An HPF Binding for MPI
DRAFT

Ian Foster

October 30, 1995

1 Motivation

HPF extends F90 with syntax for specifying data-parallel execution within a single logical thread of control, or task; the Message Passing Interface (MPI) defines functions for specifying communication between multiple tasks. An HPF binding for MPI ("HPF/MPI") would provide a standard set of functions for coupling multiple HPF tasks to form task-parallel computations. This document uses two example programs to show what such a binding would look like and how it could be used. It also discusses some relevant interface and implementation issues. The latter discussion builds on experience with a prototype HPF/MPI implementation developed with Rakesh Krishnaiyer and Alok Choudhary of Syracuse and David Kohr at Argonne, using the PGI HPF compiler as a base.

2 Using an HPF Binding for MPI

An HPF binding for MPI works as follows. A programmer initiating a computation requests (using some implementation-dependent mechanism) that a certain number of tasks be created; each task executes a specified HPF program on a specified number of processors. Tasks can call MPI functions to exchange data with other tasks, using either point-to-point or collective communication operations. When reading programs, HPF directives can be ignored, and code understood as if it implements a set of sequential tasks that communicate using MPI functions.

We illustrate this basic model by presenting HPF/MPI implementations of the 2-D FFT and 2-block Poisson solver task-parallel benchmarks proposed by Jaspal Subhlok and Scott Baden, respectively. The code in Figure 1 implements the pipelined 2-D FFT. As illustrated in Figure 2, this program is designed to be executed by two tasks. The first task performs 1-D FFTs on each column of a 2-D array; the modified array is then communicated to the second task which performs 1-D FFTs on each row. As both tasks execute the same program, a runtime test is used to determine which code is executed in each task. An alternative structure would provide two distinct programs, one containing the code executed by task 0 and the other the code executed by task 1. This structure would use half as much storage, as arrays `a` and `b` would not be replicated in the two tasks.

The code in Figure 3 implements the two-block Poisson solver, which also executes as two tasks. The tasks exchange boundary data (using `MPI_Sendrecv`, run a solver on their block, and compute a global error (using `MPI_Reduce`).

```
      program twodfft
      include 'mpihpf.h'
      integer nprocs, myid, ierr, status(MPI_STATUS_SIZE)
      integer i, j, k, ipow, N, LN, NITER
      parameter (N=256, NITER=100)
      complex a(N,N), b(N,N)
!HPF$ processors pr(Number_Of_Processors())
!HPF$ distribute a(*,BLOCK), b(BLOCK,*)

!     Initialize MPI and check that we have 2 tasks
      call MPI_Init(ierr)
      call MPI_Comm_size(MPI_COMM_WORLD, nprocs, ierr)
      if (nprocs .ne. 2) then
        print *,"2DFFT requires 2 tasks"
        stop 99
      endif

!     Determine which task am I: task 0 or task 1
      call MPI_Comm_rank(MPI_COMM_WORLD, myid, ierr)

      do k = 1, NITER
        if (myid .eq. 0) then   ! I am task 0: send to task 1
          forall(i=1:N, j=1:N) a(i,j) = (1.0,0.0)
          call colfft(N,a)
          call MPI_Send(a,N*N,MPI_COMPLEX,1,99,MPI_COMM_WORLD,ierr)
        else                    ! I am task 1: receive from task 0
          call MPI_Recv(b,N*N,MPI_COMPLEX,0,99,MPI_COMM_WORLD,
     $                 status,ierr)
          call rowfft(N,b)
        endif
      end do

!     Shut down MPI
      call MPI_Finalize(ierr)
      end
```

