Chapel at the Petascale and on the Desktop

Challenges and Potential

Brad Chamberlain, Cray Inc.

Barcelona Multicore Workshop 2010
October 22, 2010

Five Key Parallel Language Design Decisions

For Multicore, Petascale, and Beyond

Brad Chamberlain, Cray Inc.

Barcelona Multicore Workshop 2010
October 22, 2010

What is Chapel?

- A new parallel language being developed by Cray Inc.
- Part of Cray's entry in the DARPA HPCS program
- **Main Goal:** Improve programmer productivity
  - Improve the programmability of parallel computers
  - Match or beat the performance of current programming models
  - Provide better portability than current programming models
  - Improve robustness of parallel codes
- Target architectures:
  - multicore desktop machines
  - clusters of commodity processors
  - Cray architectures
  - systems from other vendors
- A work in progress, developed as open-source (BSD license)

Chapel's Origins

- **HPCS**: High Productivity Computing Systems
  - Overall goal: Raise high-end user productivity by 10x

    Productivity = Performance + Programmability + Portability + Robustness
- **Phase II:** Cray, IBM, Sun (July 2003 – June 2006)
  - Goal: Propose new productive system architectures
  - Each vendor created a new programming language
    - **Cray:** Chapel
    - **IBM:** X10
    - **Sun:** Fortress
- **Phase III:** Cray, IBM (July 2006 –)
  - Goal: Develop the systems proposed in phase II
  - Each vendor implemented a compiler for their language
    - Sun also continued their Fortress effort without HPCS funding

Outline

- Chapel Background
- Five Parallel Language Design Decisions
  1. Data- vs. Task Parallelism
  2. Global- vs. Local-view Data and Control
  3. High- vs. Low-level Abstractions
  4. Shared- vs. Distributed Memory Model
  5. Locality/Affinity Model
- Next-Generation Nodes: Manycore, GPUs
- Summary
- Possible Bonus: User-defined domain maps

Design Decision 1:
Should a parallel language support data parallelism or task parallelism?

CRAY
THE SUPERCOMPUTER COMPANY

Q1: Data vs. Task Parallelism

Data Parallel: driven by collections of data/indices

- e.g., “for every element in array *A* do the following…”
- notable examples: HPF, ZPL, …

Task Parallel: driven by specifying individual tasks

- e.g., “task 1 should do this while task 2 does that”
- notable examples: Cilk, pthreads, MPI, …

Sub-questions:

What kinds of data parallel structures should be supported?

Can tasks have dependences between one another or not?

Can the parallel concepts be nested?

A1: Data vs. Task Parallelism

Chapel supports a unified set of concepts in order to...

...express any parallelism desired in a user's program

- **Styles:** data-parallel, task-parallel, concurrency, nested, …
- **Levels:** module, function, loop, statement, expression

...target all parallelism available in the hardware

- **Systems:** multicore desktops, clusters, HPC systems, …
- **Levels:** machines, nodes, cores, instructions

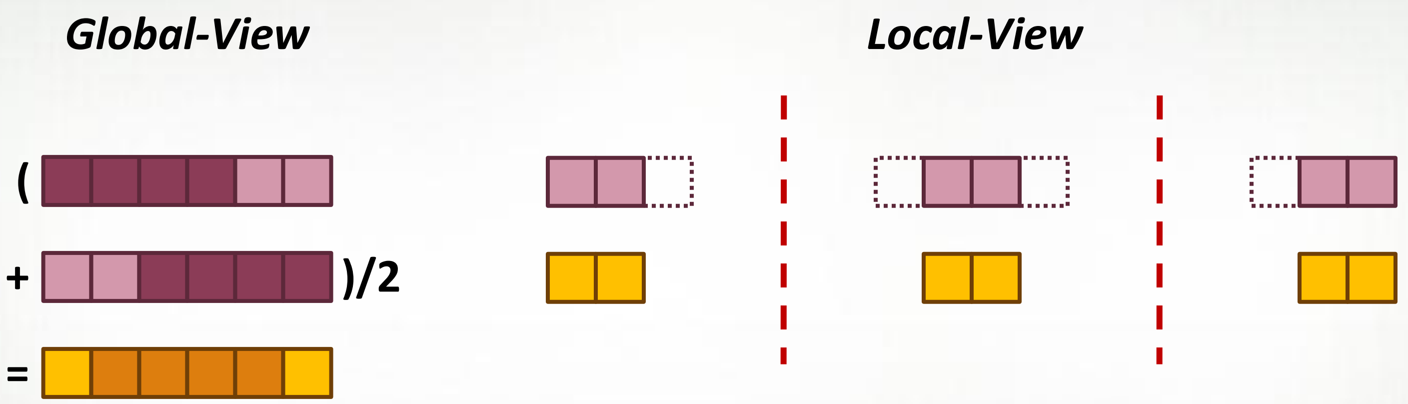
Status quo: most current parallel programming models support only a limited number of styles and system levels, leading to hybrid programming models (e.g., MPI + OpenMP)

Design Decision 2:
Should a parallel language support a global view of data structures and control flow or a local view?

