivity* Computing Systems (DARPA *et al.*)

- Goal: Raise HEC user productivity by 10× for the year 2010
- Productivity = Performance
  + Programmability
  + Portability
  + Robustness

- **Phase II**: Cray, IBM, Sun (July 2003 – June 2006)
  - Evaluated the entire system architecture's impact on productivity…
    - processors, memory, network, I/O, OS, runtime, compilers, tools, …
    - …and new languages:

      Cray: Chapel IBM: X10 Sun: Fortress

- **Phase III**: Cray, IBM (July 2006 – 2010)
  - Implement the systems and technologies resulting from phase II
  - (Sun also continues work on Fortress, without HPCS funding)

Chapel and Productivity

Chapel's Productivity Goals:

- vastly improve programmability over current languages/models
  - writing parallel codes
  - reading, modifying, porting, tuning, maintaining them
- support performance at least as good as MPI
  - competitive with MPI on generic clusters
  - better than MPI on more capable architectures
- improve portability compared to current languages/models
  - as ubiquitous as MPI, but with fewer architectural assumptions
  - more portable than OpenMP, UPC, CAF, …
- improve code robustness via improved semantics and concepts
  - eliminate common error cases altogether
  - better abstractions to help avoid other errors

Outline

- ✓ Introduction to Chapel
- ➢ Global-View Programming
- ❑ Data Parallel Examples: the Stencil Ramp
- ❑ Task Parallel Features & Examples
- ❑ Status & Summary

Parallel Programming Model Taxonomy

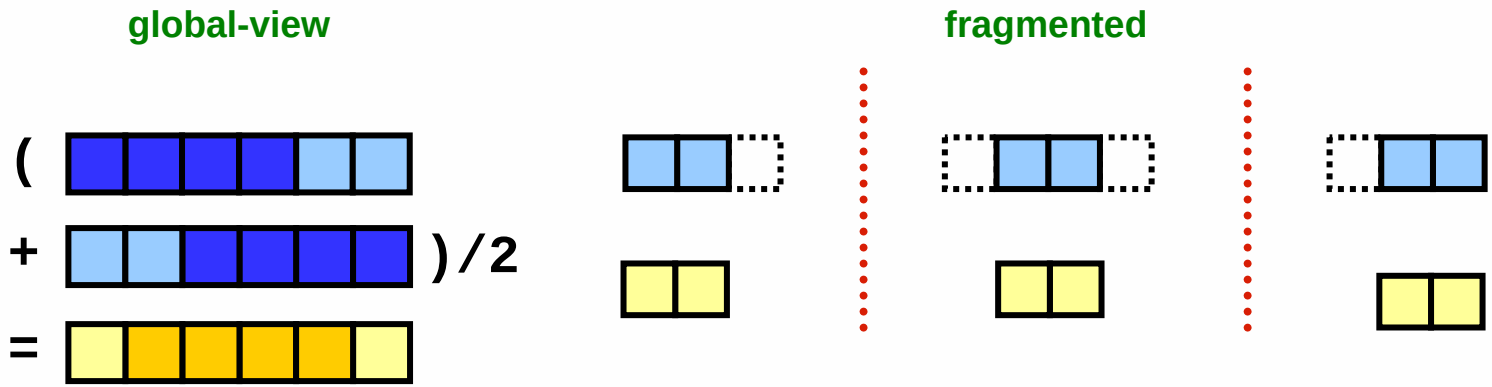
programming model: the mental model a programmer uses when coding using a language, library, or other notation

fragmented models: those in which the programmer writes code from the point-of-view of a single processor/thread

global-view models: those in which the programmer can write code that describes the computation as a whole

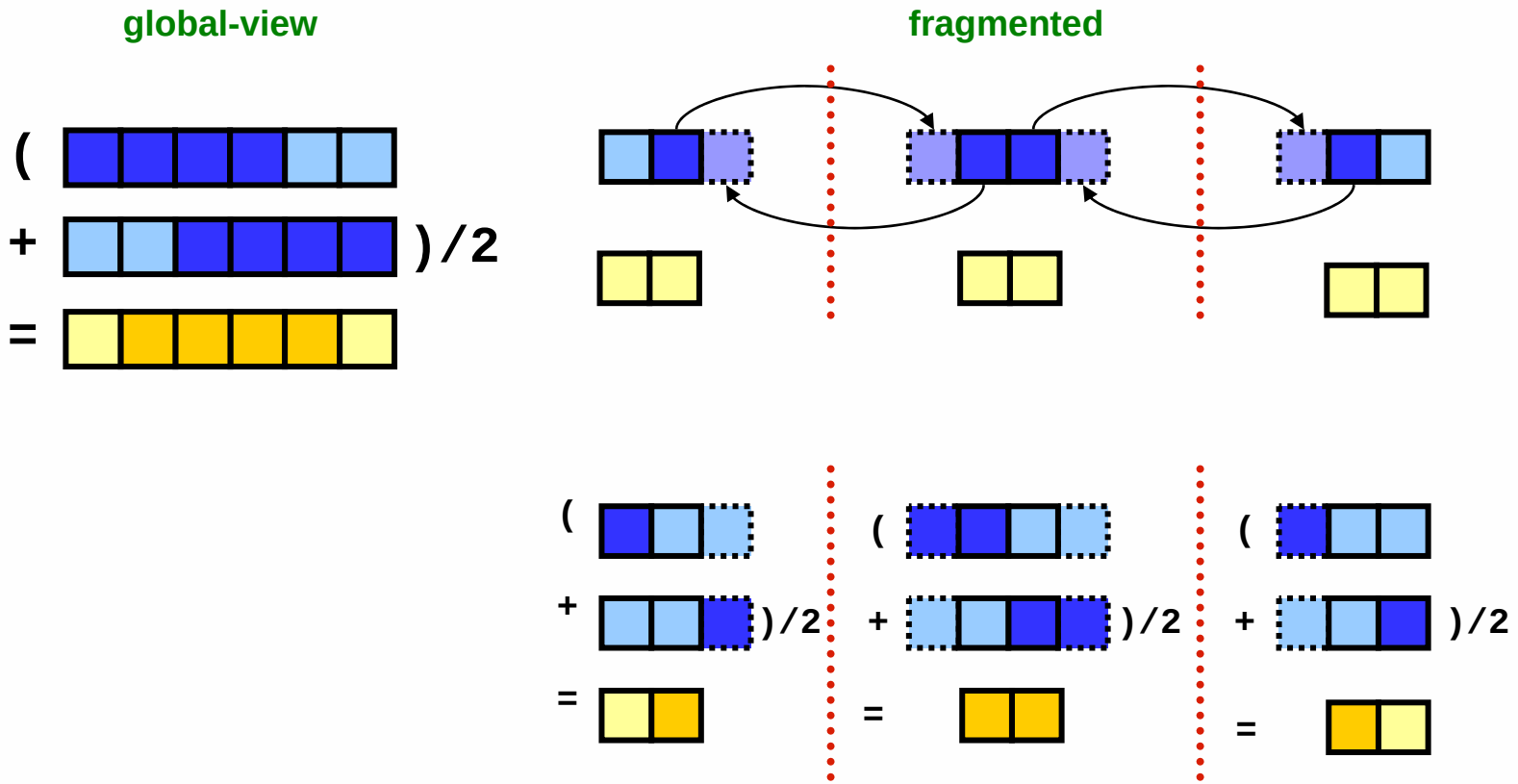
Global-view vs. Fragmented

- **Problem:** "Apply 3-pt stencil to vector"



Global-view vs. Fragmented

- **Problem:** "Apply 3-pt stencil to vector"



Parallel Programming Model Taxonomy

programming model: the mental model a programmer uses when coding using a language, library, or other notation

fragmented models: those in which the programmer writes code from the point-of-view of a single processor/thread

SPMD models: Single-Program, Multiple Data -- a common fragmented model in which the user writes one program & runs multiple copies of it, parameterized by a unique ID

global-view models: those in which the programmer can write code that describes the computation as a whole

Global-view vs. SPMD Code

- **Problem:** "Apply 3-pt stencil to vector"

global-view

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

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

SPMD

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

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

Global-view vs. SPMD Code

- **Problem:** "Apply 3-pt stencil to vector"

global-view

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

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

Assumes *numProcs* divides *n*; a more general version would require additional effort

SPMD

```
var n: int = 1000;
var locN: int = n/numProcs;
var a, b: [0..locN+1] real;
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 {
  b(i) = (a(i-1) + a(i+1))/2;
}
```

Current HPC Programming Notations

- **communication libraries:**
  - MPI, MPI-2 (fragmented, typically SPMD)
  - SHMEM, ARMCI, GASNet (SPMD)
- **shared memory models:**
  - OpenMP (global-view, trivially)
- **PGAS languages:**
  - Co-Array Fortran (SPMD)
  - UPC (SPMD)
  - Titanium (SPMD)

MPI SPMD pseudo-code

- **Problem:** "Apply 3-pt stencil to vector"

SPMD (pseudocode + MPI)

```
var n: int = 1000, locN: int = n/numProcs;
var a, b: [0..locN+1] real;
var innerLo: int = 1, innerHi: int = locN;
var numProcs, myPE: int;
var retval: int;
var status: MPI_Status;

MPI_Comm_size(MPI_COMM_WORLD, &numProcs);
MPI_Comm_rank(MPI_COMM_WORLD, &myPE);
if (myPE < numProcs-1) {
  retval = MPI_Send(&(a(locN)), 1, MPI_FLOAT, myPE+1, 0, MPI_COMM_WORLD);
  if (retval != MPI_SUCCESS) { handleError(retval); }
  retval = MPI_Recv(&(a(locN+1)), 1, MPI_FLOAT, myPE+1, 1, MPI_COMM_WORLD, &status);
  if (retval != MPI_SUCCESS) { handleErrorWithStatus(retval, status); }
} else
  innerHi = locN-1;
if (myPE > 0) {
  retval = MPI_Send(&(a(1)), 1, MPI_FLOAT, myPE-1, 1, MPI_COMM_WORLD);
  if (retval != MPI_SUCCESS) { handleError(retval); }
  retval = MPI_Recv(&(a(0)), 1, MPI_FLOAT, myPE-1, 0, MPI_COMM_WORLD, &status);
  if (retval != MPI_SUCCESS) { handleErrorWithStatus(retval, status); }
} else
  innerLo = 2;
forall i in (innerLo..innerHi) {
  b(i) = (a(i-1) + a(i+1))/2;
}
```

Communication becomes geometrically more complex for higher-dimensional arrays

Fortran+MPI 3D 27-point stencil (NAS MG *rprj3*)

```
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  i2=1,n2
      do  i1=1,n1
         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
  j = k-1
  call comm3(s,m1j,m2j,m3j,j)
  return
  end
```

Summarizing Fragmented/SPMD Models

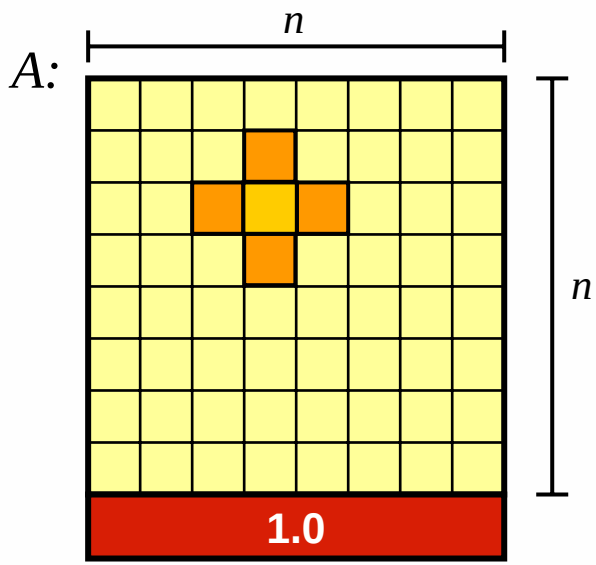
- Advantages:
  - fairly straightforward model of execution
  - relatively easy to comprehend, learn, reason about
  - relatively easy to implement
  - reasonable performance on commodity architectures
  - portable/ubiquitous
  - lots of important scientific work has been accomplished using them

- Disadvantages:
  - blunt means of expressing parallelism: cooperating executables
  - fails to abstract away architecture / implementing mechanisms
  - obfuscates algorithms with many low-level details
    - error-prone
    - brittle code: difficult to read, maintain, modify, *experiment*
    - "MPI: the assembly language of parallel computing"

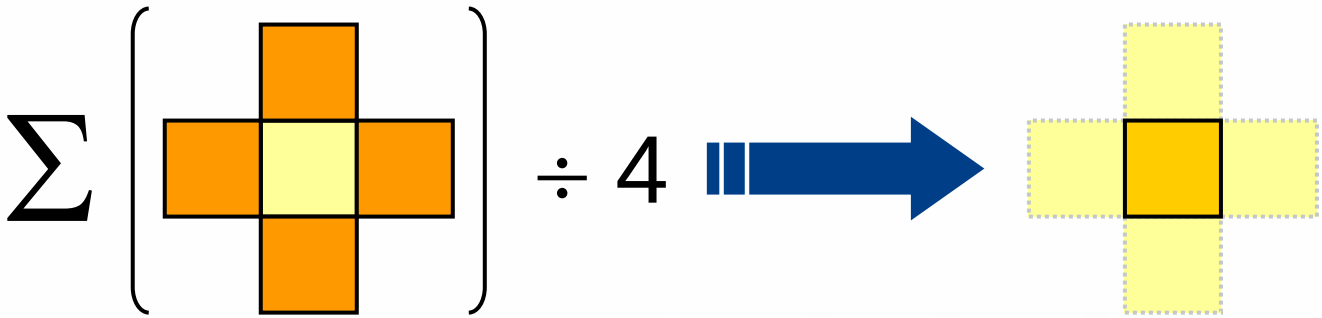
Outline

- ✓ Introduction to Chapel
- ✓ Global-View Programming
- ➢ Data Parallel Examples: the Stencil Ramp
- ❑ Task Parallel Features & Examples
- ❑ Status & Summary

Example 1: Jacobi Iteration



repeat until max change < ε



Jacobi Iteration in Chapel

```
config const n = 6,
             epsilon = 1.0e-5;

const BigD: domain(2) = [0..n+1, 0..n+1],
          D: subdomain(BigD) = [1..n, 1..n],
   LastRow: subdomain(BigD) = D.exterior(1,0);

var A, Temp : [BigD] real;

A[LastRow] = 1.0;

do {
  [(i,j) in D] Temp(i,j) = (A(i-1,j) + A(i+1,j)
                            + A(i,j-1) + A(i,j+1)) / 4.0;

  var delta = max reduce abs(A(D) - Temp(D));
  A(D) = Temp(D);
} while (delta > epsilon);

writeln(A);
```

