Chapel Background

Brad Chamberlain

PRACE Winter School
12 February 2009

Chapel

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

Themes:

- **general parallel programming**
  - data-, task-, and nested parallelism
  - express general levels of software parallelism
  - target general levels of hardware parallelism
- ***global-view*** **abstractions**
- ***multiresolution*** **design**
- **control of locality**
- **reduce gap between mainstream & parallel languages**

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:

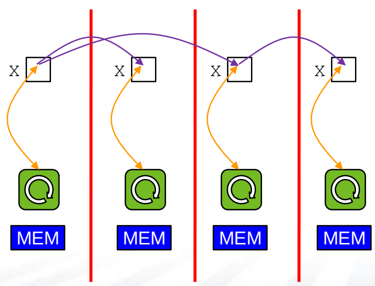
- vastly improve programmability over current languages/models
  - writing parallel codes
  - reading, modifying, porting, tuning, maintaining, evolving them
- support performance at least as good as MPI
  - competitive with MPI on generic clusters
  - better than MPI on more capable architectures
- 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

- Chapel's Themes, Context, and Goals
- the Parallel Language Landscape
  - distributed memory programming
  - shared memory programming
  - PGAS languages
  - HPCS languages
- Programming Model Terminology

Distributed Memory Programming

- **Characteristics:**
  - execute multiple binaries simultaneously & cooperatively
  - each binary has its own local namespace
  - binaries transfer data via communication calls
- **Examples:** MPI, PVM, SHMEM, sockets, …



My Evaluation of MPI

Strengths

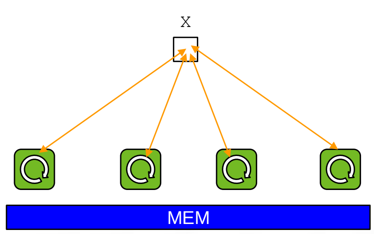
- \+ a very general parallel programming model
- \+ most scientific HPC results in the past decade achieved using it
- \+ it runs on most parallel platforms
- \+ it is relatively easy to implement (or, that's the conventional wisdom)
- \+ for many architectures, it can result in near-optimal performance
- \+ it serves as a strong foundation for higher-level technologies

Weaknesses

- – only supports parallelism at the "cooperating executable" level
  - applications and architectures contain parallelism at many levels
  - doesn't reflect how one abstractly thinks about parallel algorithms
- – encodes too much about "how" data should be transferred rather than simply "what data" (and possibly "when")
  - can mismatch architectures with different data transfer capabilities
- – obfuscates algorithms with many low-level details
  - tedious and error-prone details, arguably best left to the compiler

Shared Memory Programming

- **Characteristics:**
  - execute multiple cooperating threads within one process
  - threads have shared namespace
  - coordinate data accesses via synchronization primitives
- **Examples:** OpenMP, pthreads, Java, …



My Evaluation of OpenMP

Strengths

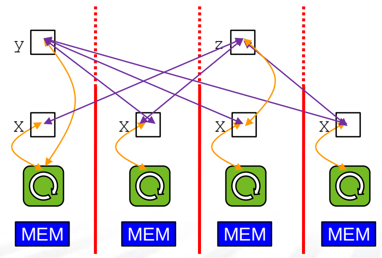
+ supports finer-grain parallelism -- e.g., loop-level
+ can be mixed with other programming models (parallel & sequential)
  - supports existing languages, code bases
  - supports incremental parallelization
+ consortium effort, broad support among vendors

Weaknesses

– shared memory bugs can be notoriously difficult to track down
– no precise control over locality and affinity
  - makes it difficult to scale to large-scale systems

(Traditional) PGAS Programming Models

- **Characteristics:**
  - execute an SPMD program (Single Program, Multiple Data)
  - all binaries share a namespace
    - namespace is partitioned, permitting reasoning about locality
    - binaries also have a local, private namespace
  - compiler introduces communication to satisfy remote references
- **Examples:** UPC, Co-Array Fortran (Fortran 2008), Titanium



My Evaluation of Traditional PGAS Languages

Strengths

+ supports distributed memory architectures
  - particularly ideal for networks with RDMA support
+ raises the level of abstraction compared to MPI
+ nice support for pointer-based data structures across multiple nodes
+ some support for distributed arrays

Weaknesses

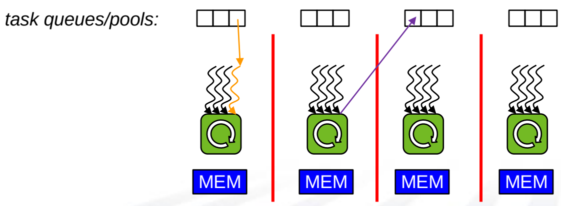
– SPMD programming/execution model is too restrictive
  - algorithms and architectures support parallelism at many levels
– subject to similar synchronization bugs as shared-memory programs
– distributed arrays more restricted than one would ideally like
  - CAF: bookkeeping challenges when local arrays aren't uniform
  - UPC: limited to 1D block-cyclic arrays