CRAY
THE SUPERCOMPUTER COMPANY

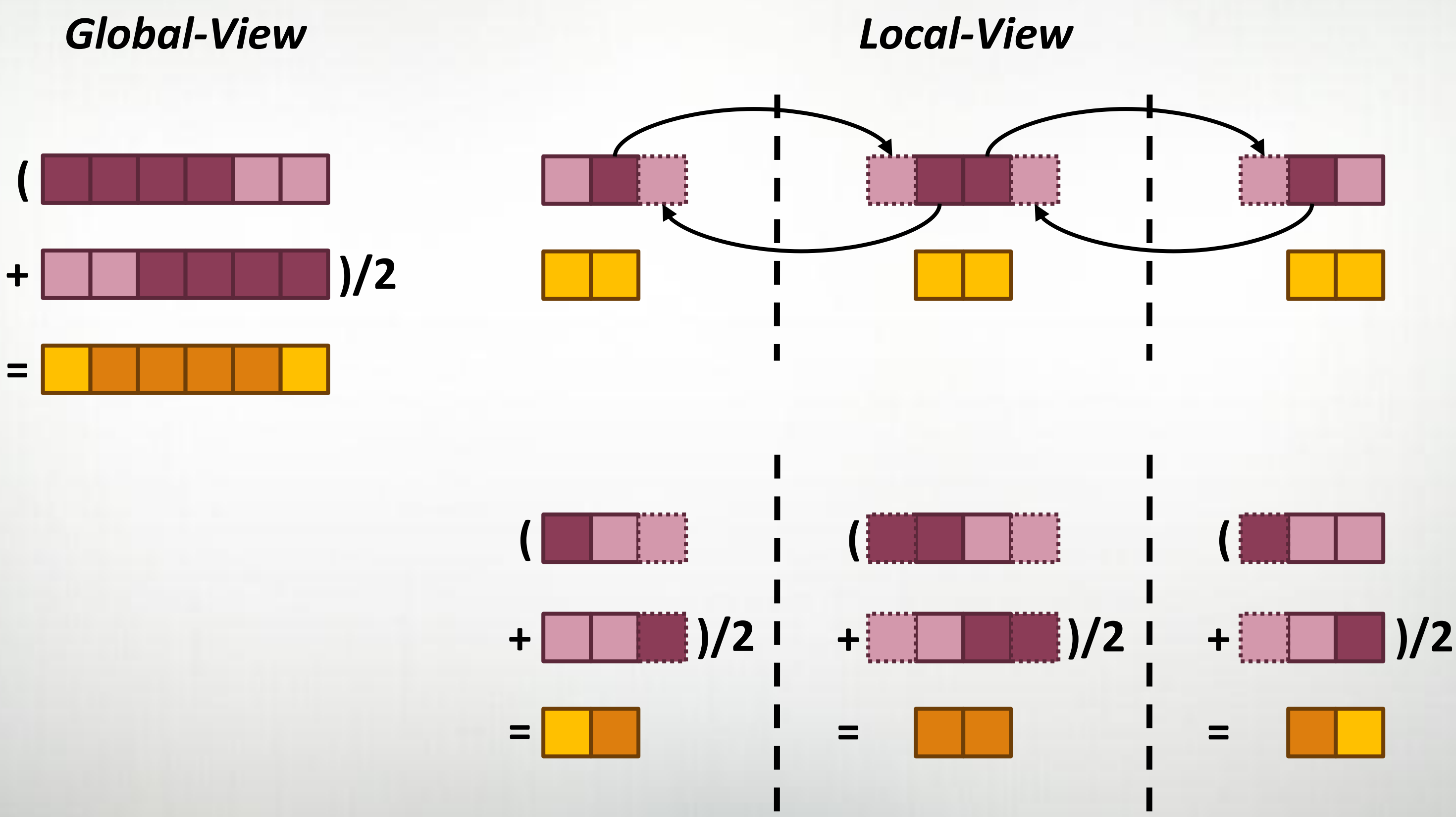
Q2: Global- vs. Local-View Data/Control

In pictures: "Apply a 3-Point Stencil to a vector"



