Errata for 'Using MPI-2'

July 10, 2005

p 25 This is not an errata but is a clarification. The question is about the choice of `bufsize` as `filesize/numprocs+1` rather than `filesize/numprocs`. The clarification is:

> The reason for using `bufsize = filesize/numprocs + 1` is for the case where `filesize < numprocs`. If the "`+1`" is left out, then all processes read zero elements. This way, the whole file will be read. The cost is that if `filesize` is evenly divided by `numprocs`, a less than optimal number of elements is read by each process.

Thanks to Chieh-Sen Huang <huangcs@math.nsysu.edu.tw>.

p 27 The line

```
theFile.Read(buf, bufsize, MPI_INT, &status );
```

should read

```
theFile.Read(buf, bufsize, MPI_INT, status );
```

p 31 The first sentence after the pair of `MPI_Win_create` calls reads

> over the communicator specified in its last argument

but should read

> over the communicator specified in its second-to-last argument

since the MPI Window object is returned in the last argument.

Thanks to Brad Penoff.

Thanks to Jeff Squyres <squyres@cse.nd.edu>.

p 52,64,66,76,156 The figures on these pages are missing shading within some of the rectangles. Only Figure 3.4 on page 62 must be replaced to be understood.

p 62 Figure 3.4 is missing the "shaded portion" that is mentioned in the preceeding paragraph. Postscript for this figure is available at `http://www.mcs.anl.gov/mpi/usingmpi2/view.eps`.

Thanks to Takao Hatazaki.

p 73 In Figure 3.11, `MPI_File_open` uses `MPI_COMM_WORLD`, but it should use the communicator that was returned from `MPI_Cart_create`.

Thanks to Takao Hatazaki.

p 76 In Figure 3.12, the ghost cell area should be shaded.

Thanks to Takao Hatazaki.

p 77 In Figure 3.13, `MPI_File_open` uses `MPI_COMM_WORLD`, but it should use the communicator that was returned from `MPI_Cart_create`.

Thanks to Takao Hatazaki.

p 117 In Figure 3.34, `MPI_Dims_create(nprocs,...)` should be called after calling `MPI_Comm_size(...,&nprocs)`, not before.

Thanks to Takao Hatazaki.

p 127 On the 3rd line from the bottom, "We are tempted to say that the sum of the values of j printed by the two processes ..." should say "threads" instead of "processes". Processes isn't wrong, but the rest of the text is talking about threads.

Thanks to Takao Hatazaki.

p 128,129,131 In the figure 4.14, 4.16, and 4.17, `print` should be replaced with `printf`.

Thanks to Takao Hatazaki.

p 129 In Figure 4.16, add at the top

```
volatile int i;
...
```

to indicate that `i` must also be declared `volatile`.

Thanks to Brian Toonen <toonen@mcs.anl.gov>.

p 131 Replace Figure 4.17 with

```
volatile int i = 0;
int j = 0;
while (i < 10) {
    lock();
    if (i < 10) {
        i = i + 1;
```

```
        j = j + 1;
    }
    unlock();
}
printf( "j = %d\n", j );
```

The reason for the second test is that two thread could both test `i < 10` when `i` is `9`, and the (in the original code), both would increment `i`. The revised code performs a quick test outside of the lock; if the test is true, the thread acquires the lock and performs the test again. If the test is now false, the thread releases the lock without incrementing `i`; if the test is still true, then the thread increments `i`.

Thanks to Brian Toonen <toonen@mcs.anl.gov>.

p 138 and p 203 The call to `MPI_Win_create` in Figure 5.2 and in Figure 6.10 passes `NULL` as the buffer pointer for the case where the buffer size is zero. This is correct, but Fortran users will need to use `MPI_BOTTOM` instead. To make the example clear to both C and Fortran programmers, consider using `MPI_BOTTOM` instead of `NULL` when no buffer is be provided to `MPI_Win_create` (buffer size is zero).

Thanks to Takao Hatazaki <Takao.Hatazaki@JP.COMPAQ.com>.

p 141 In Table 5.3, `Aint`, `Info`, and `Intracomm` are missing the `MPI::` prefix.

Thanks to Takao Hatazaki.

p 145 In Table 5.6, `Aint` and `Datatype` are missing `MPI::` in the binding for `MPI::Win::Put`.

Thanks to Takao Hatazaki.

p 151 The comment starting "We need a fence between..." should be placed before the second to the last `MPI_Win_fence` call to more clearly indicate the the reason for that `MPI_Win_fence` call.

Thanks to Takao Hatazaki.

p154–156 All occurences of "right" should be replaced with "top" and all occurences of "left" should be replaced with "bottom". The Figures have the correct description of the decomposition and the code.