PGAS: What's in a Name?

	memory model	programming model	execution model	data structures	communication
MPI	distributed memory	cooperating seq. processes (often SPMD in practice)		manually created distributed arrays	APIs
OpenMP	shared memory	global-view parallelism	shared memory multithreaded	shared memory arrays	N/A
PGAS Languages: CAF	PGAS	Single Program, Multiple Data (SPMD)		co-arrays	co-array refs
PGAS Languages: UPC				1D dist. arrays/ distributed pointers	implicit
PGAS Languages: Titanium				class-based arrays/ distributed pointers	method-based
Chapel	PGAS	global-view parallelism	PGAS multithreaded	global-view distributed arrays	implicit

Asynchronous PGAS (APGAS)

- **Characteristics:**
  - a term coined by IBM's X10 group (to the best of my knowledge)
  - uses the PGAS memory model
  - programming/execution models are richer than SPMD
    - each node can execute multiple tasks/threads
    - nodes can create work for one another
- **Examples:** X10, Chapel, Fortress (?)



X10 in a Nutshell (reflecting my opinions)

- Originally influenced by Java
  - emphasis on type safety, OOP design, small core language
  - also ZPL: support for global-view domains and arrays
- Has since diverged from Java; influenced by Scala, others
- Similar concepts to what you'll hear about today in Chapel
  - yet a fairly different syntax and design aesthetic
- Main differences from Chapel
  - X10 semantics tend to distinguish between local and remote data
  - X10 is a purer object-oriented language
    - for example, arrays have reference rather than value semantics

```
A = B; // alias or copy if A and B are arrays?
```

- For more information:
  - http://www.research.ibm.com/x10/
  - http://x10.codehaus.org/
  - http://sf.net/projects/x10

Fortress in a Nutshell (again, my opinions)

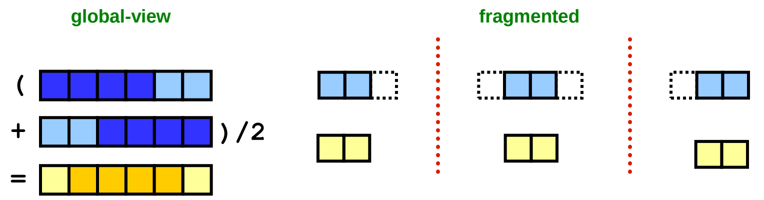
- The most blue-sky, clean-slate of the HPCS languages
- **Goal:** define language semantics in libraries, not compiler:
  - data structures and types (including scalars types?)
  - operators, typecasts
  - operator precedence
  - in short, as much as possible to support future changes, languages
- Other themes:
  - implicitly parallel -- most things are parallel by default
  - supports mathematical notation, symbols, operators
  - functional semantics
  - hierarchical representation of target architecture's structure
  - units of measurement in the type system (meters, seconds, miles, …)
- For more information:
  - http://research.sun.com/projects/plrg/
  - http://projectfortress.sun.com/Projects/Community/

Outline

- Chapel's Themes, Context, and Goals
- the Parallel Language Landscape
- Programming Model Terminology
  - *global-view* vs. *fragmented* programming models
  - *multiresolution languages*
  - a first taste of Chapel

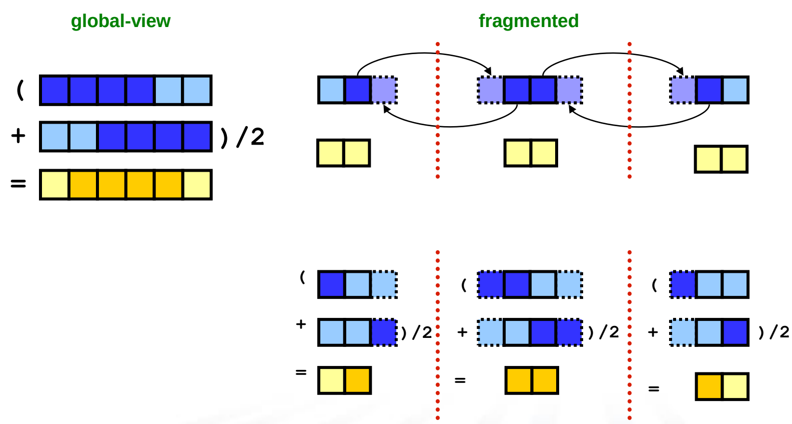
Global-view vs. Fragmented

Problem: "Apply 3-pt stencil to vector"



Global-view vs. Fragmented

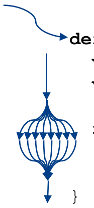
Problem: "Apply 3-pt stencil to vector"



Global-view vs. SPMD Code

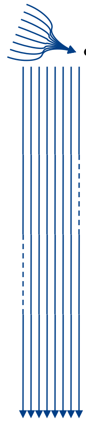
Problem: "Apply 3-pt stencil to vector"

global-view



```
def main() {
  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



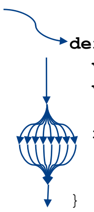
```
def main() {
  var n: int = 1000;
  var locN: int = n/numProcs;
  var a, b: [0..locN+1] real;