Q2: Global- vs. Local-View Data/Control

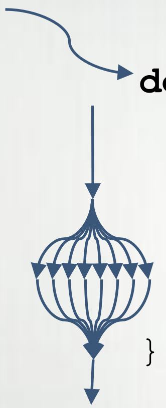
In pictures: "Apply a 3-Point Stencil to a vector"



Q2: Global- vs. Local-View Data/Control

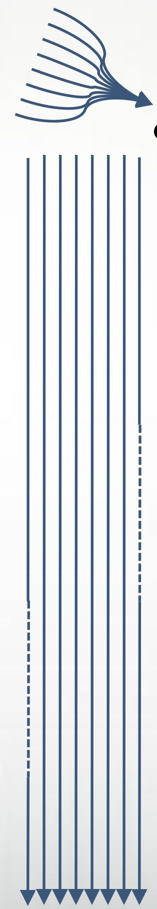
In code: "Apply a 3-Point Stencil to a vector"

Global-View



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

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

Local-View (SPMD)



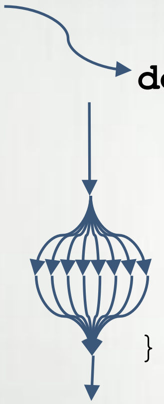
```
def main() {
  var n = 1000;
  var p = numProcs(),
      me = myProc(),
      myN = n/p,
  var A, B: [0..myN+1] real;

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

Bug: Refers to uninitialized values at ends of A

Q2: Global- vs. Local-View Data/Control

In code: "Apply a 3-Point Stencil to a vector"

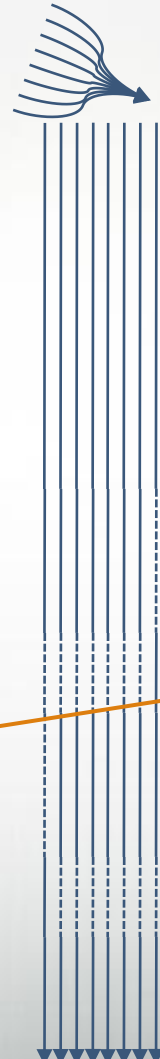
Global-View



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

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

Communication becomes geometrically more complex for higher-dimensional arrays

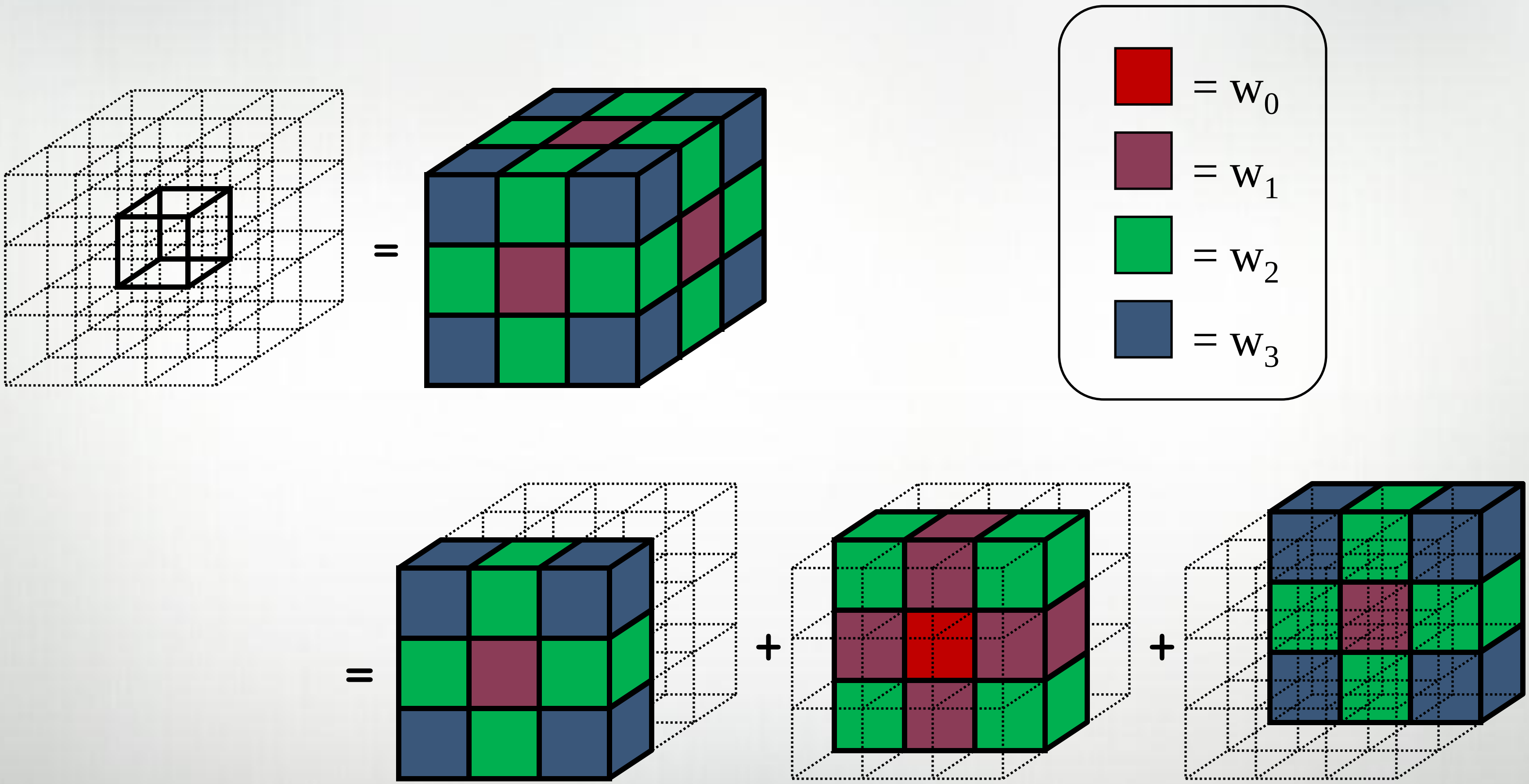
Local-View (SPMD)