Jacobi Iteration in Chapel

```
config const n = 6,
             epsilon = 1.0e-5;

const BigD: domain(2) = [0..n+1, 0..n+1],
          D: subdomain(BigD) = [1..n, 1..n],
   LastRow: subdomain(BigD) = D.exterior(1,0);

var A, Temp : [BigD] real;

A[Las

do {
  [(i

  var
  A(D
} whi

writeln(A);
```

Declare program parameters

const ⇒ can't change values after initialization

config ⇒ can be set on executable command-line

```
prompt> jacobi --n=10000 --epsilon=0.0001
```

note that no types are given; inferred from initializer

n ⇒ **integer** (current default, 32 bits)

epsilon ⇒ **floating-point** (current default, 64 bits)

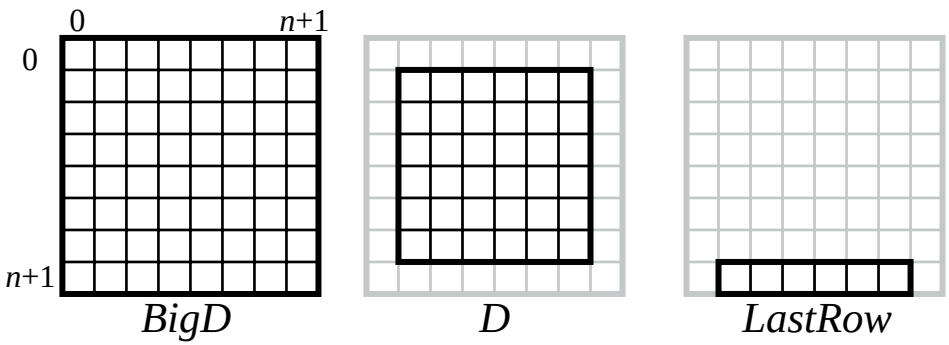
Jacobi Iteration in Chapel

```
config const n = 6,
             epsilon = 1.0e-5;

const BigD: domain(2) = [0..n+1, 0..n+1],
          D: subdomain(BigD) = [1..n, 1..n],
    LastRow: subdomain(BigD) = D.exterior(1,0);
```

Declare domains (first class index sets)

domain(2) ⇒ 2D arithmetic domain, indices are integer 2-tuples

subdomain(*P*) ⇒ a domain of the same type as *P* whose indices are guaranteed to be a subset of *P*'s



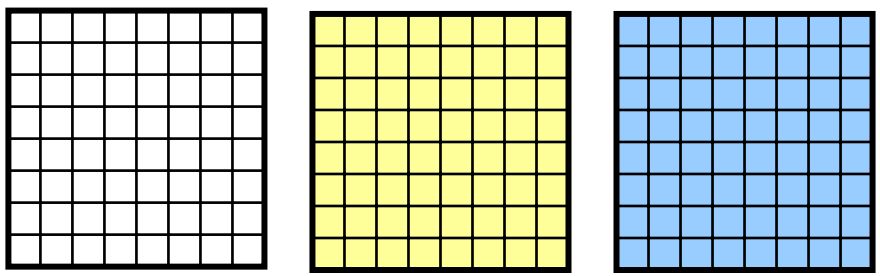
exterior ⇒ one of several built-in domain generators

Jacobi Iteration in Chapel

```
config const n = 6,
             epsilon = 1.0e-5;

const BigD: domain(2) = [0..n+1, 0..n+1],
          D: subdomain(BigD) = [1..n, 1..n],
    LastRow: subdomain(BigD) = D.exterior(1,0);

var A, Temp : [BigD] real;
```

Declare arrays

var ⇒ can be modified throughout its lifetime
: *T* ⇒ declares variable to be of type *T*
: *[D] T* ⇒ array of size *D* with elements of type *T*
(no initializer) ⇒ values initialized to default value (0.0 for reals)

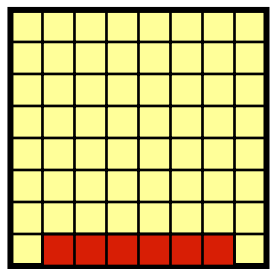


Jacobi Iteration in Chapel

```
config const n = 6,
             epsilon = 1.0e-5;

const BigD: domain(2) = [0..n+1, 0..n+1],
          D: subdomain(BigD) = [1..n, 1..n],
   LastRow: subdomain(BigD) = D.exterior(1,0);

var A, Temp : [BigD] real;

A[LastRow] = 1.0;
```

Set Explicit Boundary Condition

indexing by domain ⇒ slicing mechanism
array expressions ⇒ parallel evaluation



A

Jacobi Iteration in Chapel

Compute 5-point stencil

[(*i,j*) in *D*] ⇒ parallel forall expression over *D*'s indices, binding them to new variables *i* and *j*

Note: since $(i,j) \in D$ and $D \subseteq BigD$ and *Temp*: [*BigD*]
⇒ no bounds check required for *Temp(i,j)*
with compiler analysis, same can be proven for A's accesses

```
  [(i,j) in D] Temp(i,j) = (A(i-1,j) + A(i+1,j)
                          + A(i,j-1) + A(i,j+1)) / 4.0;

  var delta = max reduce abs(A(D) - Temp(D));
  A(D) = Temp(D);
} while (delta > epsilon);

writeln(A);
```

Jacobi Iteration in Chapel

```
config const n = 6,
             epsilon = 1.0e-5;

const BigD: domain(2) = [0..n+1, 0..n+1],
```

Compute maximum change

***op* reduce** ⇒ collapse aggregate expression to scalar using *op*

Promotion: *abs()* and – are scalar operators, automatically promoted to work with array operands

```
do {
  [(i,j) in D] Temp(i,j) = (A(i-1,j) + A(i+1,j)
                          + A(i,j-1) + A(i,j+1)) / 4.0;

  var delta = max reduce abs(A(D) - Temp(D));
  A(D) = Temp(D);
} while (delta > epsilon);

writeln(A);
```

Jacobi Iteration in Chapel

```
config const n = 6,
             epsilon = 1.0e-5;

const BigD: domain(2) = [0..n+1, 0..n+1],
          D: subdomain(BigD) = [1..n, 1..n],
   LastRow: subdomain(BigD) = D.exterior(1,0);

var

A[La

do {
  [(i,j) in D] Temp(i,j) = (A(i-1,j) + A(i+1,j)
                          + A(i,j-1) + A(i,j+1)) / 4.0;

  var delta = max reduce abs(A(D) - Temp(D));
  A(D) = Temp(D);
} while (delta > epsilon);

writeln(A);
```

Copy data back & Repeat until done

uses slicing and whole array assignment
standard *do…while* loop construct

Jacobi Iteration in Chapel

```
config const n = 6,
             epsilon = 1.0e-5;

const BigD: domain(2) = [0..n+1, 0..n+1],
          D: subdomain(BigD) = [1..n, 1..n],
    LastRow: subdomain(BigD) = D.exterior(1,0);

var A, Temp : [BigD] real;

A[LastRow] = 1.0;

do {
  [(
                                                  j)
                                                 1)) / 4.0;

  var delta = max reduce abs(A(D) - Temp(D));
  A(D) = Temp(D);
} while (delta > epsilon);

writeln(A);
```

Write array to console

If written to a file, parallel I/O would be used

Jacobi Iteration in Chapel

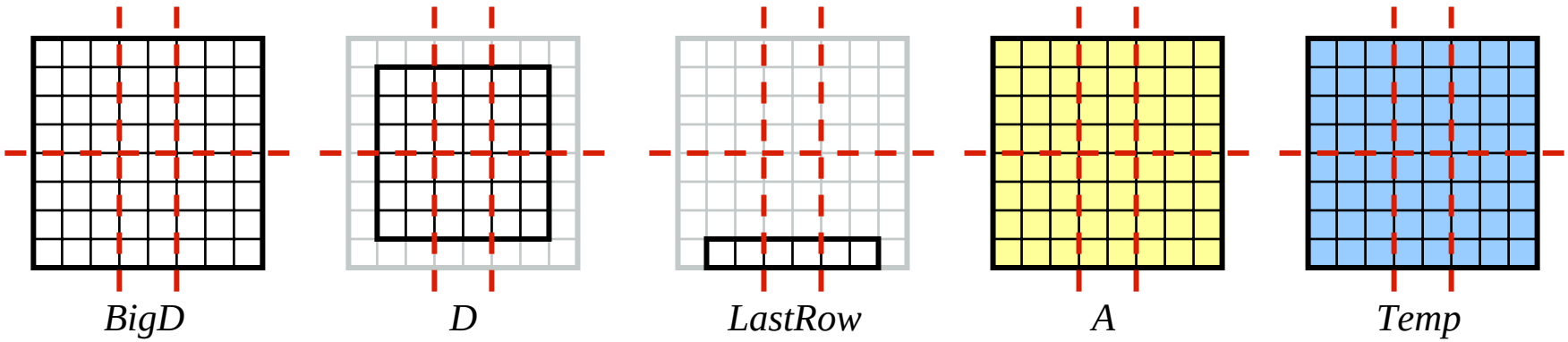
```
config const n = 6,
             epsilon = 1.0e-5;

const BigD: domain(2) = [0..n+1, 0..n+1] distributed (Block),
          D: subdomain(BigD) = [1..n, 1..n],
   LastRow: subdomain(BigD) = D.exterior(1,0);

var A, Temp : [BigD] real;
```

With this change, same code runs in a distributed manner

Domain distribution maps indices to *locales*

⇒ decomposition of arrays & default location of iterations over locales

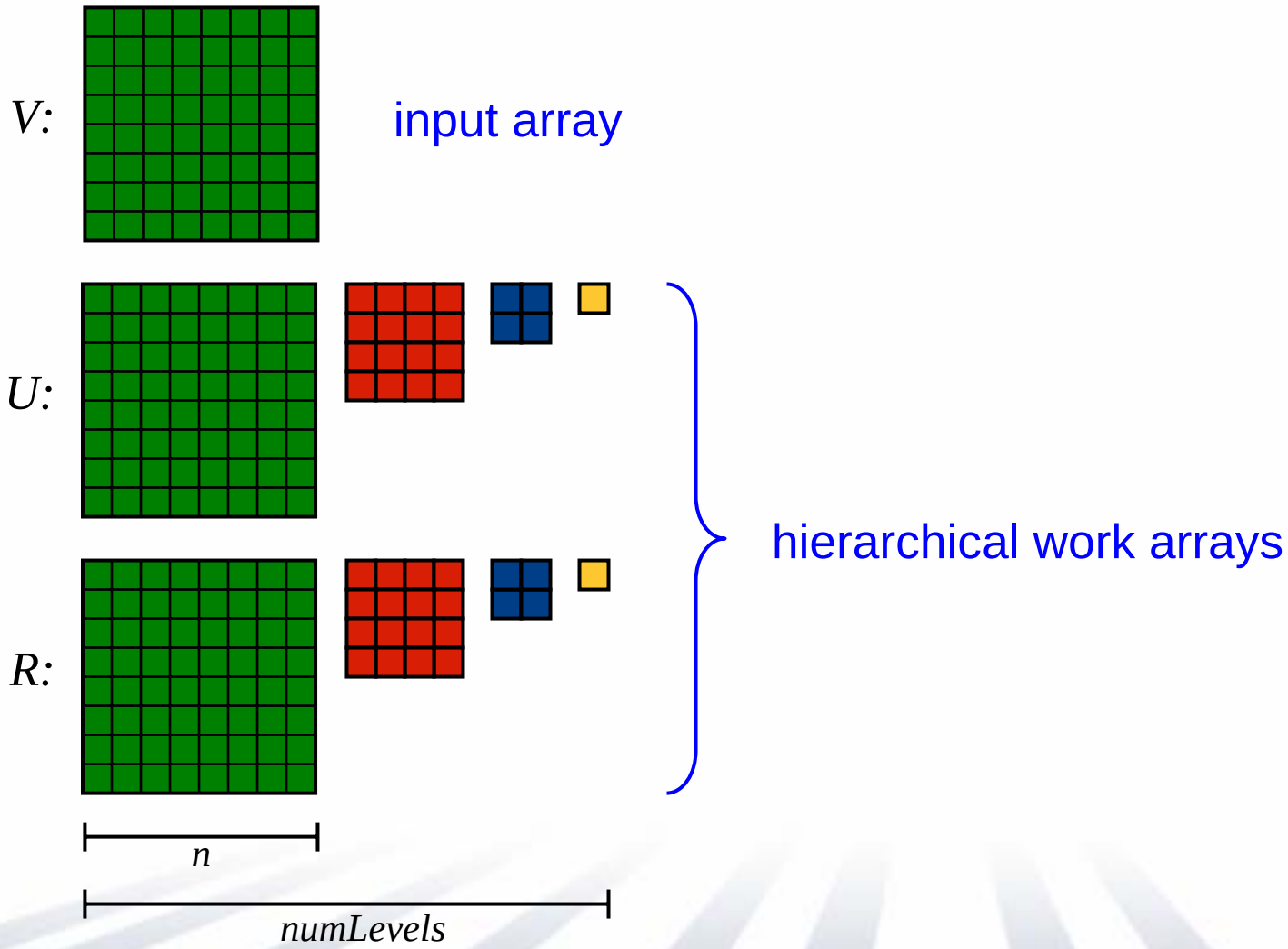
Subdomains inherit parent domain's distribution