  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"

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

global-view



```
def main() {
  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

```
def main() {
  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;
  }
}
```

MPI SPMD pseudo-code

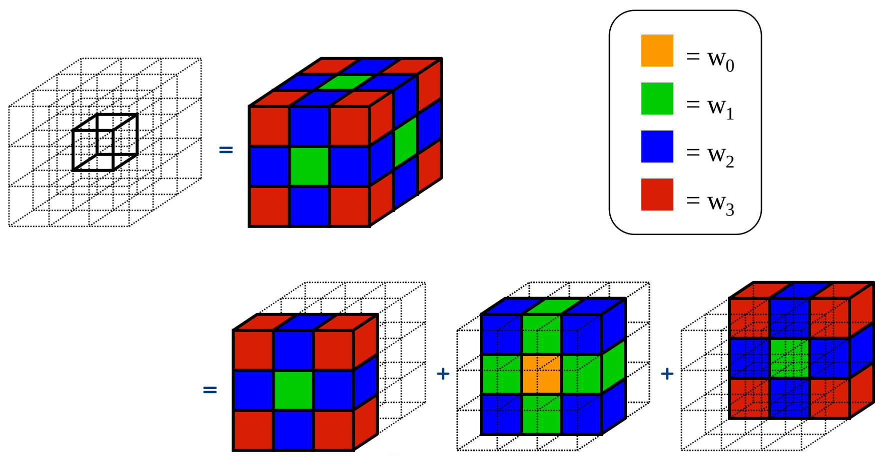
Problem: "Apply 3-pt stencil to vector"

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

rprj3 stencil from NAS MG

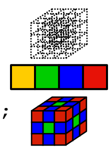


NAS MG *rprj3* stencil in Fortran + MPI

```
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

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

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

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

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

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

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

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


NAS MG *rprj3* stencil 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 ijk in S.domain do
    S(ijk) = + reduce [offset in Stencil]
                      (w3d(offset) * R(ijk + offset*R.stride));
}
```



Our previous work in ZPL showed that compact, global-view codes like these can result in performance that matches or beats hand-coded Fortran+MPI

Summarizing Fragmented/SPMD Models

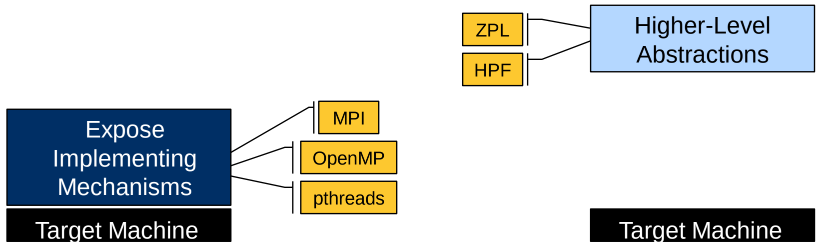
- Advantages:
  - fairly straightforward model of execution
  - relatively easy to implement
  - reasonable performance on commodity architectures
  - portable/ubiquitous
  - lots of important scientific work has been accomplished with them

- 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”

Current HPC Programming Notations

	data / control
communication libraries:	
MPI, MPI-2	fragmented / fragmented/SPMD
SHMEM, ARMCI, GASNet	fragmented / SPMD
shared memory models:	
OpenMP, pthreads	global-view / global-view (trivially)
PGAS languages:	
Co-Array Fortran	fragmented / SPMD
UPC	global-view / SPMD
Titanium	fragmented / SPMD
HPCS languages:	
Chapel	global-view / global-view
X10 (IBM)	global-view / global-view
Fortress (Sun)	global-view / global-view

Parallel Programming Models: Two Camps



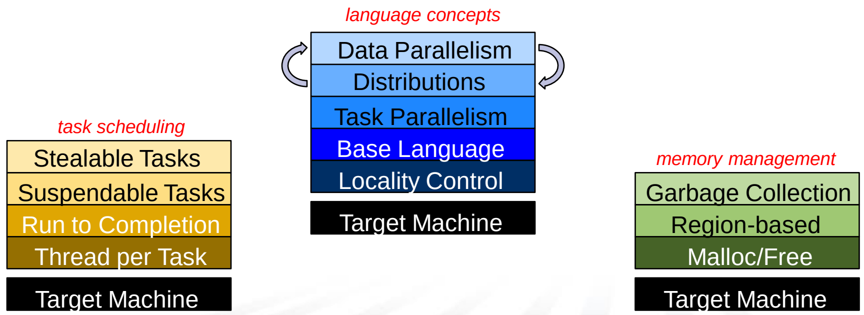
"Why is everything so painful?"

"Why do my hands feel tied?"

Multiresolution Language Design

Our Approach: Permit the language to be utilized at multiple levels, as required by the problem/programmer

- provide high-level features and automation for convenience
- provide the ability to drop down to lower, more manual levels
- use appropriate separation of concerns to keep these layers clean



Questions?