```
def main() {
  var n = 1000;
  var p = numProcs(),
      me = myProc(),
      myN = n/p,
      iLo = 1,
      iHi = myN;
  var A, B: [0..myN+1] real;

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

Assumes p divides n

rprj3 Stencil from NAS MG

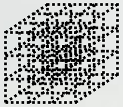


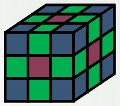
Local-view *rprj3* Stencil (Fortran + MPI)

```fortran
      subroutine rprj3( r,m1k,m2k,m3k,s,m1j,m2j,m3j,k )

      implicit none

      include 'mpinpb.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)

      if( debug_vec(0) .ge. 1 )then
         call rep_nrm(s,m1j,m2j,m3j,'   rprj3',k-1)
      endif

      if( debug_vec(4) .ge. k )then
         call showall(s,m1j,m2j,m3j)
      endif

      return
      end


      subroutine norm2u3(r,n1,n2,n3,rnm2,rnmu,nx,ny,nz)

      implicit none

      include 'mpinpb.h'

      integer n1, n2, n3, nx, ny, nz
      double precision rnm2, rnmu, r(n1,n2,n3)
      double precision s, a, ss
      integer i3, i2, i1, ierr

      double precision dn

      dn = 1.0d0*nx*ny*nz

      s=0.0D0
      rnmu = 0.0D0
      do  i3=2,n3-1
         do  i2=2,n2-1
            do  i1=2,n1-1
               s=s+r(i1,i2,i3)**2
               a=abs(r(i1,i2,i3))
               if(a.gt.rnmu)rnmu=a
            enddo
         enddo
      enddo

      call mpi_allreduce(rnmu,ss,1,dp_type,
     >     mpi_max,mpi_comm_world,ierr)
      rnmu = ss
      call mpi_allreduce(s, ss, 1, dp_type,
     >     mpi_sum,mpi_comm_world,ierr)
      s = ss
      rnm2=sqrt( s / dn )

      return
      end


      subroutine rep_nrm(u,n1,n2,n3,title,kk)

      implicit none

      include 'mpinpb.h'
      include 'globals.h'

      integer n1, n2, n3, kk
      double precision u(n1,n2,n3)
      character*8 title

      double precision rnm2, rnmu

      call norm2u3(u,n1,n2,n3,rnm2,rnmu,nx(kk),ny(kk),nz(kk))
      if( me .eq. root )then
         write(*,7)kk,title,rnm2,rnmu
 7       format(' Level',i2,' in ',a8,': norms =',D21.14,D21.14)
      endif

      return
      end


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

      implicit none

      include 'mpinpb.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 ready( axis, -1, kk )
               call ready( axis, +1, kk )

               call give3( axis, +1, u, n1, n2, n3, kk )
               call give3( axis, -1, u, n1, n2, n3, kk )

               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
         call zero3(u,n1,n2,n3)
      endif
      return
      end


      subroutine ready( axis, dir, k )

      implicit none

      include 'mpinpb.h'
      include 'globals.h'

      integer axis, dir, k
      integer buff_id,buff_len,i,ierr

      buff_id = 3 + dir
      buff_len = nm2

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

      msg_id(axis,dir,1) = msg_type(axis,dir) +1000*me

      call mpi_irecv( buff(1,buff_id), buff_len,
     >     dp_type, nbr(axis,-dir,k), msg_type(axis,dir),
     >     mpi_comm_world, msg_id(axis,dir,1), ierr)
      return
      end


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

      implicit none

      include 'mpinpb.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

            call mpi_send(
     >           buff(1, buff_id ), buff_len,dp_type,
     >           nbr( axis, dir, k ), msg_type(axis,dir),
     >           mpi_comm_world, ierr)