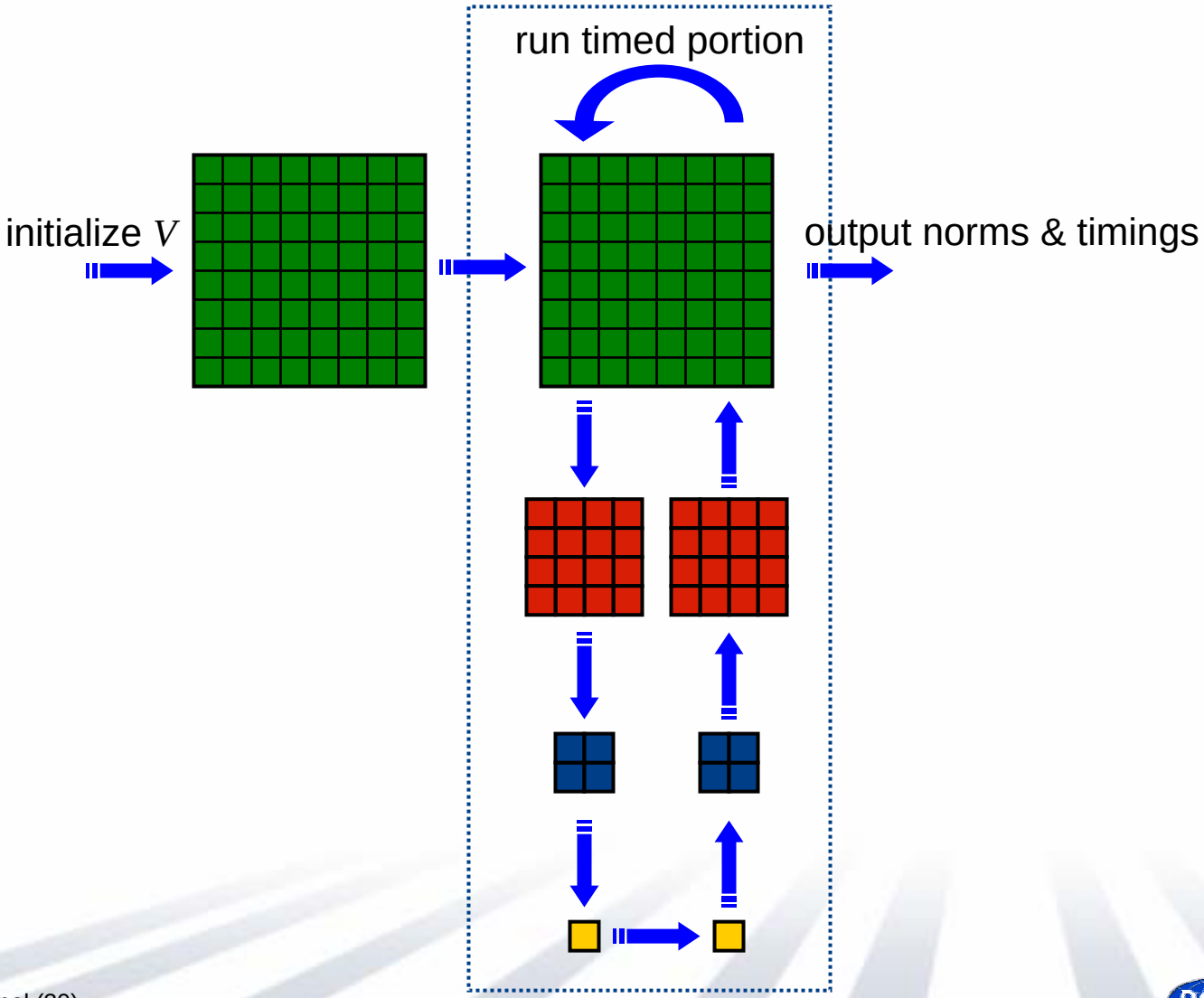
Jacobi Iteration in Chapel

```
config const n = 6,
             epsilon = 1.0e-5;

const BigD: domain(2) = [0..n+1, 0..n+1] distributed (Block),
          D: subdomain(BigD) = [1..n, 1..n],
   LastRow: subdomain(BigD) = D.exterior(1,0);

var A, Temp : [BigD] real;

A[LastRow] = 1.0;

do {
  [(i,j) in D] Temp(i,j) = (A(i-1,j) + A(i+1,j)
                            + A(i,j-1) + A(i,j+1)) / 4.0;

  var delta = max reduce abs(A(D) - Temp(D));
  [ij in D] A(ij) = Temp(ij);
} while (delta > epsilon);

writeln(A);
```

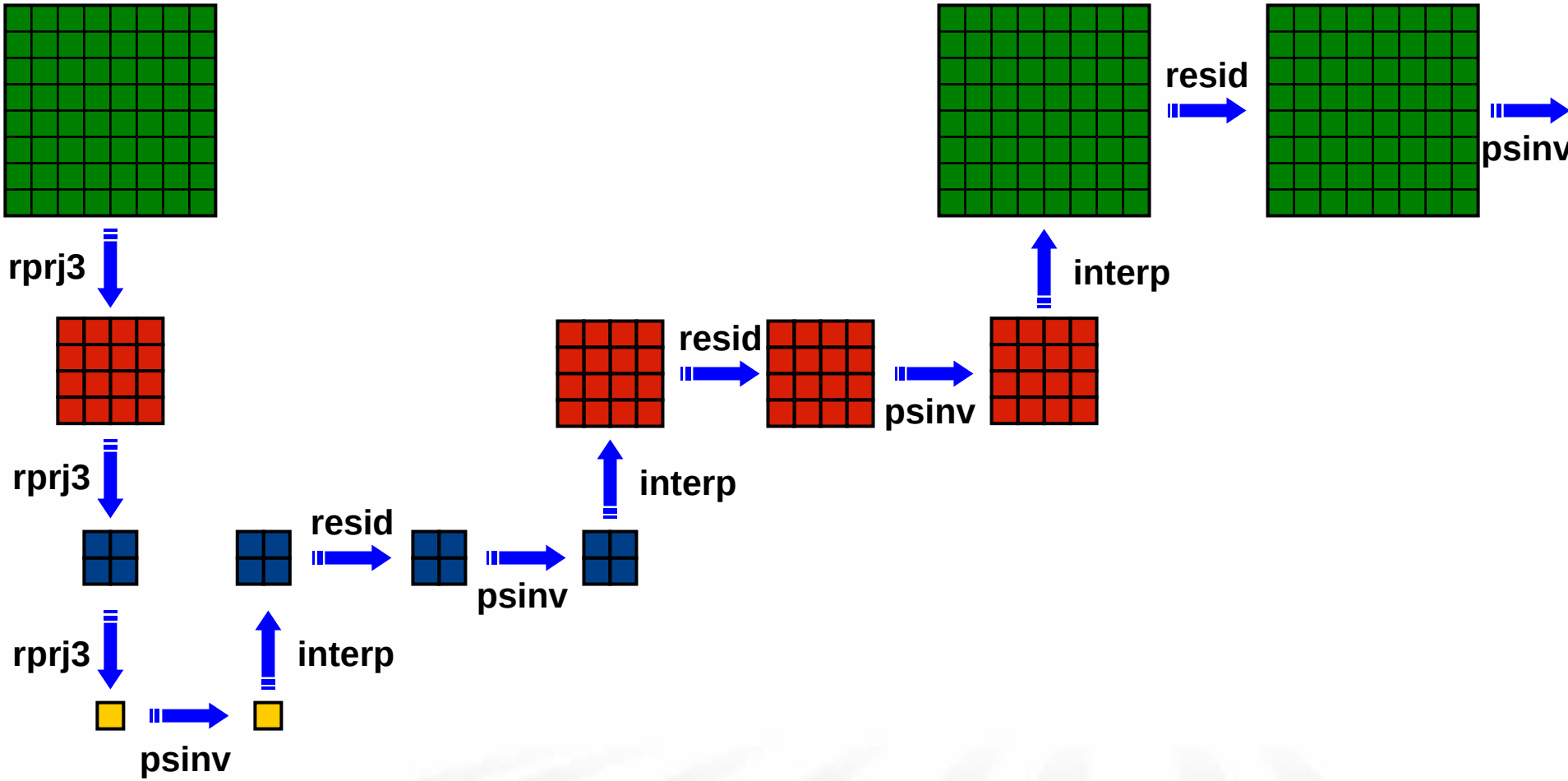
Example 2: Multigrid



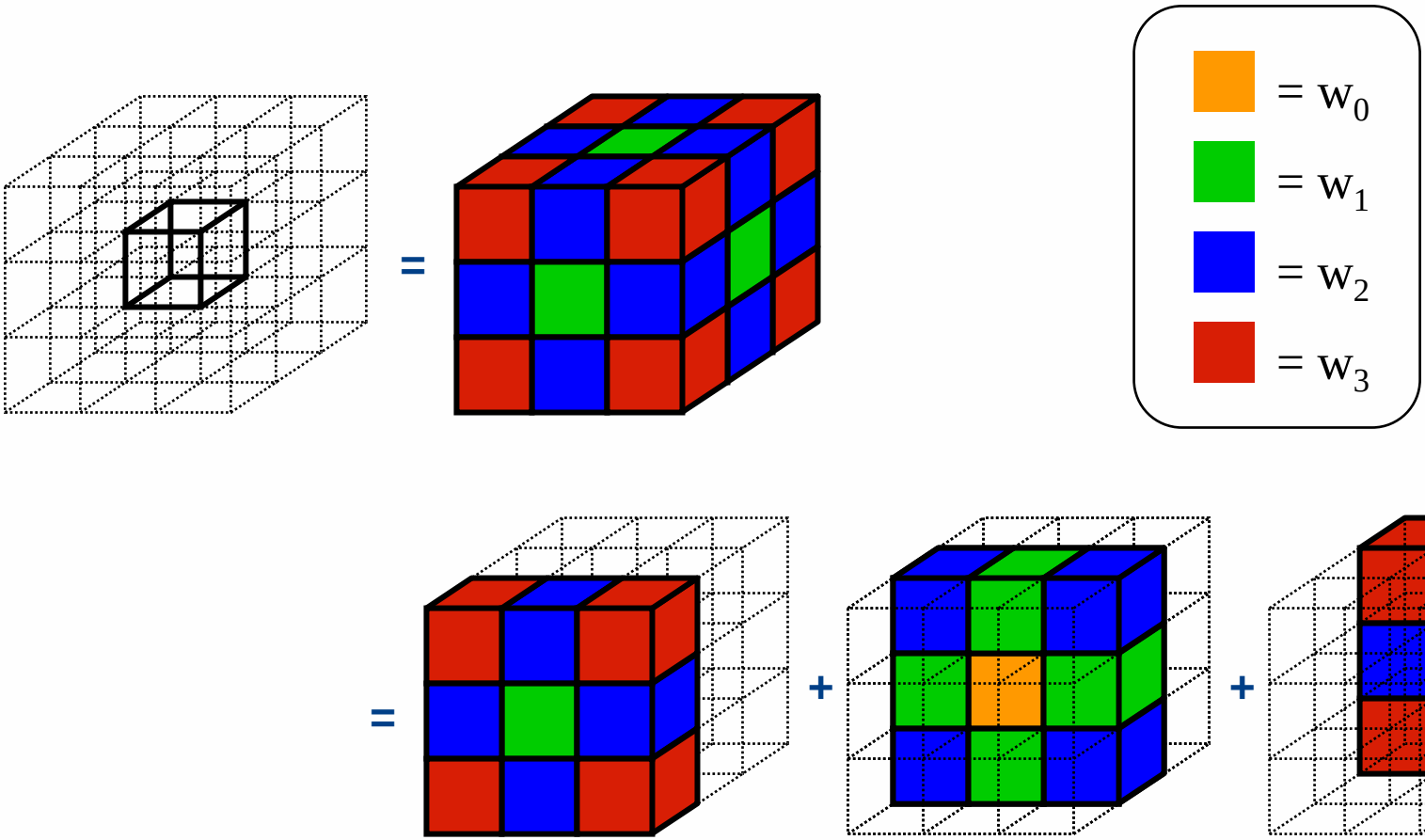
Overview of NAS MG



MG's projection/interpolation cycle



Multigrid: 27-Point Stencils



Multigrid: Stencils in Chapel

- Can write them out explicitly, as in Jacobi…

```
def rprj3(S, R) {
  param w: [0..3] real = (0.5, 0.25, 0.125, 0.0625);
  const Rstr = R.stride;

  forall ijk in S.domain do
    S(ijk) = w(0) * R(ijk)
           + w(1) * (R(ijk+Rstr*(1,0,0)) + R(ijk+Rstr*(-1,0,0))
                   + R(ijk+Rstr*(0,1,0)) + R(ijk+Rstr*(0,-1,0))
                   + R(ijk+Rstr*(0,0,1)) + R(ijk+Rstr*(0,0,-1)))
           + w(2) * (R(ijk+Rstr*(1,1,0)) + R(ijk+Rstr*(1,-1,0))
                   + R(ijk+Rstr*(-1,1,0)) + R(ijk+Rstr*(-1,-1,0))
                   + R(ijk+Rstr*(1,0,1)) + R(ijk+Rstr*(1,0,-1))
                   + R(ijk+Rstr*(-1,0,1)) + R(ijk+Rstr*(-1,0,-1))
                   + R(ijk+Rstr*(0,1,1)) + R(ijk+Rstr*(0,1,-1))
                   + R(ijk+Rstr*(0,-1,1)) + R(ijk+Rstr*(0,-1,-1)))
           + w(3) * (R(ijk+Rstr*(1,1,1) + R(ijk+Rstr*(1,1,-1))
                   + R(ijk+Rstr*(1,-1,1) + R(ijk+Rstr*(1,-1,-1))
                   + R(ijk+Rstr*(-1,1,1) + R(ijk+Rstr*(-1,1,-1))
                   + R(ijk+Rstr*(-1,-1,1) + R(ijk+Rstr*(-1,-1,-1)));
}
```

Multigrid: Stencils in Chapel

- …or, note that a stencil is simply a reduction over a small subarray expression
- Thus, stencils can be written in a “syntactically scalable” way using reductions:

```
def rprj3(S, R) {
  const Stencil: domain(3) = [-1..1, -1..1, -1..1], // 27-points
        w: [0..3] real = (0.5, 0.25, 0.125, 0.0625), // 4 wgts
        w3d = [(i,j,k) in Stencil] w((i!=0) + (j!=0) + (k!=0));

  forall ijk in S.domain do
    S(ijk) = + reduce [off in Stencil]
                          (w3d(off) * R(ijk + R.stride*off));
}
```

Fortran+MPI NAS MG *rprj3* stencil

```
      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
         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,i2,1) = 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,n3) = 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( i1,n2-
     1,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
               buff_len = buff_len + 1
               buff(buff_len,  buff_id ) = u( 1,  i2,i3)
            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
        j = k-1
        call comm3(s,m1j,m2j,m3j,j)
        return
        end
```

Example 3: Fast Multipole Method (FMM)

```
var OSgfn, ISgfn: [lvl in Levels] [SpsCubes(lvl)] [Sgfns(lvl)] [1..3] complex;
```