Thanks to Takao Hatazaki.

p156 The sample code in Figure 5.12 uses an apparently unsafe combination of `MPI_Send` and `MPI_Recv`. This is acceptable for this code because only a single integer is being sent and no MPI implementation is so restrictive that `MPI_Send` will block with a single integer. However, it would not be incorrect, and deadlock detection tools might flag this usage. In any case, an `MPI_Sendrecv` is a better way to implement this step. (The code is only "apparently" unsafe because the domain is not periodic, and, as explained in *Using MPI*, the code only serializes, it does not deadlock. However, using MPI_Sendrecv is a better solution.)

p 158, line 2 The code fragment

```
newtype = MPI::Datatype::Match_size( MPI::TYPECLASS_INTEGER,
                                     sizeof(MPI::Aint )
```

should be

```
newtype = MPI::Datatype::Match_size( MPI::TYPECLASS_INTEGER,
                                     sizeof(MPI::Aint ) );
```

p 159 The example code in the figure is incorrect. The corrected code is given below.

```
subroutine exchng1( a, nx, s, e, win, &
                    bottom_nbr, top_nbr )
use mpi
integer nx, s, e
double precision a(0:nx+1,s-1:e+1)
integer win, bottom_nbr, top_nbr
integer ierr

call MPI_WIN_FENCE( 0, win, ierr )
! Put top edge into top neighbor's ghost cells
call MPI_PUT( a(1,e), nx, MPI_DOUBLE_PRECISION, &
              top_nbr, 1, nx, MPI_DOUBLE_PRECISION, win, ierr )
! Get top edge from top neighbor's first column
call MPI_GET( a(1,e+1), nx, MPI_DOUBLE_PRECISION, &
              top_nbr, nx + 3, nx, MPI_DOUBLE_PRECISION, win, ierr )
call MPI_WIN_FENCE( 0, win, ierr )

return
end
```

Thanks to Bo-Wen Shen <bwshen@hera.gsfc.nasa.gov> and Takao Hatazaki.

p 159 Change

> Instead of putting data into ghost cells only on remote processes, we can put data into the ghost cells of the process on the top, starting at a displacement of one, and we can get the ghost cells for our part of the grid on the bottom edge by getting grid data from the first column of the process on the bottom.

to

> Instead of putting data into ghost cells only on remote processes, we can put data into the ghost cells of the process on the top, starting at a displacement of one, and we can get the ghost cells for our part of the grid on the top edge by getting grid data from the first column of the process on the top.

Thanks to Takao Hatazaki.

p 159 Change "left" to "bottom" and "right" to "top" in

> Also note that there is no explicit reference to the `left_nbr` in the above code: the "get from right neighbor" replaces the "put to left neighbor."

p 160 Add a closing parenthesis at the end of

> (e.g., we must send `nx+1` values starting from `a(0,m)` rather than `nx` values starting from `a(1,m)`).

Thanks to Takao Hatazaki.

p 160 `double precision a(sx-1:ex+1,sy-1:sy+1)`

should be

```
double precision a(sx-1:ex+1,sy-1:ey+1)
```

Thanks to Takao Hatazaki.

p 161 The same comment holds for the use of `MPI_Send` and `MPI_Recv` here as for page 156: this code should also use `MPI_Sendrecv` instead of `MPI_Send` and `MPI_Recv`.

Thanks to Takao Hatazaki.

p 164

```
do i=1,ny
     buf(i) = a(1,i-sy+1)
enddo
call MPI_WIN_FENCE( 0, winbuf, ierr )
call MPI_PUT( buf, ny, MPI_DOUBLE_PRECISION, top_nbr, &
              0, ny, MPI_DOUBLE_PRECISION, winbuf, ierr )
... similar code for the bottom edge
```

should be

```
do i=1,ny
     buf(i) = a(1,sy+i-1)
enddo
call MPI_WIN_FENCE( 0, winbuf, ierr )
call MPI_PUT( buf, ny, MPI_DOUBLE_PRECISION, left_nbr, &
              0, ny, MPI_DOUBLE_PRECISION, winbuf, ierr )
... similar code for the right edge
```

Thanks to Takao Hatazaki.

p 164–165 The example in Figure 5.18 assumes that nx and ny are the same on all processes.

Thanks to Takao Hatazaki.

p 166 Change

> It would be better to move the data in `t` and immediately add it to `s` to form `w`.

to

> It would be better to move the data in `t` and immediately add it to the `t` for rank zero to form `w` on rank zero.

Thanks to Takao Hatazaki.

p 167 In Table 5.16, `Aint`, `Datatype`, and `Op` should have the `MPI::` prefix.

Thanks to Takao Hatazaki.

p 167 and p 168 In the code fragment, `Win_create` should be `Win::Create`.

Thanks to Takao Hatazaki.

p 171 There is a closing parenthesis missing in

> ...(e.g., `A = 0` for an array `A` in Fortran)

Thanks to Takao Hatazaki.

p 171 The word "that" should be "than" in the first sentence in the last paragraph:

> ... less restrictive than writing to memory ...

Thanks to Brad Penoff.

p 186 In Table 6.6, `Aint` and `Info` are missing `MPI::` in the binding for `MPI::Win::Alloc_mem`.