         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

            call mpi_send(
     >           buff(1, buff_id ), buff_len,dp_type,
     >           nbr( axis, dir, k ), msg_type(axis,dir),
     >           mpi_comm_world, ierr)

         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

            call mpi_send(
     >           buff(1, buff_id ), buff_len,dp_type,
     >           nbr( axis, dir, k ), msg_type(axis,dir),
     >           mpi_comm_world, ierr)

         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

            call mpi_send(
     >           buff(1, buff_id ), buff_len,dp_type,
     >           nbr( axis, dir, k ), msg_type(axis,dir),
     >           mpi_comm_world, ierr)

         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

            call mpi_send(
     >           buff(1, buff_id ), buff_len,dp_type,
     >           nbr( axis, dir, k ), msg_type(axis,dir),
     >           mpi_comm_world, ierr)

         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

            call mpi_send(
     >           buff(1, buff_id ), buff_len,dp_type,
     >           nbr( axis, dir, k ), msg_type(axis,dir),
     >           mpi_comm_world, ierr)

         endif
      endif

      return
      end


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

      implicit none

      include 'mpinpb.h'
      include 'globals.h'

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

      integer buff_id, indx

      integer status(mpi_status_size), ierr

      integer i3, i2, i1

      call mpi_wait( msg_id( axis, dir, 1 ),status,ierr)
      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 )

      implicit none

      include 'mpinpb.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 showall(z,n1,n2,n3)

      implicit none

      include 'mpinpb.h'

      integer n1,n2,n3,i1,i2,i3,i,ierr
      double precision z(n1,n2,n3)
      integer m1, m2, m3

      m1 = min(n1,18)
      m2 = min(n2,14)
      m3 = min(n3,18)

      write(*,*)'  '
      do  i=0,nprocs-1
         if( me .eq. i )then
            write(*,*)' id = ', me
            do  i3=1,m3
               do  i1=1,m1
                  write(*,6)(z(i1,i2,i3),i2=1,m2)
               enddo
               write(*,*)' - - - - - - - '
            enddo
            write(*,*)'  '
 6          format(15f6.3)
         endif
         call mpi_barrier(mpi_comm_world,ierr)
      enddo

      return
      end


      subroutine zero3(z,n1,n2,n3)

      implicit none

      include 'mpinpb.h'

      integer n1, n2, n3
      double precision z(n1,n2,n3)
      integer i1, i2, i3

      do  i3=1,n3
         do  i2=1,n2
            do  i1=1,n1
               z(i1,i2,i3)=0.0D0
            enddo
         enddo
      enddo

      return
      end
bradc-lnx%
```

Global-view *rprj3* Stencil (in Chapel)

```
def rprj3(S: [?SD], R: [?RD]) {
  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 SD do
    S[ijk] = + reduce [offset in Stencil]
                      (W3D[offset] * R[ijk + RD.stride*offset]);
}
```





Our previous work in ZPL demonstrated that such compact codes can result in better performance than Fortran + MPI while also supporting more flexibility at runtime.*

***specifically, the Fortran + MPI *rprj3* code shown previously assumes that *p* and *n* are both specified at compile-time and powers of two.**