1D array over levels of the hierarchy

Example 3: Fast Multipole Method (FMM)

```
var OSgfn, ISgfn: [lvl in Levels] [SpsCubes(lvl)] [Sgfns(lvl)] [1..3] complex;
```

1D array over levels of the hierarchy

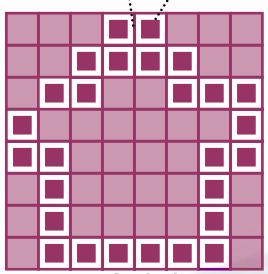
…of 3D sparse arrays of cubes (per level)

…of 1D vectors

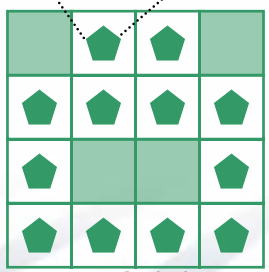
…of 2D discretizations of spherical functions, (sized by level)

…of complex values

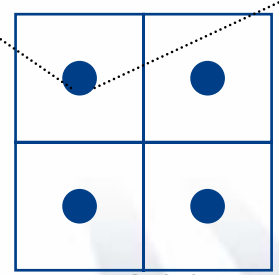
$x + y \cdot i$



OSgfn(1)



OSgfn(2)



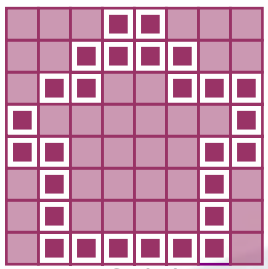
OSgfn(3)

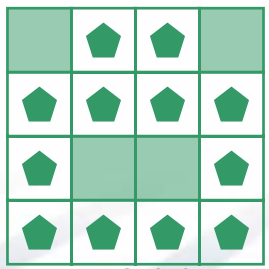
FMM: Supporting Declarations

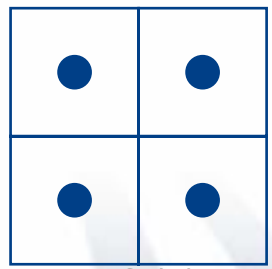
```
var OSgfn, ISgfn: [lvl in Levels] [SpsCubes(lvl)] [Sgfns(lvl)] [1..3] complex;
```

previous definitions:

```
var n: int = …;
var numLevels: int = …;

var Levels: domain(1) = [1..numLevels];

var scale: [lvl in Levels] int = 2**(lvl-1);
var SgFnSize: [lvl in Levels] int = computeSgFnSize(lvl);

var LevelBox: [lvl in Levels] domain(3) = [(1,1,1)..(n,n,n)] by scale(lvl);
var SpsCubes: [lvl in Levels] sparse subdomain(LevelBox) = …;

var Sgfns: [lvl in Levels] domain(2) = [1..SgFnSize(lvl), 1..2*SgFnSize(lvl)];
```


OSgfn(1)


OSgfn(2)

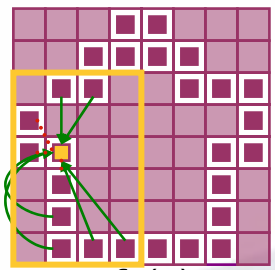

OSgfn(3)

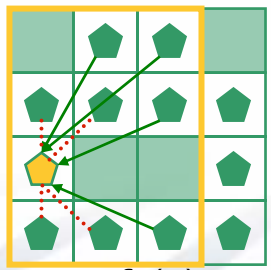
FMM: Computation

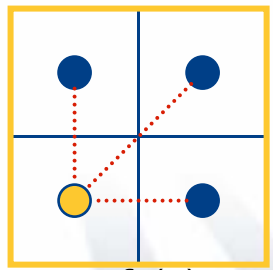
```
var OSgfn, ISgfn: [lvl in Levels] [SpsCubes(lvl)] [Sgfns(lvl)] [1..3] complex;
```

outer-to-inner translation:

```
for lvl in [1..numLevels) by -1 {
  …
  forall cube in SpsCubes(lvl) {
    forall sib in out2inSiblings(lvl, cube) {
      const Trans = lookupXlateTab(cube, sib);

      atomic ISgfn(lvl)(cube) += OSgfn(lvl)(sib) * Trans;
    }
  }
  …
}
```



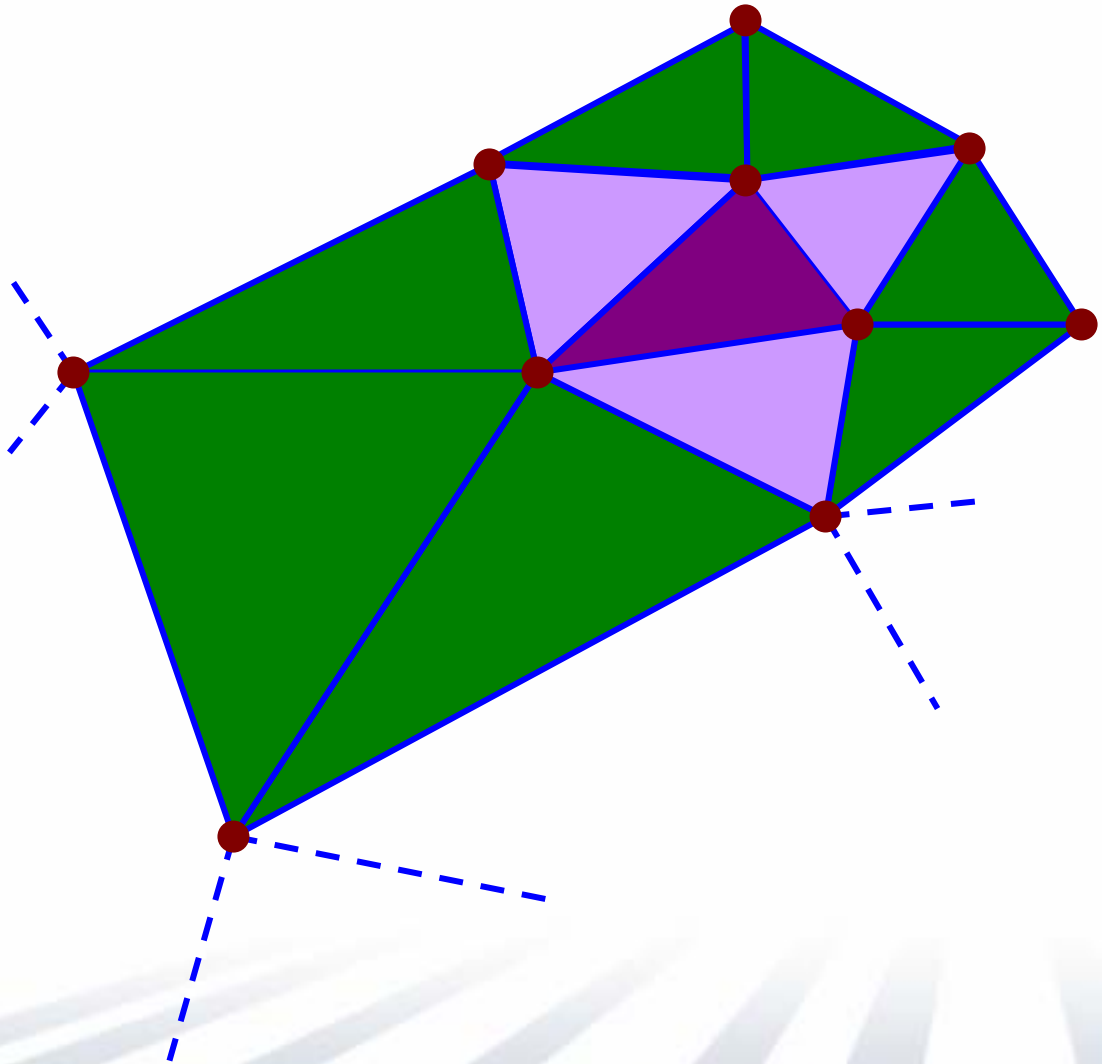




Fast Multipole Method: Summary

- Chapel code captures structure of data and computation far better than sequential Fortran/C versions (let alone MPI versions of them)
  - cleaner, more succinct, more informative
  - rich domain/array support plays a big role in this
- Parallelism shifts at different levels of hierarchy
  - Global view and syntactic separation of concerns helps here
  - Imagine writing in a fragmented language
- Code very clear to Boeing engineer familiar with FMM
- Yet, I've elided some non-trivial code (data distribution)

Example 4: Stencils on Unstructured Grids

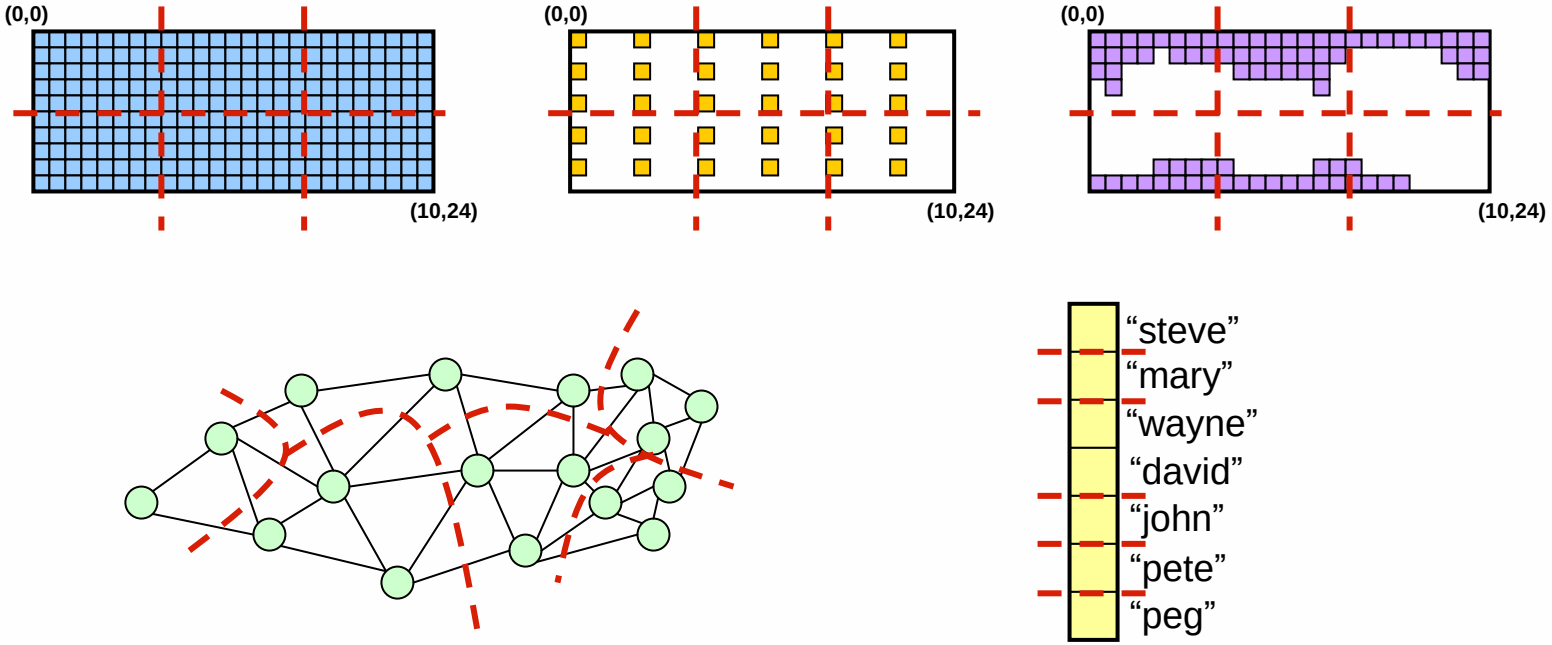


Key Data Parallel Challenge: Distributions

A domain's indices may be distributed…

…implying a distribution for its arrays

…and a default work assignment for iteration & slicing using the domain

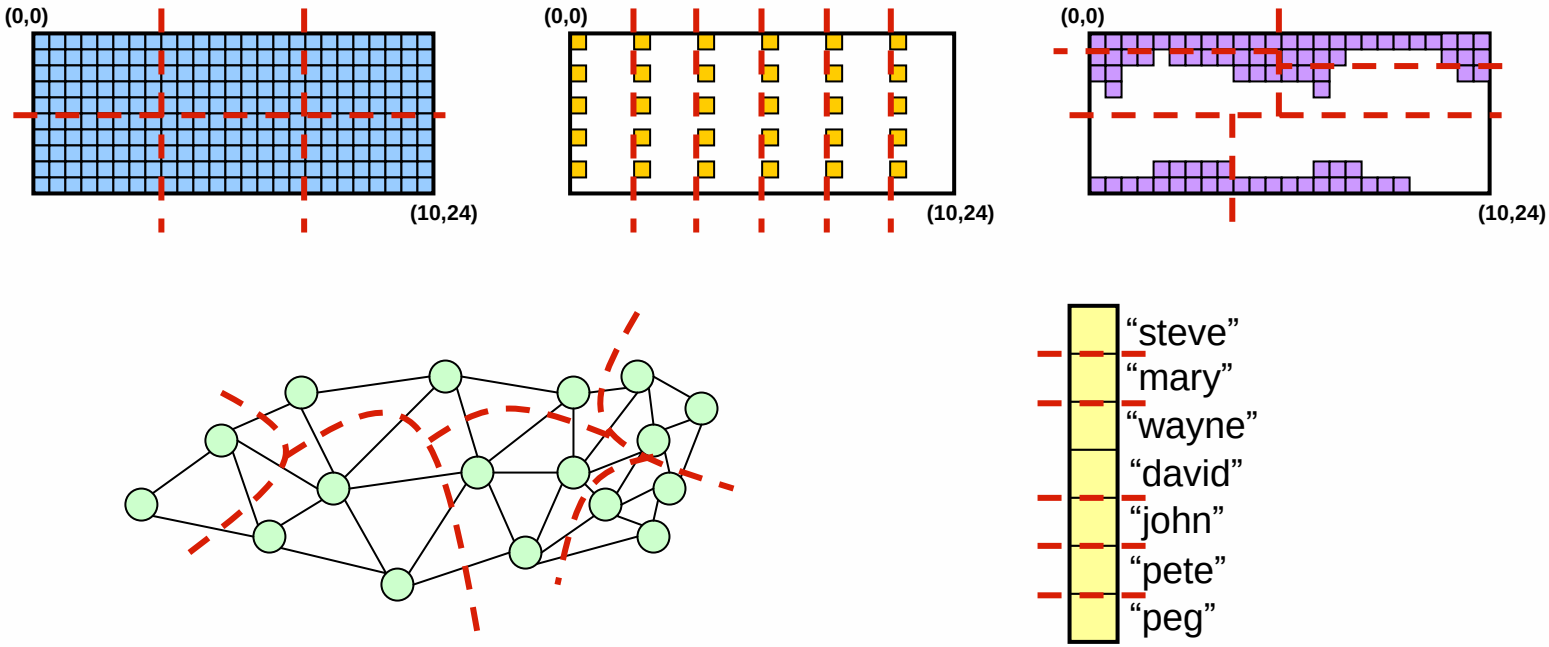


Key Data Parallel Challenge: Distributions

A domain's indices may be distributed…

…implying a distribution for its arrays

…and a default work assignment for iteration & slicing using the domain



Distributions Overview

Distributions: "recipes for distributed arrays"