Figure 1: HPF/MPI Implementation of 2-D FFT

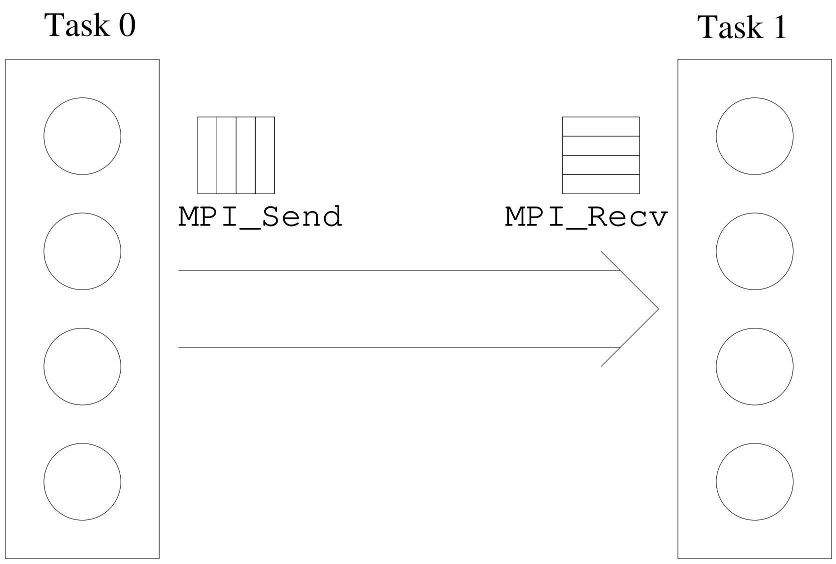

Figure 2: Operation of the HPF/MPI 2-D FFT code, in the case where four processors are allocated to each of the two tasks. The first task, which performs 1-D FFTs on each column of the array `a` (distributed by column) performs `MPI_Send` operations on this array. The second task performs `MPI_Recv` operations into the array `b`, which is distributed by row.

3 Interface Issues

The following are some of the many issues that must be addressed in defining an HPF binding for MPI. Most are also F90 binding issues. (Thanks to Chuck Koelbel and Bill Saphir for discussions on these topics.)

1. An HPF binding for MPI should presumably be identical to (or perhaps a slight generalization of) an F90 binding for MPI. Unfortunately an F90 binding has not yet been defined. The existing F77 binding is not adequate for F90 features such as array subsections and derived data types. The lack of sequence association in F90 also complicates things.

2. It would be nice to define an F90 binding as a module, however this is problematic because F90 interface blocks cannot specify an equivalent to the "void *" used in many MPI function interfaces.

3. The passing of array sections to the MPI asynchronous communication routines (Isend, Irecv, etc) can be problematic because an F90 compiler may copy on call and return. (In fact, a compiler is allowed to copy even full arrays, though no compilers do.) A clear description of what MPI assumes is required.