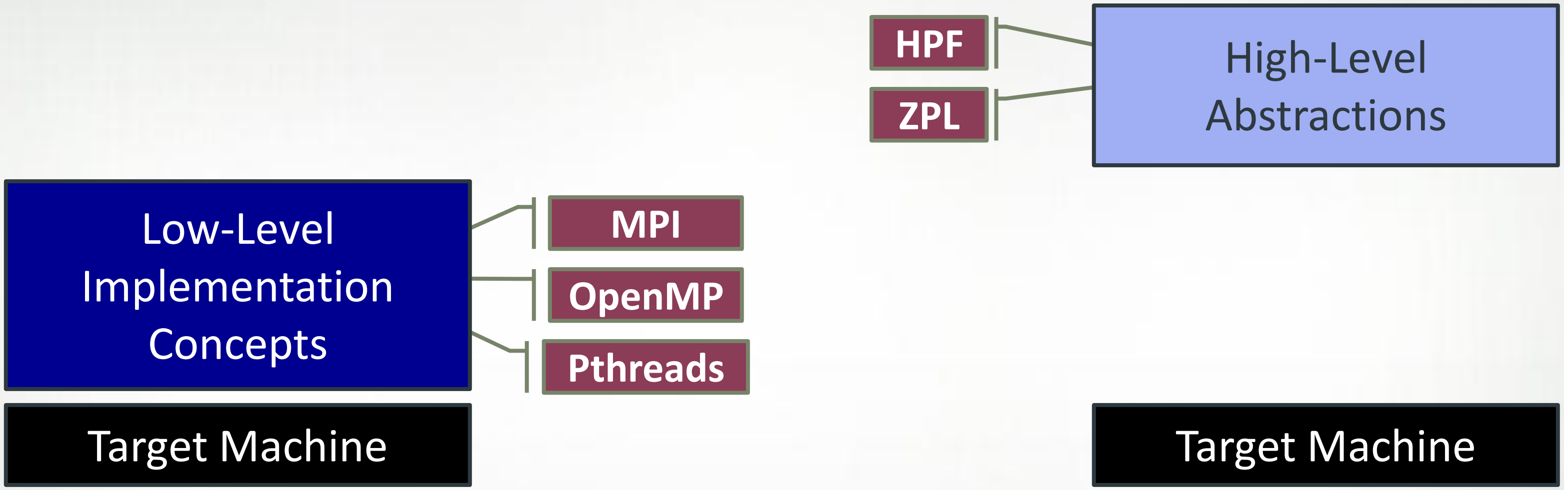
A2: Global- *and* Local-View Programming

- This choice is not exclusive: A language can support both global and local views, and we believe it should
- In particular, Chapel does:

```
def main() {
  coforall loc in Locales do
    on loc do
      MySPMDProgram(loc.id, Locales.numElements);
}

def MySPMDProgram(me, p) {
  ...
}
```

Design Decision 3:
What level of abstraction should a parallel language support?

Q3: High- vs. Low-level Abstractions



"Why is everything so difficult?"
"Why don't my programs port trivially?"

"Why don't I have more control?"

Q3: High- vs. Low-level Abstractions

Low-level / Control-oriented: closer to the machine

- *e.g.*, C, MPI, OpenMP, CUDA, …

\+ general; good performance control

\+ easier to implement

\- tend to require more user effort to program

\- more brittle w.r.t. architectural changes

  - *e.g.*, MPI works for clusters, but is inadequate for GPUs

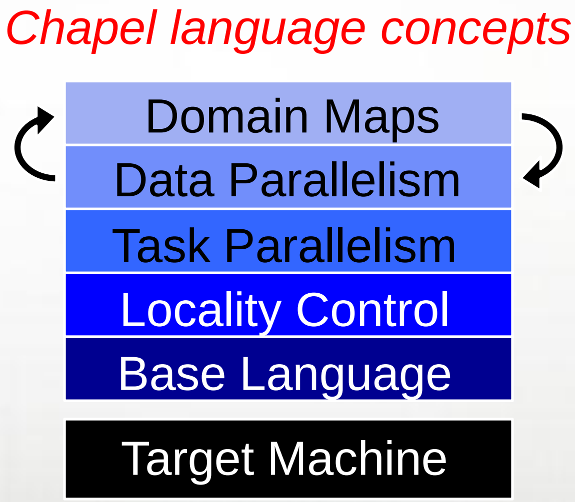
High-level / Programmability-oriented: more abstract, hides details

- *e.g.*, ZPL, HPF, NESL, …
- reverse benefits/liabilities from above

A3: Multiresolution Language Design

Multiresolution Languages: Layered, multi-tiered design

- higher levels for programmability, productivity
- lower levels for performance, control
- higher-level concepts built in terms of the lower



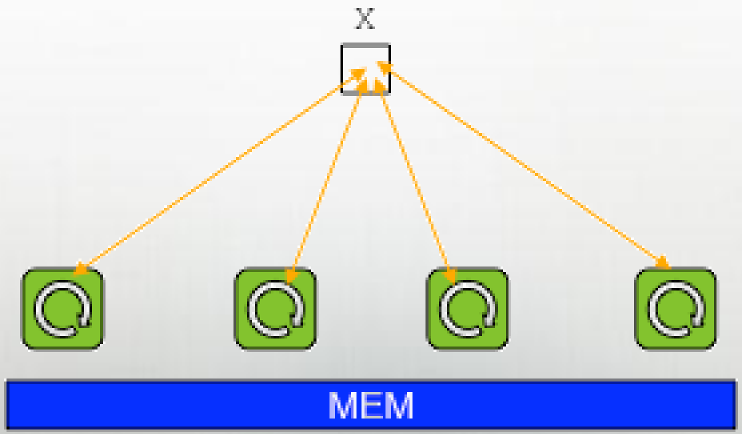
- typically a bigger language, though with good design, not necessarily a kitchen sink

Design Decision 4:
Should a parallel language support a shared-memory or distributed-memory view of data?