- Intuitively, distributions implement the lowering…
  from: the user's global view of distributed data aggregates
  to: the fragmented implementation for distributed memory machines
- Define two primary things:
  - mapping of indices to locales
  - implementation of domain indices/array elements within a locale
- Author must implement an interface which supports:
  - allocation/reallocation of indices and elements
  - mapping functions (*e.g.*, index-to-locale, index-to-value)
  - iterators: parallel/serial; global/local
  - communication idioms
- Chapel provides a standard library of distributions…
  …written using the same mechanism as user-defined distributions
  …tuned for different platforms to maximize performance

Outline

- ✓ Introduction to Chapel
- ✓ Global-View Programming
- ✓ Data Parallel Examples: the Stencil Ramp
- ➢ Task Parallel Features & Examples
- ❑ Status & Summary

Task Parallelism: Task Creation

- *begin:* creates a task for future evaluation

```
begin DoThisTask();
WhileContinuing();
TheOriginalThread();
```

- *co-begin:* supports a structured list of sibling tasks:

```
computePivot(lo, hi, data);
cobegin {
  Quicksort(lo, pivot, data);
  Quicksort(pivot, hi, data);
} // implicit synchronization here
```

```
cobegin {
  ComputeTaskA(…);
  ComputeTaskB(…);
  ComputeTaskC(…);
} // implicit synch
```

- *co-forall:* loop-style construct for generating sibling tasks

```
coforall e in Edges {
  exploreEdge(e);
} // implicit synchronization here
```

Task Parallelism: Task Coordination

- *sync variables:* store full/empty state along with value

```
var result: sync real;     // result is initially empty
cobegin {
  … = result;              // block until full, leave empty
  result = …;              // block until empty, leave full
}
result.readFF();           // read when full, leave full;
                           // other variations also supported
```

- *single-assignment variables:* writable once only

```
var result: single real = begin f(); // result initially empty
…                      // do some other things
total += result;       // block until result has been filled
```

- *atomic sections:* support transactions against memory

```
atomic {
  newnode.next = insertpt;
  newnode.prev = insertpt.prev;
  insertpt.prev.next = newnode;
  insertpt.prev = newnode;
}
```

Task Parallelism: Task Placement

- *on clauses:* indicate where tasks should execute
  - in a data-driven manner…

```
computePivot(lo, hi, data);
cobegin {
  on A(lo)    do Quicksort(lo, pivot, data);
  on A(pivot) do Quicksort(pivot, hi, data);
}
```

  - …or by naming machine resources explicitly

```
// Chapel provides: const Locale: [0..numLocales-1] locale;
on Locale(0) begin gatherResults();
coforall loc in 1..numLocales-1 {
  on Locale(loc) do compute();
}
```

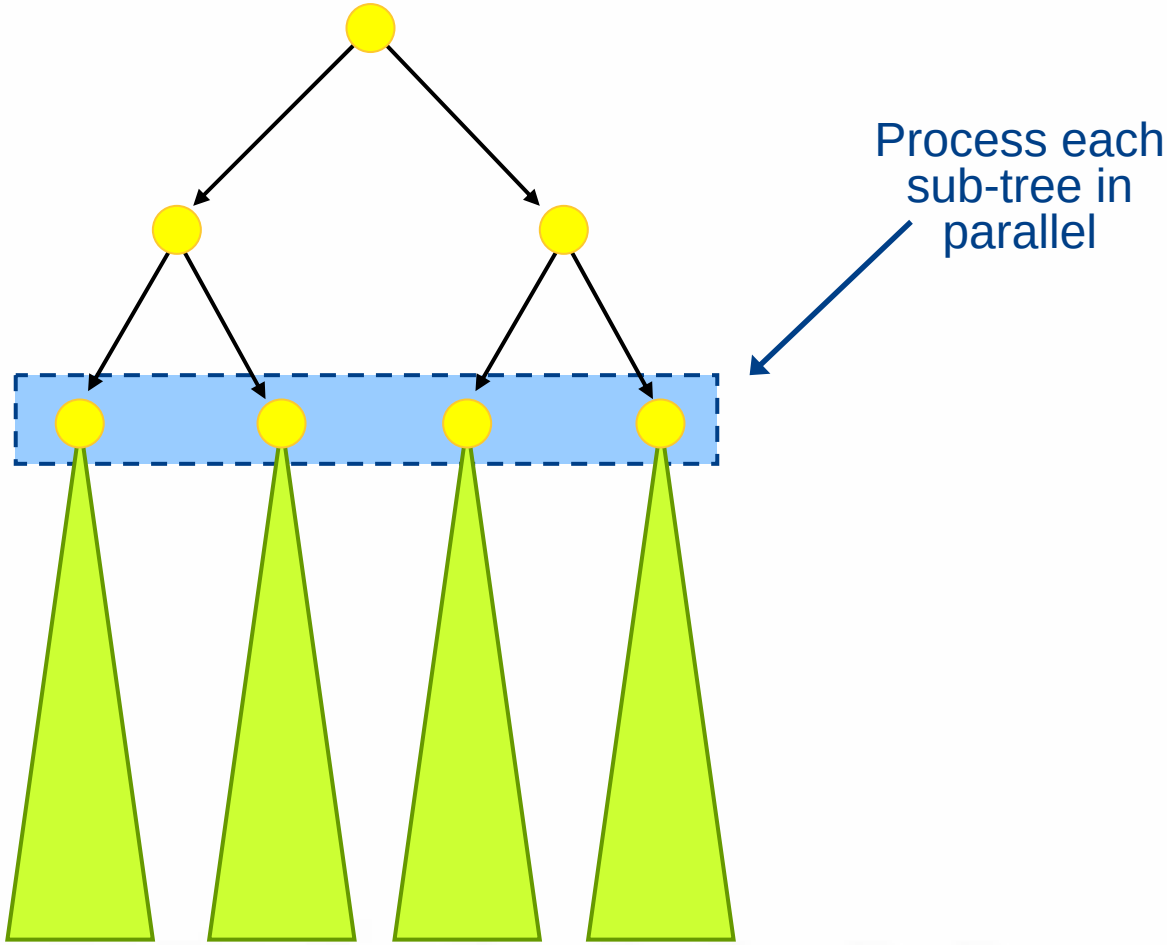
Example 1: UTS

UTS: Unbalanced Tree Search Benchmark

- performs exhaustive search on a variety of random, unbalanced trees
- requires dynamic load balancing to achieve good speedup
- employs cryptographically strong tree generation using SHA1
- joint effort between Ohio State, U. of North Carolina, U. of Maryland

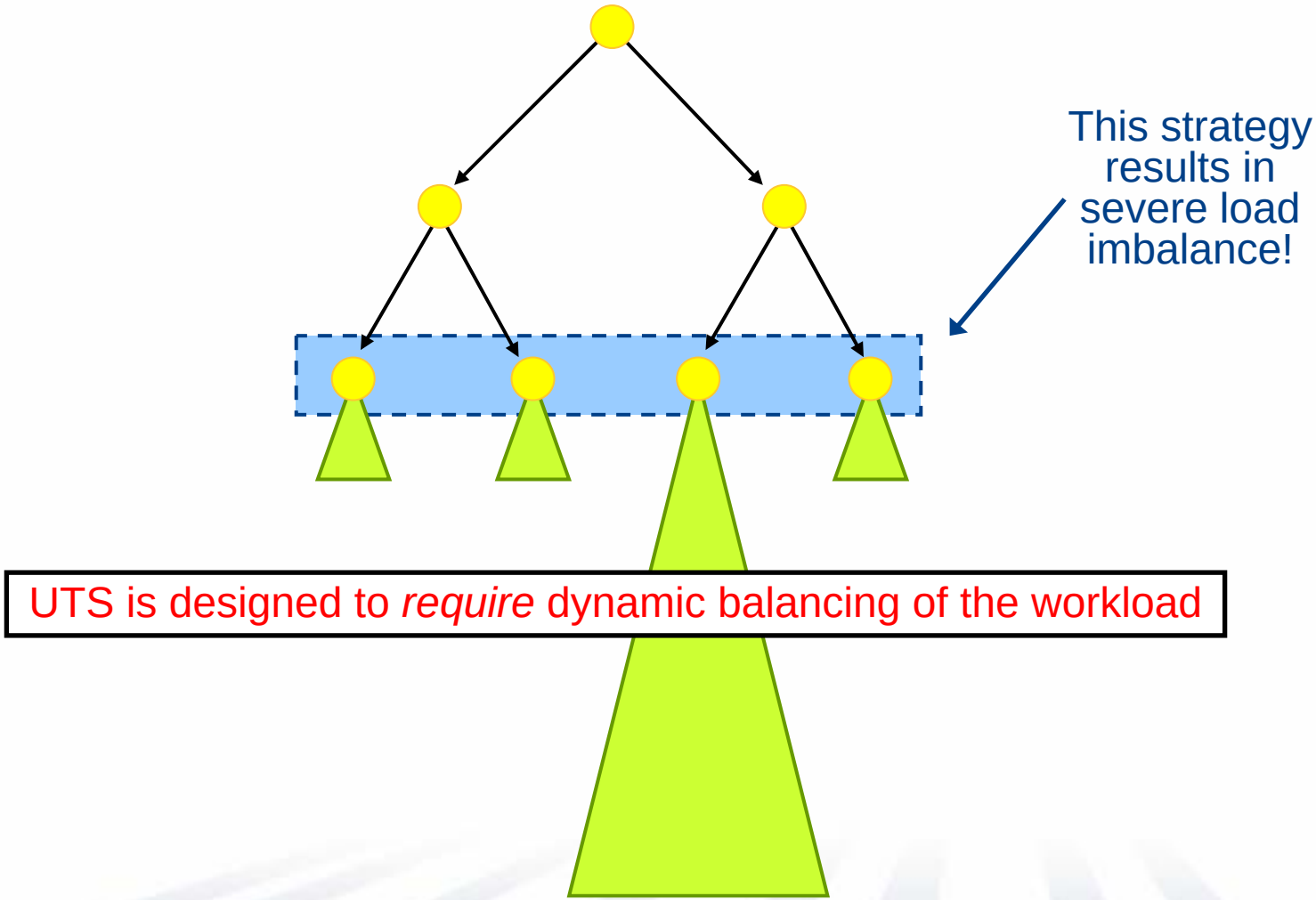
contents adapted from James Dinan, the Ohio State University

Consider a balanced Tree …



contents adapted from James Dinan, the Ohio State University

What if the tree is unbalanced?



contents adapted from James Dinan, the Ohio State University

UTS: Naive Algorithm (Sketch)

```
exploreTree(rand(SEED));

def exploreTree(id) {
  const numChildren = computeNumChildren(rand(id));
  coforall c in 1..numChildren do
    exploreTree(rand(id,c));
}
```

The Big Question

Q: How will this *coforall* be implemented?

- spawn a thread per task?
- use a fixed number of threads and a task pool?
- use some sort of peer-based work-sharing or stealing algorithm a la Cilk or the Cray MTA?

A: Could be any of these; semantics are separate from implementing mechanisms

UTS: Naive Algorithm (Sketch)

```
exploreTree(rand(SEED));

def exploreTree(id) {
  const numChildren = computeNumChildren(rand(id));
  coforall c in 1..numChildren do
    exploreTree(rand(id,c));
}
```

An Open Question

Q: For the user who wants more control over their program's implementation, what is the equivalent of a *distribution* for task-parallel codes?

(*i.e.*, How should the recipe for throttling parallelism be expressed and specified?)

Chapel Load Balancing Code

```
var numTasks: sync int = 0;
var terminated: single bool;
```

Declare synchronized variables to throttle number of live tasks

```
def balance_load(inout q: DeQueue(TreeNode)) {
  if (q.size > 2*chunkSize && numTasks.readXX() < MAX_TASKS) {

    // Split chunkSize nodes into a new queue
    var work = q.split(chunkSize);

    // Spawn a new worker on this queue
    numTasks += 1;
    begin create_tree(work);
  }
}
```

Check if we have surplus work and whether the runtime is running low on available tasks

Spawn off surplus work and rely on the runtime to balance the load.

```
def create_tree(inout q: DeQueue(TreeNode)) {
  ...

  // Update task counts; detect termination

  var numTasks_l = numTasks;
  numTasks_l -= 1;
  if numTasks_l == 0 then
    terminated = true;
  numTasks = numTasks_l;
}
```

When a task completes it updates the shared counter and checks to see if it is the last one.