4. Asynchronous operations can also be problematic if an HPF compiler reorders statements so that a variable modified by an asynchronous receive is read before it is set. For example, if `y = x` is hoisted in the following:

```
call MPI_Irecv(x, N, MPI_REAL, ..., request)
...
```

```
      program block2
      include 'mpihpf.h'
!HPF$ processors pr(NUMBER_OF_PROCESSORS())
      integer, parameter :: n1=256, n2=128, MaxIter=100
      real, dimension(0:n1+1, 0:n1+1)     :: A, At
      real, dimension(n1:n1+n2+1, 0:n2+1) :: B, Bt
!HPF$ distribute A(*, BLOCK) onto pr
!HPF$ align At(i, j) with A(i, j)
!HPF$ distribute B(*, BLOCK) onto pr
!HPF$ align Bt(i, j) with B(i, j)
      intrinsic maxval
      double precision maxA, maxB, maxAB,  maxval
      integer nprocs, myid, ierr, status(MPI_STATUS_SIZE)

      call MPI_Init(ierr)
      call MPI_Comm_rank(MPI_COMM_WORLD, myid, ierr)

      if (myid .eq. 0) then
        A = 0.0
        A(1:n1, 1:n1) = 1.0
      else
        B = 0.0
        B(n1+1:n1+n2, 1:n2) = 1.0
      endif
      n1n2 = min(n1,n2)

      do iter = 1, Maxiter
        if (myid .eq. 0) then   ! Code executed by task 0
          call MPI_Sendrecv(A(n1,1:n1n2),   n1n2, MPI_REAL, 1, 99,
    &                       A(n1+1,1:n1n2), n1n2, MPI_REAL, 0, 99,
    &                       MPI_COMM_WORLD, status, ierr)
          At = A
          call Smooth(A, 1, n1, 1, n1)
          maxA = maxval(abs(A-At))
          call MPI_Reduce(maxA, maxAB, 0, MPI_REAL, MPI_MAX,
    &                     MPI_COMM_WORLD, ierr)
          print *, "Error = ",maxAB
        else                    ! Code executed by task 1
          call MPI_Sendrecv(B(n1+1,1:n1n2), n1n2, MPI_REAL, 0, 99,
    &                       B(n1,1:n1n2),   n1n2, MPI_REAL, 1, 99,
    &                       MPI_COMM_WORLD, status, ierr)
          Bt = B
          call Smooth(B, n1+1, n1+n2, 1, n2)
          maxB = maxval(abs(B-Bt))
          call MPI_Reduce(maxB, maxAB, 0, MPI_REAL, MPI_MAX,
    &                     MPI_COMM_WORLD, ierr)
        endif
      end do

      call MPI_Finalize(ierr)
      end
```

Figure 3: HPF/MPI Implementation of Two-block Poisson Solver

```
call MPI_Wait(request)
y = x
```

This is a potential problem in the C or F77 bindings too, but an HPF compiler has more incentive to reorder code. We may require a directive of the form "this variable can change in this region."

5. The environment in which an HPF extrinsic executes needs to be defined more precisely than at present. If using MPI for low-level communication, extrinsics no longer execute in `MPI_COMM_WORLD`. We could define an `HPF_COMM_WORLD`, but probably a better approach is to define an enquiry function that returns the appropriate communicator. (These functions could also return information about the tasks that comprise the HPF/MPI computation.)

6. Note that as MPI-2 defines mechanisms for dynamic process management and remote data access, these same mechanisms can be incorporated into an HPF binding for MPI.

4 Implementation Issues

We outline the techniques that might be used to implement an HPF binding for MPI, and highlight problematic issues.

1. An HPF implementation that is to support HPF/MPI must be modified to create multiple tasks and initialize the data structures (e.g., MPI communicators) used for intratask and intertask communications. Note that if an HPF implementation uses MPI as its underlying communication layer, then intratask communication will now use a communicator other than `MPI_COMM_WORLD`.

2. HPF/MPI functions will be implemented by a library invoked via HPF's extrinsic function interface. This library must determine how the data to be communicated are distributed, and then uses low-level communication mechanisms to transfer data. Data distribution information can be obtained via enquiry functions such as `HPF_Distribution` (prior to calling an extrinsic) or by using HPF's local routine library (within the extrinsic).

3. The data transfer associated with an `MPI_Send/MPI_Recv` pair can be performed in a variety of ways, with different performance tradeoffs. A simple strategy is to gather the data on the sending side (e.g., at sending processor 0), transfer it to the receiving side in a single message (e.g., to receiving processor 0), and then perform a scatter operation to redistribute it. Alternatively, the sending and receiving processors can first exchange data distribution information and then perform the data transfer directly, without the gather/scatter. In the latter case, algorithms developed for array redistribution can be used to compute an efficient communication schedule.

4. A programmer can use MPI's persistent send and receive operations (`MPI_Send_init`, etc.) to indicate that the same communication operation is to be performed many times (e.g., in a pipeline). An HPF/MPI implementation of these operations can improve performance by caching and reusing a communication schedule. (A difficulty is that MPI's persistent communication operations do not define a full channel. An HPF binding for MPI may motivate an MPI-2 proposal for full channel support.)