Q4: Shared- vs. Distributed Memory Model

Shared Memory

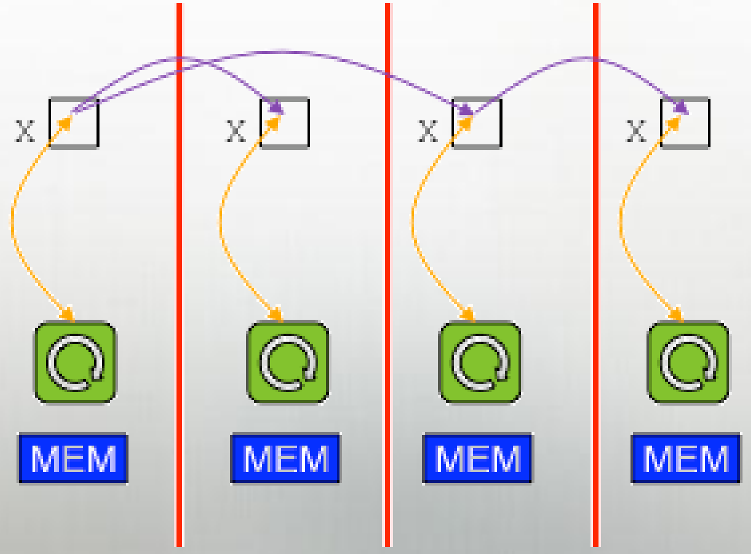
+ considered simpler, more like traditional programming
    - "if you want to access something, simply name it"
- no support for expressing locality/affinity; limits scalability
- bugs can be subtle, difficult to track down (race conditions)
- tend to require complex memory consistency models



Q4: Shared- vs. Distributed Memory Model

Distributed Memory

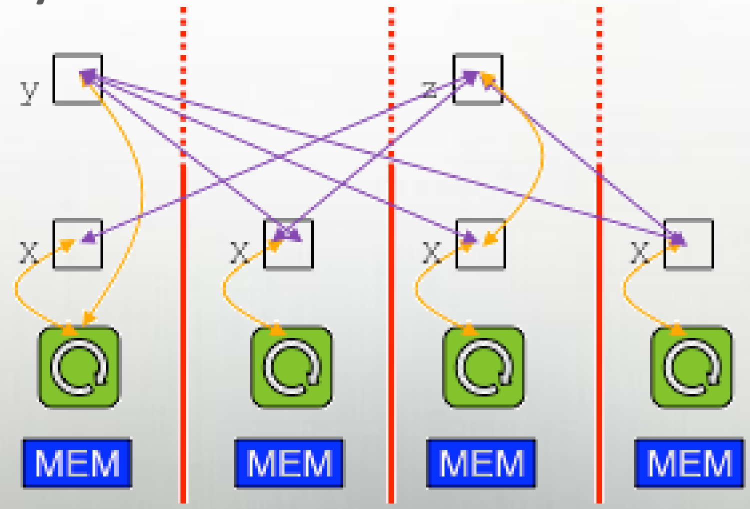
+ a more constrained model; you can only access local data
- communication must be used to get copies of remote data
- only supports coarse-grain task parallelism
- intermixes semantics of data transfer with synchronization
- has frustrating classes of bugs of its own
  - e.g., recvs without matching sends, buffer overflows, etc.



A4: PGAS Memory Model

PGAS: Partitioned Global Address Space

- supports a shared namespace, like shared-memory
- supports a strong sense of ownership and locality
  - each variable is stored in a particular memory segment
  - tasks can access any visible variable, local or remote
  - local variables are cheaper to access than remote ones
- retains many of the downsides of shared-memory



Design Decision 5:
How should a parallel programming language support the user's ability to reason about locality/affinity?