UTS MPI Load Balancing Code

```
#include <stdint.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <mpi.h>
#include "uts_dm.h"
#include "dequeue.h"
#include "ctrk.h"

#define DBG_GEN    1
#define DBG_CHUNK  2
#define DBG_TOKEN  4
#define DBG_MSGCNT 8

//#define DEBUG_LEVEL (DBG_GEN | DBG_CHUNK)
#define DEBUG_LEVEL 0

#define DEBUG(dbg_class, command) if (DEBUG_LEVEL & (dbg_class)) {
command; }

enum uts_tags { MPIWS_WORKREQUEST = 1, MPIWS_WORKRESPONSE,
MPIWS_TDTOKEN, MPIWS_STATS };
enum colors { BLACK = 0, WHITE_PINK, RED };
char *color_names[] = { "BLACK", "WHITE", "PINK", "RED" };

typedef struct {
  enum colors color;
  long send_count;
  long recv_count;
} td_token_t;

/** Global State **/
static StealStack  stealStack; // Stores the UTS-related stats
static dequeue_    localQueue; // double ended queue of local only work
static enum colors my_color;   // Ring-based termination detection
static int         last_steal; // Rank of last thread stolen from
static long        chunks_recvd; // Total messages received
static long        chunks_sent;  // Total messages sent
static long        ctrl_recvd; // Total messages received
static long        ctrl_sent;  // Total messages sent

/** Global Parameters: Set in ss_init() **/
static int comm_size, comm_rank;
static const int default_polling_interval = 32;

/** Global Communication handles **/
static MPI_Request  wrin_request;  // Incoming steal request
static MPI_Request  wrout_request; // Outbound steal request
static MPI_Request  iw_request;    // Outbound steal request
static MPI_Request  td_request;    // Term. Detection listener

/** Global Communication Buffers **/
static long         wrin_buff;     // Buffer for accepting incoming work requests
static long         wrout_buff;    // Buffer to send outgoing work requests
static void         *iw_buff;      // Buffer to receive incoming work
static td_token_t  td_token;       // Dijkstra's token

/** Work Data Structures */

/*************************************************************
 *  functions                                                *
 *************************************************************/

void * release(StealStack *s);

/* Fatal error */
void ss_abort(int error)
{
  MPI_Abort(MPI_COMM_WORLD, error);
}

char * ss_get_par_description()
{
  return "MPI Workstealing";
}

/** Make progress on any outstanding WORKREQUESTs or WORKRESPONSES
 */
void ws_make_progress(StealStack *s)
{
  MPI_Status  status;
  int         flag, index;
  void        *work;

  // Test for incoming work_requests
  MPI_Test(&wrin_request, &flag, &status);

  if (flag) {
    // Got a work request
    ++ctrl_recvd;

    /* Repost that work request listener */
    MPI_Irecv(&wrin_buff, 1, MPI_LONG, MPI_ANY_SOURCE,
MPIWS_WORKREQUEST, MPI_COMM_WORLD,
        &wrin_request);

    index = status.MPI_SOURCE;

    /* Check if we have any surplus work */
    if (s->localWork > 2*s->chunk_size) {
      work = release(s);
      DEBUG(DBG_CHUNK, printf(" -Thread %d: Releasing a chunk to thread
%d\n", comm_rank, index));
      ++chunks_sent;
      MPI_Send(work, s->chunk_size*s->work_size, MPI_BYTE, index,
          MPIWS_WORKRESPONSE, MPI_COMM_WORLD);
      free(work);

      /* If we a node to our left steals from us, our color becomes black */
      if (index < comm_rank) my_color = BLACK;
    } else {
      // Send a "no work" response
      ++ctrl_sent;
      MPI_Send(NULL, 0, MPI_BYTE, index, MPIWS_WORKRESPONSE,
MPI_COMM_WORLD);
    }
  }

  return;
}
```

```
/* release k values from bottom of local stack */
void * release(StealStack *s)
{
  StealStackNode *node;
  void *work;

  /* Get a node from the back of the queue to release */
  node = deq_popBack(localQueue);

  if (node) {
    /* If this node is not full we can't release it. */
    if (node->head != s->chunk_size)
      ss_error("release(): Attempted to release a non-full node", 1);
    work = node->work;

    ctrk_put(comm_rank, node->work);

    free(node);
    s->localWork -= s->chunk_size;
    s->nRelease ++;

    return work;
  }
  else {
    ss_error("release(): Do not have a chunk to release", 1);
    return NULL; // Unreachable
  }
}

/**
 * ensure local work exists, find it if it doesnt
 *  returns process id where work is stolen from if no can be found locally
 *  returns -1 if no local work exists and none could be stolen
 **/
int ensureLocalWork(StealStack *s)
{
  MPI_Status  status;
  int         flag;

  if (s->localWork < 0)
    ss_error("ensureLocalWork(): localWork count is less than 0!", 2);

  /* If no more work */
  while (s->localWork == 0) {
    if (my_color == PINK)
      ss_setState(s, SS_IDLE);
    else
      ss_setState(s, SS_SEARCH);

    /* Check if we should post another steal request */
    if (wrout_request == MPI_REQUEST_NULL && my_color != PINK) {
      if (iw_buff == NULL) {
        iw_buff = malloc(s->chunk_size*s->work_size);
        if (!iw_buff)
          ss_error("ensureLocalWork(): Out of memory\n", 5);
      }

      /* Send the request and wait for a work response */
      last_steal = (last_steal + 1) % comm_size;
      if (last_steal == comm_rank) last_steal = (last_steal + 1) % comm_size;

      DEBUG(DBG_CHUNK, printf("Thread %d: Asking thread %d for work\n",
comm_rank, last_steal));

      ++ctrl_sent;

      MPI_Isend(&wrout_buff, 1, MPI_LONG, last_steal,
MPIWS_WORKREQUEST, MPI_COMM_WORLD, &wrout_request);
      MPI_Irecv(iw_buff, s->chunk_size*s->work_size, MPI_BYTE, last_steal,
MPIWS_WORKRESPONSE,
          MPI_COMM_WORLD, &iw_request);
    }

    // Call into the stealing progress engine and update our color
    ws_make_progress(s);

    // Test for incoming work
    MPI_Test(&iw_request, &flag, &status);
    if (flag && wrout_request != MPI_REQUEST_NULL) {
      int work_rcv;

      MPI_Get_count(&status, MPI_BYTE, &work_rcv);

      if (work_rcv > 0) {
        StealStackNode *node;

        ++chunks_recvd;
        DEBUG(DBG_CHUNK, printf(" -Thread %d: Incoming Work received, %d
bytes\n", comm_rank, work_rcv));

        if (work_rcv != s->work_size * s->chunk_size) {
          ss_error("ws_make_progress(): Work received size does not equal
chunk size", 10);
        }

        /* Create a new node to attach this work to */
        node = (StealStackNode*)malloc(sizeof(StealStackNode));

        if (!node)
          ss_error("ensureLocalWork(): Out of virtual memory.", 10);

        node->head = s->chunk_size;
        node->work = iw_buff;
        iw_buff    = NULL;

        ctrk_get(comm_rank, node->work);

        /* Push stolen work onto the back of the queue */
        s->nSteal++;
        s->localWork += s->chunk_size;
        deq_pushBack(localQueue, node);
#ifdef TRACE
        /* Successful Steal */
        ss_markSteal(s, status.MPI_SOURCE);
#endif
      }
      else {
        // Received "No Work" message
        ++ctrl_recvd;
      }

      // Clear on the outgoing work_request
      MPI_Wait(&wrout_request, &status);
    }

    /* Test if we have the token */
    MPI_Test(&td_request, &flag, &status);

    if (flag) {
      enum colors next_token;
      int         forward_token = 1;

      DEBUG(DBG_TOKEN, printf("ensureLocalWork(): Thread %d
received %s token\n", comm_rank, color_names[td_token.color]));
      switch (td_token.color) {
        case WHITE:
          if (s->localWork == 0) {
            if (comm_rank == 0 && my_color == WHITE) {
              if (td_token.recv_count != td_token.send_count) {
                // There are outstanding messages, try again
                DEBUG(DBG_MSGCNT, printf(" TD_RING: In-flight work,
recirculating token\n"));
                my_color = WHITE;
                next_token = WHITE;
              } else {
                // Termination detected, pass RED token
                my_color = PINK;
                next_token = PINK;
              }
            } else if (my_color == WHITE) {
              next_token = WHITE;
            } else {
              // Every time we forward the token, we change
              // our color back to white
              my_color = WHITE;
              next_token = BLACK;
            }

            // forward message
            forward_token = 1;
          }
          else {
            forward_token = 0;
          }
          break;
        case PINK:
          if (comm_rank == 0) {
            if (td_token.recv_count != td_token.send_count) {
              // There are outstanding messages, try again
              DEBUG(DBG_MSGCNT, printf(" TD_RING: ReCirculating pink
token nr=%ld ns=%ld\n", td_token.recv_count, td_token.send_count));
              my_color = PINK;
              next_token = PINK;
            } else {
              // Termination detected, pass RED token

my_color = RED;
              next_token = RED;
            }
          } else {
            my_color = PINK;
            next_token = PINK;
          }

          forward_token = 1;
          break;
        case BLACK:
          // Non-Termination; Token must be recirculated
          if (comm_rank == 0)
            next_token = WHITE;
          else {
            my_color = WHITE;
            next_token = BLACK;
          }
          forward_token = 1;
          break;
        case RED:
          // Termination; Set our state to RED and circulate term message
          my_color = RED;
          next_token = RED;

          if (comm_rank == comm_size - 1)
            forward_token = 0;
          else
            forward_token = 1;
          break;
      }

      /* Forward the token to the next node in the ring */
      if (forward_token) {
        td_token.color = next_token;

        /* Update token counters */
        if (comm_rank == 0) {
          if (td_token.color == PINK) {
            td_token.send_count = ctrl_sent;
            td_token.recv_count = ctrl_recvd;
          } else {
            td_token.send_count = chunks_sent;
            td_token.recv_count = chunks_recvd;
          }
        } else {
          if (td_token.color == PINK) {
            td_token.send_count += ctrl_sent;
            td_token.recv_count += ctrl_recvd;
          } else {
            td_token.send_count += chunks_sent;
            td_token.recv_count += chunks_recvd;
          }
        }

        DEBUG(DBG_TOKEN, printf("ensureLocalWork(): Thread %d
forwarding %s token\n", comm_rank, color_names[td_token.color]));
        MPI_Send(&td_token, sizeof(td_token_t), MPI_BYTE,
(comm_rank+1)%comm_size,
            MPIWS_TDTOKEN, MPI_COMM_WORLD);

        if (my_color != RED) {
          /* re-Post termination detection listener */
          int j = (comm_rank == 0) ? comm_size - 1 : comm_rank - 1; //
Receive the token from the processor to your left
          MPI_Irecv(&td_token, sizeof(td_token_t), MPI_BYTE, j,
MPIWS_TDTOKEN, MPI_COMM_WORLD, &td_request);
        }
      }

      if (my_color == RED) {
        // Clean up outstanding requests.
        // This is safe now that the pink token has mopped up all outstanding
messages.
        MPI_Cancel(&wrin_request);
        if (iw_request != MPI_REQUEST_NULL)
          MPI_Cancel(&iw_request);
        // Terminate
        return -1;
      }
    }
  }

  return 0;  // Local work exists
}

/* restore stack to empty state */
void mkEmpty(StealStack *s)
{
  deq_mkEmpty(localQueue);
  s->localWork = 0;
}

/* initialize the stack */
StealStack* ss_init(int *argc, char ***argv)
{
  StealStack* s = &stealStack;

  MPI_Init(argc, argv);

  MPI_Comm_size(MPI_COMM_WORLD, &comm_size);
  MPI_Comm_rank(MPI_COMM_WORLD, &comm_rank);

  s->nNodes    = 0;
  s->maxDepth  = 0;
  s->nAcquire  = 0;
  s->nRelease  = 0;
  s->nSteal    = 0;
  s->nFail     = 0;
  localQueue = deq_create();
  mkEmpty(s);

  return s;
}

int ss_start(int work_size, int chunk_size)
{
  int i;
  StealStack* s = &stealStack;

  s->work_size = work_size;
  s->chunk_size = chunk_size;

  if (polling_interval == 0) {
    // Set a default polling interval
    polling_interval = default_polling_interval;
  }

  if (comm_rank == 0)
    printf("Progress engine polling interval = %d\n", polling_interval);

  // Start searching for work at the next thread to our right
  wrout_request = MPI_REQUEST_NULL;
  wrout_buff    = chunk_size;
  last_steal    = comm_rank;
  iw_request    = MPI_REQUEST_NULL;
  iw_buff       = NULL; // Allocated on demand
  chunks_sent   = 0;
  chunks_recvd  = 0;
  ctrl_sent     = 0;
  ctrl_recvd    = 0;

  // Termination detection
  my_color      = WHITE;

  td_token.color = BLACK;

  // Setup non-blocking recieve for recieving shared work requests
  MPI_Irecv(&wrin_buff, 1, MPI_LONG, MPI_ANY_SOURCE,
MPIWS_WORKREQUEST, MPI_COMM_WORLD, &wrin_request);

  /* Set up the termination detection receives */

  if (comm_rank == 0) {
    // Thread 0 initially has a black token
    td_request = MPI_REQUEST_NULL;
  } else {
    /* Post termination detection listener */
    j = (comm_rank == 0) ? comm_size - 1 : comm_rank - 1; // Receive
the token from the processor to your left

    MPI_Irecv(&td_token, sizeof(td_token_t), MPI_BYTE, j,
MPIWS_TDTOKEN, MPI_COMM_WORLD,
        &td_request);
  }

  return 1;
}

void ss_stop()
{
  DEBUG(DBG_MSGCNT, printf(" Thread %d: chunks_sent=%d,
chunks_recvd=%d\n", comm_rank, chunks_sent, chunks_recvd));
  DEBUG(DBG_MSGCNT, printf(" Thread %d: ctrl_sent  =%d, ctrl_recvd
=%d\n", comm_rank, ctrl_sent, ctrl_recvd));
  return;
}

void ss_finalize()
{
  MPI_Finalize();
}

/* local push */
void ss_put_work(StealStack *s, void* node_c)
{
  StealStackNode* n;
  void *work;

  /* If the stack is empty, push an empty StealStackNode. */
  if (deq_isEmpty(localQueue)) {
    n = malloc(sizeof(StealStackNode));
    work = malloc(s->chunk_size*s->work_size);
    if (!n || !work) ss_error("ss_put_work(): Out of virtual memory", 3);
    n->work = work;
    n->head = 0;
    deq_pushFront(localQueue, n);
  }

  n = deq_peekFront(localQueue);

  /* If the current StealStackNode is full, push a new one. */
  if (n->head == s->chunk_size) {
    n = malloc(sizeof(StealStackNode));
    work = malloc(s->chunk_size*s->work_size);
    if (!n || !work) ss_error("ss_put_work(): Out of virtual memory", 3);
    n->head = 0;
    n->work = work;
    deq_pushFront(localQueue, n);
  }
  else if (n->head > s->chunk_size)
    ss_error("ss_put_work(): Block has overflowed!", 3);

  /* Copy the work to the local queue, increment head */
  memcpy(((uint8_t*)n->work)+(s->work_size*n->head), node_c, s->work_size);

  n->head++;
  s->localWork++;
  s->maxDepth = max(s->globalWork + s->localWork, s->maxDepth);

  /* If there is sufficient local work, release a chunk to the global queue */
  if (s->nNodes % polling_interval == 0) {
    ss_setState(s, SS_OVH);
    ws_make_progress(s);
    ss_setState(s, SS_WORK);
  }
}

/**
  if no work is found no local work is found, and
  none can be stolen, return original s and c is null
  if work is found, return the StealStack and set c to
  return node
**/
int ss_get_work(StealStack *s, void* node_c)
{
  //int victimId;
  StealStackNode* n;

  /* Call ensureLocalWork() to make sure there is work on our local queue.
   * If the local queue is empty, this will get work from the global queue */
  if (ensureLocalWork(s) == -1) {
    DEBUG(DBG_GEN, printf("StealStack::pop - stack is empty and no work can be
found\n");fflush(NULL));
    ss_setState(s, SS_IDLE);
    node_c = NULL;
    return STATUS_TERM;
  }

  /* We have work */
  ss_setState(s, SS_WORK);

  /* ensureLocalWork() ensures that the local work queue is not empty,
   * so at this point we know there must be work available */
  n = deq_peekFront(localQueue);

  /* head always points at the next free entry in the work array */
  n->head--;
  memcpy(node_c,((uint8_t*)n->work) + ((s->work_size)*(n->head)),s->work_size);

  /* This chunk in the queue is empty so dequeue it */
  if(n->head == 0) {
    deq_popFront(localQueue);
    free(n->work);
    free(n);
  }
  else if (n->head < 0) {
    /* This happens if an empty chunk is left on the queue */
    fprintf(stderr, "ss_get_work(): called with n->head = 0, s->localWork=%d or %d
(mod %d)\n",
        s->localWork, s->localWork % s->chunk_size, s->chunk_size);
    ss_error("ss_get_work(): Underflow!", 5);
  }

  s->nNodes++;
  s->localWork--;

  return STATUS_HAVEWORK;
}

/* Returns true to the thread that has the stats
 * s should be able to hold NUM_THREADS stealstacks! */
int ss_gather_stats(StealStack *s, int *count)
{
  int i;
  MPI_Status status;

  *count = comm_size;

  /* Gather stats onto thread 0 */
  if (comm_rank > 0) {
    MPI_Send(&stealStack, sizeof(StealStack), MPI_BYTE, 0, MPIWS_STATS,
MPI_COMM_WORLD);
    return 0;
  } else {
    memcpy(s, &stealStack, sizeof(StealStack));

  for(i = 1; i < *count; i++) {
      MPI_Recv(&(s[i]), sizeof(StealStack), MPI_BYTE, i, MPIWS_STATS,
MPI_COMM_WORLD, &status);
    }
  }
  return 1;
}

int ss_get_thread_num() {
  return comm_rank;
}

int ss_get_num_threads() {
  return comm_size;
}
```