Thanks to Takao Hatazaki.

p 191 The binding for the C++ version of `MPI::Win::Get_attr` should not include the Window as a parameter. This was an error in the MPI standard that has been corrected in the MPI-2 errata.

p196 On line 4 from the bottom, `MPI_Win_lock` should be replaced with `MPI_Win_unlock`.

Thanks to Takao Hatazaki.

p 202 Replace `MPE_Counter_delete` with `MPE_Counter_free`.

Thanks to Takao Hatazaki.

p 203 The example uses the name `old_comm` for the input communicator. This parallels the MPI-1 version of this routine.

Thanks to Takao Hatazaki.

p 206 The root boxes are missing the slanted lines mentioned in the caption. These were lost when the book was produced. Postscript for this figure is available at `http://www.mcs.anl.gov/mpi/usingmpi2/treesteps.eps`.

Thanks to Takao Hatazaki.

p 206 Replace the sentance that begins "Thus, to compute the sum" with

> Thus, to compute the sum, we need only add up the contributions from the sibling of the node, the sibling of its parent, the sibling of its grandparent, and the sibling of grandparent's parent.

The original text had confusing use of plurals.

Thanks to Takao Hatazaki.

p 208 Replace Figure 6.16 with:

```
/* Get the largest power of two smaller than size */
mask = 1;
while (mask < size) mask <<= 1;
mask >>= 1;

level = 0;
idx   = 0;
while (mask >= 1) {
    if (rank < mask) {
        /* go to left for acc_idx, go to right for
           get_idx. set idx=acc_idx for next iteration */
        acc_idx[level] = idx + 1;
        get_idx[level] = idx + mask*2;
        idx            = idx + 1;
    }
    else {
        /* go to right for acc_idx, go to left for
           get_idx. set idx=acc_idx for next iteration */
        acc_idx[level] = idx + mask*2;
        get_idx[level] = idx + 1;
        idx            = idx + mask*2;
    }
    level++;
    rank = rank % mask;
    mask >>= 1;
}
```

Thanks to Rajeev Thakur.

p 210 In line 2, note that the `size` mutexes are distributed across the processes, with one fetch-and-increment tree used for each mutex. The first `size` processes get one mutex (assuming the `size` is less than or equal to the number of processes).

Thanks to Takao Hatazaki.

p 218 Replace `\relax0` with `\0`.

Thanks to Takao Hatazaki.

p 221 Figure 6.24. A better design here would keep `win` with `head`, either both as global variables, or, better, a struct containing both passed to `FindElm`.

Thanks to Brian Toonen <toonen@mcs.anl.gov>.

p 224 The last sentance in Section 6.9.3 is not correct. In order to use the order of statements to keep the updates to the list correct, it is necessary, as discussed in Section 4.3.2, to apply a write barrier before the assignment `last_ptr->next = new_ptr`.

Thanks to Brian Toonen <toonen@mcs.anl.gov>.

p 224–5 Note that these routines could benefit from having a shared read, exclusive write version of `MPE_Mutex_lock`.

Thanks to Brian Toonen <toonen@mcs.anl.gov>.

p 227–8 The insertion of a new head element is not handled correctly here. The key problem here is that the head element is not in the local window (it is local to each process) and thus cannot be updated by a remote process. The fix is to keep a dummy "head" on each local list that is stored in the window and thus can be updated remotely.

Thanks to Brian Toonen <toonen@mcs.anl.gov>.

p 231 In the discussion of `MPI_MODE_NOPUT`, replace `MPI_Win_complete` with `MPI_Win_wait`.

Thanks to Takao Hatazaki.

p235 In the footnote, it is more appropriate to use MPMD (Multiple Program Multiple Data) rather than MIMD.

Thanks to Takao Hatazaki.

p267 The `counter_nxtval` routine in Figure 8.1 should be

```
/* Any process can call this to fetch and increment by value */
void counter_nxtval( MPI_Comm counter_comm, int incr, int *value )
{
    MPI_Send(&incr, 1, MPI_INT, 0, 0, counter_comm);
    MPI_Recv(value, 1, MPI_INT, 0, 0, counter_comm, MPI_STATUS_IGNORE);
}
```

(the arguments to `MPI_Send` were wrong).

Thanks to Takao Hatazaki.

p 281–2 The example in Figures 9.2 and 9.3 does not properly terminate the thread. The simplest fix is to add an `pthread_detach` in Figure 9.2 after the `pthread_create` call. A better solution would save the thread id returned by `pthread_create` in a new field in the `params_struct` structure, and perform a `pthread_join` in `free_fn` (in Figure 9.3) before freeing the structure. This approach would allow a non-trivial implementation of the cancel function (`cancel_fn` in Figure 9.3).

p 302 Insert "as" into "well as the" in the first line on the page.

Thanks to Rusty Lusk <lusk@mcs.anl.gov>.