THE SUPERCOMPUTER COMPANY

Q5: Locality/Affinity Model (w.r.t. Parallelism)

locality-oblivious: model has no real notion of locality

- (see shared-memory bullet from previous question)

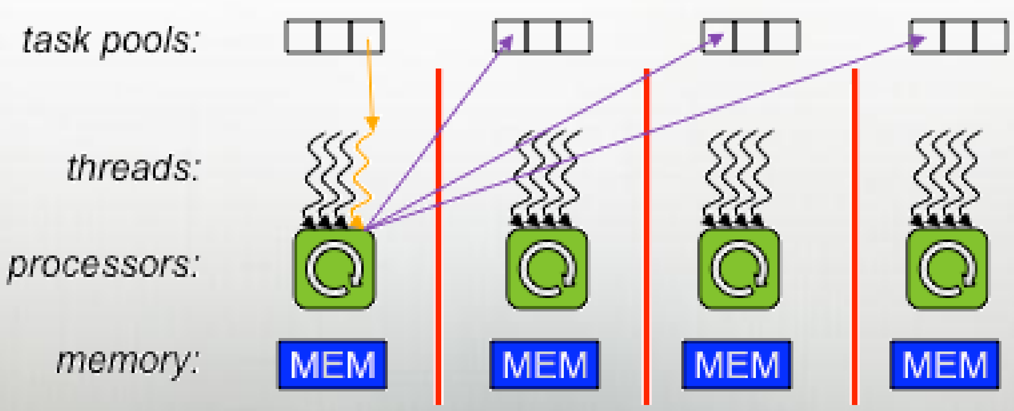
locality-constrained: locality and parallelism are expressed using the same concept

- *e.g.*, MPI ranks serve as both the unit of locality and parallelism
- implications for utilizing multicore processors:
  - programmer has to use a hybrid model
  - or has to ignore locality within a node
  - or work outside of the abstract programming model

A5: Distinct Concepts for Parallelism vs. Locality

Characteristics:

- Chapel has distinct concepts for parallelism vs. locality
  - *task:* unit of parallel work that supports concurrent execution
  - *locale:* region of target architecture with processors and memory
- resulting programming/execution model richer than SPMD
  - each locale can execute multiple tasks
  - tasks can create work for any locale
  - a more appropriate model for multicore



Summary: Design Decisions and Chapel's Answers

1. Data- vs. Task Parallelism?
   - support both (and composition) for the sake of generality
2. Global- vs. Local-view Data and Control?
   - support both: global- for productivity, local- for control
3. High- vs. Low-level Abstractions?
   - use a multiresolution design to get the best of both worlds
4. Shared- vs. Distributed Memory Model?
   - PGAS supports shared memory advantages with scalability
5. Locality/Affinity Model?
   - use distinct concepts for parallelism vs. locality

Where do your current parallel programming models fall?

Outline

- Chapel Background
- Five Parallel Language Design Decisions
- Next-Generation Nodes: Manycore, GPUs
- Summary

Processor Architecture Trends

Expected Processor Trends:

- multicore -> manycore
- increasing use of accelerators (e.g., GPGPUs)

Impacts on Programming Model:

- growing need to pay attention to locality within a node
  - desktop parallel programming will increasingly resemble cluster
  - HPC parallel programming will only become more complex
- growing need to deal with heterogeneity
  - different processor types/capabilities/limitations
  - different memory types/properties

We believe that Chapel is well-positioned for these challenges given the choices described earlier

Next-Generation Nodes and Design Decisions

1. Data- vs. Task Parallelism?
   - task- to launch asynchronous computations
   - data- to leverage SIMD computation units
2. Global- vs. Local-view Data and Control?
3. High- vs. Low-level Abstractions?
   - HW will be complex enough that the value of high-level global-view abstractions will only grow
   - yet desire for lower-level control will always remain
4. Shared- vs. Distributed Memory Model?
   - shared memory doesn't match hierarchy/heterogeneity
   - yet distributed memory feels like overkill for an accelerator
5. Locality/Affinity Model?
   - will only become more important given trends

Summary

Through Chapel's design choices…

- general forms of composable parallelism
- global- and local-view programming
- multiresolution design
- PGAS memory model
- distinct concepts for locality and parallelism

…we believe it is well-positioned for productive desktop/petascale parallel programming today

…and for the desktop/exascale machines of tomorrow where these decisions become more important

Current/Future Work

- Generalize Locale Concept to Support Hierarchies
  - single level of locality was sufficient for petascale
  - next-generation nodes will require more
- Domain Maps for Next-generation Nodes
  - to support global-view arrays on accelerators, e.g.
- Performance Improvements
  - communication optimizations
  - loop nest idioms

For More Information

- http://chapel.cray.com: papers, presentations, language specification, and other general information
- https://sourceforge.net/projects/chapel: download Chapel and view/contribute to its development
- chapel_info@cray.com: for general questions to the team (SourceForge-based mailing lists also exist)
- Attend our SC10 Tutorial, Monday November 15th

Questions?

THE SUPERCOMPUTER COMPANY