contents adapted from James Dinan, the Ohio State University

Example 2: MADNESS

- ***MADNESS:***
  - Multiresolution ADaptive NumErical Scientific Simulation
  - a framework for scientific simulation in many dimensions using adaptive multiresolution methods in multiwavelet bases
- People:
  - Gregory Beylkin (University of Colorado), George Fann (Oak Ridge National Laboratory), Zhenting Gan (CCSG), Robert Harrison (CCSG), Martin Mohlenkamp (Ohio University), Fernando Perez (University of Colorado), P. Sadayappan (The Ohio State University), Takeshi Yanai (CCSG)

contents adapted from James Dinan, the Ohio State University

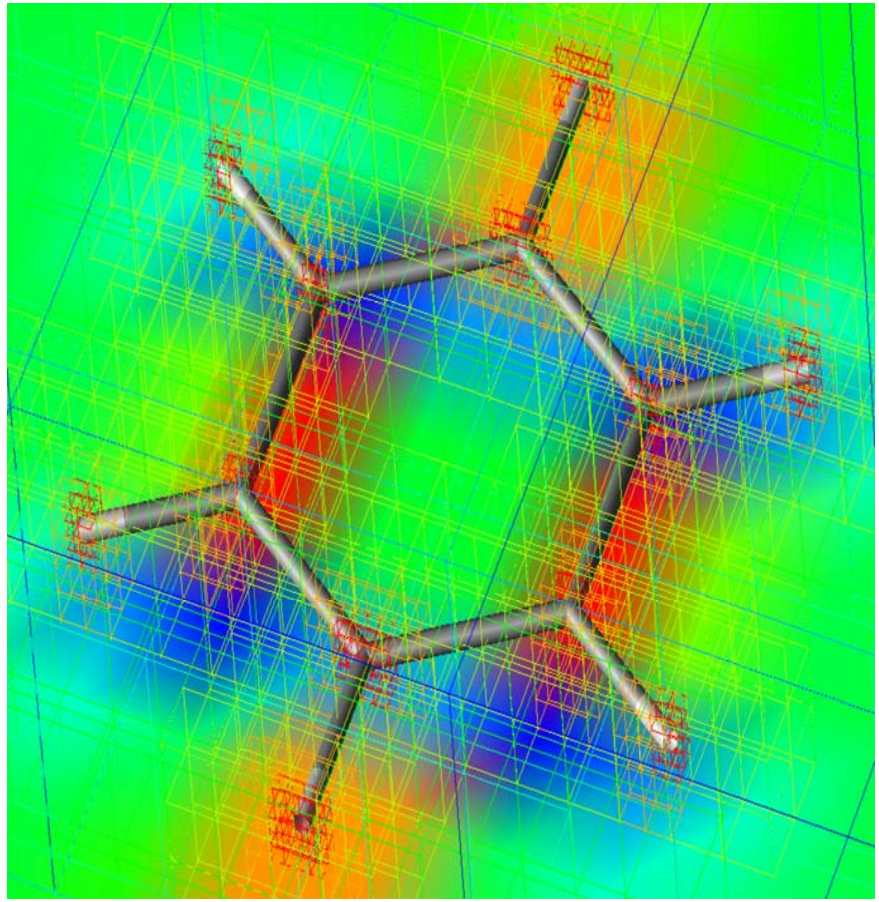
What does Madness do?

- Think of Madness as a math library
- Numerical representations for analytic functions
  - Stored in the scaling function (Gauss Legendre Polynomial) and Multiwavelet bases
  - Operations on functions become fast with guaranteed precision
  - Differential and Integral operators become O(n) in numerical representation
- Applications that can benefit from Madness include:
  - Density Functional Theory (DFT) (Quantum chemistry domain)
    - Explore electronic structure of many-body systems
  - Fluid dynamics
  - Climate modeling
  - Etc …

contents adapted from James Dinan, the Ohio State University

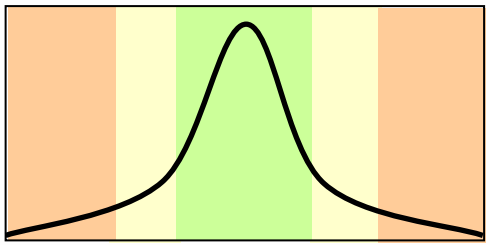
Numerical Representation for Functions

- Analytic function is *projected* into the numerical representation
- Approximate the function using basis functions
  - Similar to Fourier, but basis functions have compact support
  - Approximation is over a closed interval of interest
- Recursively subdivide the analytic function spatially to achieve desired accuracy
- Avoid extra computation in uninteresting areas
- Store the result in a *Function Tree*
  - 1d: Binary Tree
  - 2d: Quad Tree
  - 3d: Oct Tree



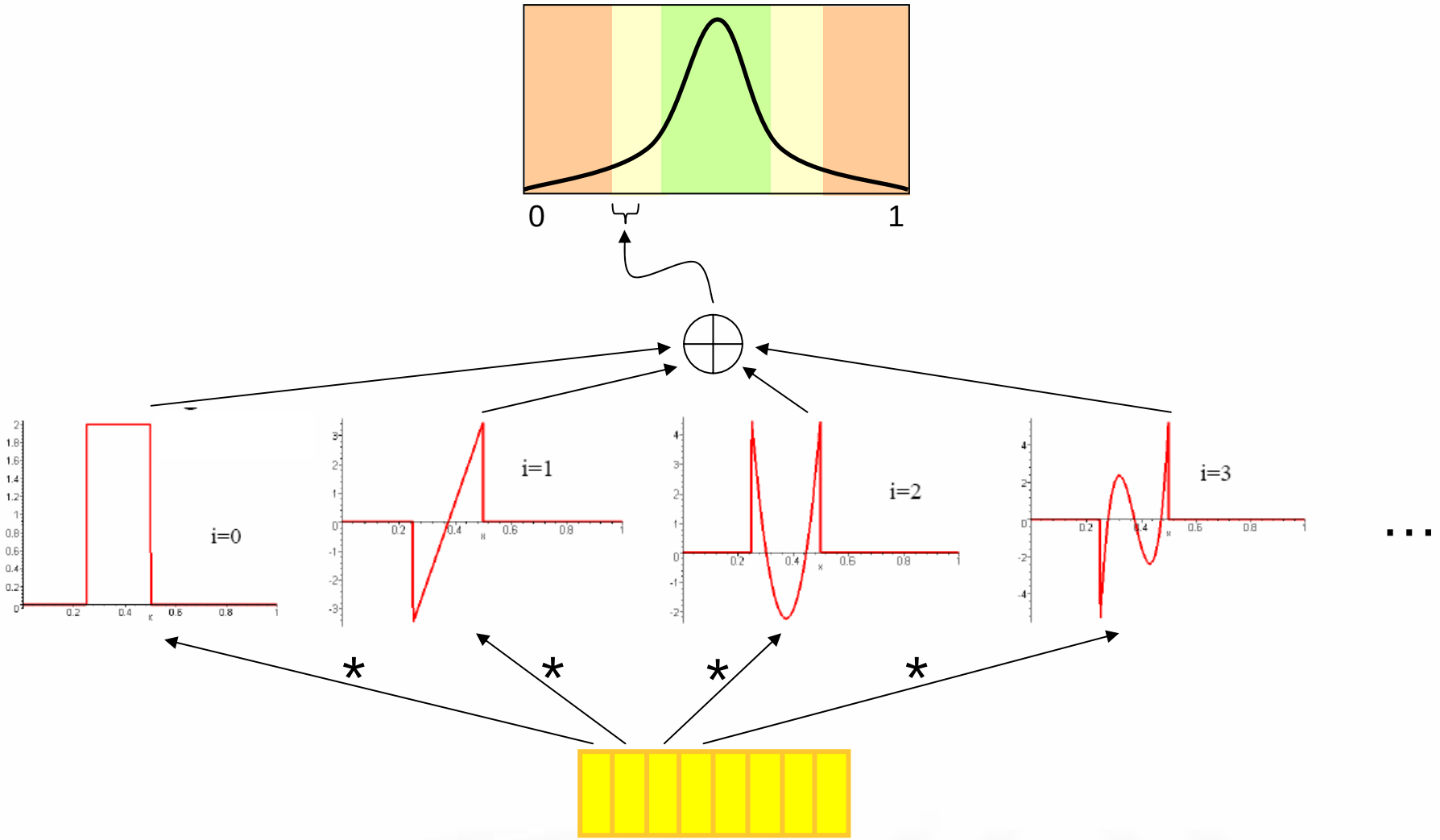
contents adapted from James Dinan, the Ohio State University

The 1d Function Tree of a Gaussian



contents adapted from James Dinan, the Ohio State University

Function Evaluation in the Numerical Representation

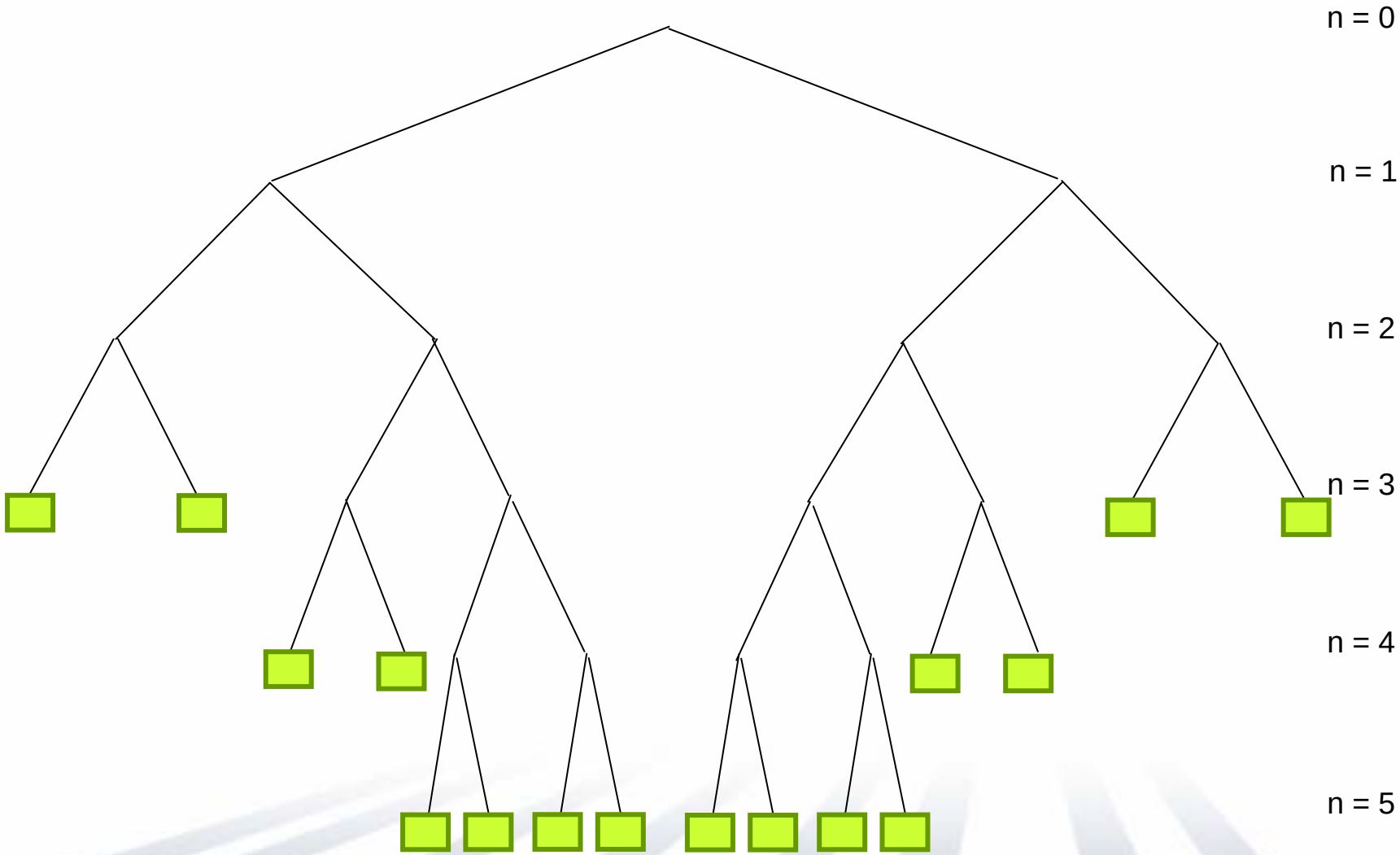


contents adapted from James Dinan, the Ohio State University

Core Algorithm: Differentiation

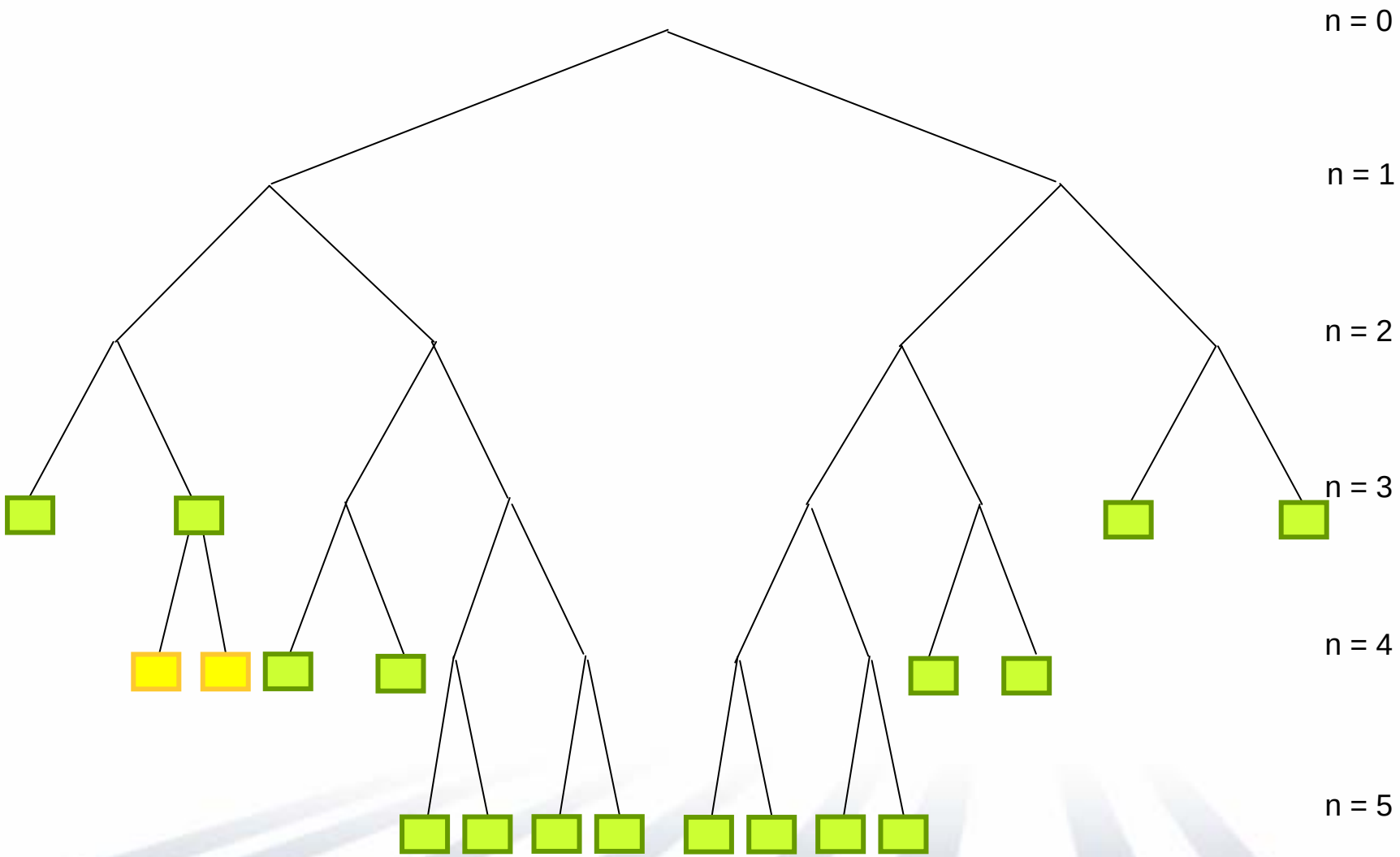
- Perform: `df = f.diff()`
- Walk down the tree and everywhere that we have coefficients, perform differentiation
- Performing differentiation involves getting our left and right neighbors and applying the derivative operator

Differentiation: I have neighbors



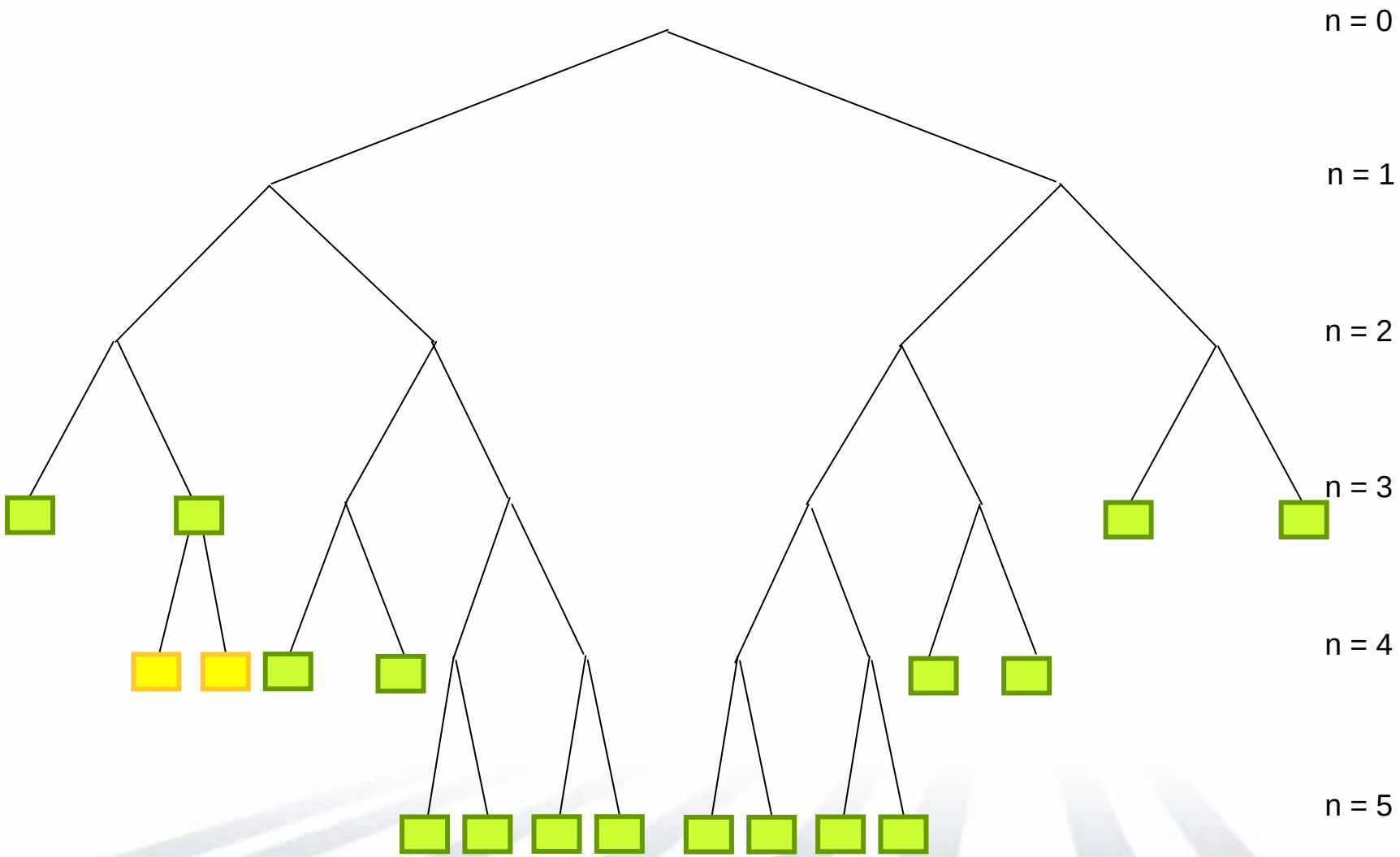
contents adapted from James Dinan, the Ohio State University

Differentiation: I'm too fine



contents adapted from James Dinan, the Ohio State University

Differentiation: I'm too coarse



contents adapted from James Dinan, the Ohio State University

Serial Differentiation Code

```
def diff (n = 0, l = 0, result) {
    if !s.has_coeffs(n, l) {
        // Run down tree until we hit scaling function coefficients

            diff(n+1, 2*l  , result);
            diff(n+1, 2*l+1, result);

    } else {

            var sm = get_coeffs(n, l-1);
            var sp = get_coeffs(n, l+1);
            var s0 = s[n, l];


        // We have s0, check if we found sm and sp at this level
        if !isNone(sm) && !isNone(sp) {
            var r = rp*sm + r0*s0 + rm*sp;
            result.s[n, l] = r * 2.0**n;
        } else {
            recur_down(n, l);

                diff(n+1, 2*l  , result);
                diff(n+1, 2*l+1, result);
}   }   }
```


contents adapted from James Dinan, the Ohio State University

Parallel Differentiation Code

```
def diff (n = 0, l = 0, result) {
    if !s.has_coeffs(n, l) {
        // Run down tree until we hit scaling function coefficients
        cobegin {
            diff(n+1, 2*l  , result);
            diff(n+1, 2*l+1, result);
        }
    } else {
        cobegin {
            var sm = get_coeffs(n, l-1);
            var sp = get_coeffs(n, l+1);
            var s0 = s[n, l];
        }

        // We have s0, check if we found sm and sp at this level
        if !isNone(sm) && !isNone(sp) {
            var r = rp*sm + r0*s0 + rm*sp;
            result.s[n, l] = r * 2.0**n;
        } else {
            recur_down(n, l);
            cobegin {
                diff(n+1, 2*l  , result);
                diff(n+1, 2*l+1, result);
}   }   }   }
```

Perform recursive calls in parallel

Get neighboring coefficients in parallel

Perform recursive calls in parallel

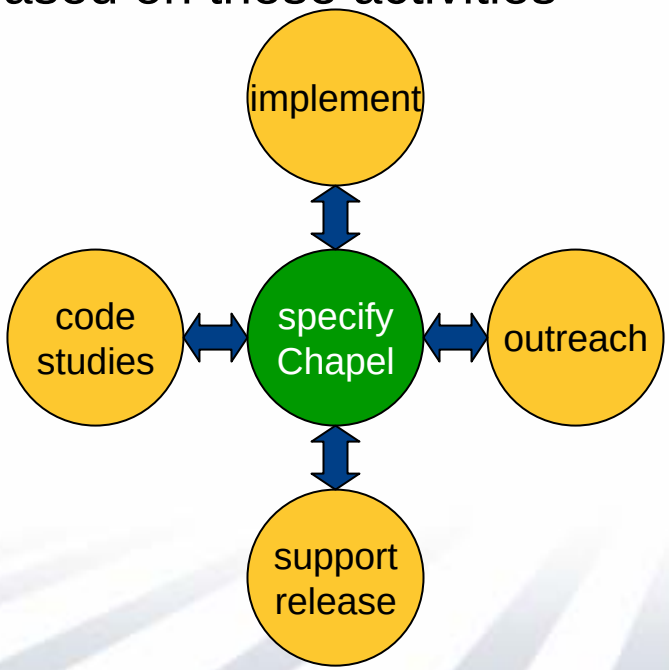
contents adapted from James Dinan, the Ohio State University

Outline

- ✓ Introduction to Chapel
- ✓ Global-View Programming
- ✓ Data Parallel Examples: the Stencil Ramp
- ✓ Task Parallel Features & Examples
- ➢ Status & Summary

Chapel Work

- Chapel Team's Focus:
  - specify Chapel syntax and semantics
  - implement prototype Chapel compiler
  - code studies of benchmarks, applications, and libraries in Chapel
  - community outreach to inform and learn from users, colleagues
  - support users evaluating our preliminary releases
  - refine language based on these activities



Chapel Status

- Draft language specification publicly available
- Portable prototype compiler development progressing
  - Current State:
    - most base language features implemented
    - task parallelism supported within one locale
    - data parallel features implemented, but single-threaded
    - performance competitive with hand-coded C for key 1D kernels
  - Current work/Next steps:
    - support for multi-locale task parallelism
    - parallel implementation of data parallel features
    - continue improving performance for broader set of idioms
- June 2007 release made to ~12 early users at ~8 sites
  - to provide early evaluation and feedback…
    …on language
    …on implementation – usefulness and portability
  - initial response positive, (cautiously) optimistic

Academic Collaborations

- **Vikram Adve & Robert Bocchino (UIUC):** software transactional memory for distributed memory computation

- **Franz Franchetti (CMU):** SPIRAL back-end targeting Chapel to leverage its portability

- **Dan Grossman, Larry Snyder (UW):** co-sponsoring a seminar studying and evaluating Chapel, Fall `07

Chapel Contributors

- **Current:**
  - Brad Chamberlain
  - Steven Deitz
  - Samuel Figueroa
  - Mary Beth Hribar
  - David Iten

- **Alumni:**
  - David Callahan
  - Hans Zima (CalTech/JPL)
  - John Plevyak
  - Wayne Wong
  - Shannon Hoffswell
  - Roxana Diaconescu (CalTech)
  - Mark James (JPL)
  - Mackale Joyner (2005 intern, Rice University)
  - Robert Bocchino (2006 intern, UIUC)
  - James Dinan (2007 intern, OSU)

Summary

- Chapel's goal: "solve the parallel programming problem"
  - general parallel programming
    - data-, task-, nested parallelism
    - general architectures
    - interoperability with other parallel models
  - able to achieve high performance
    - user can tune data and computation for locality
    - user-defined distributions for data aggregates
    - explicit control for task parallelism
    - ability to work at multiple levels of detail within one program
- We have our work cut out for us…

…but what enjoyable work it is!

For More Information…

http://chapel.cs.washington.edu

chapel_info@cray.com

bradc@cray.com