(HPSC **5576** ELIZABETH JESSUP)

HIGH PERFORMANCE SCIENTIFIC COMPUTING

:: Homework / 7

:: Student / Florian Rappl

1 problem / **10** points

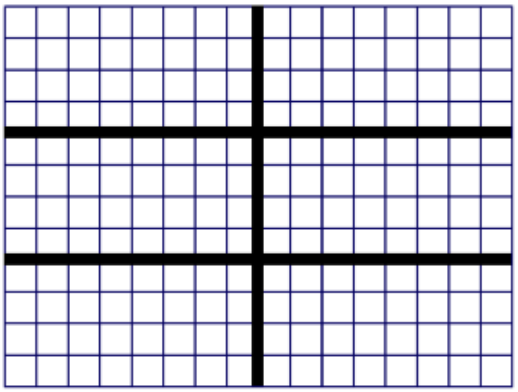
Problem 1

Task:

Write a short program demonstrating the use of *MPI_Type_create_darray()* to read data from a single disk file, distributing it to multiple processors, using MPI-IO, *10 pts*

Consider an array in a file containing the elements { 0, 1, 2, 3, ... } stored as consecutive binary (4-byte) integers.

Write a program that uses MPI-IO and *MPI_Type_create_darray()* to read the first 192 elements of this file and distribute it among 6 processors in a (x=2,y=3) Cartesian topology, allocating 8 columns and 4 rows to each processor as shown:



0	1
2	3
4	5

After reading the array, have each processor print out the subarray containing the 32 values that were assigned to it. Verify that the processors read the global array properly; that is, if you arrange the distributed memory according to the process topology, it recreates the array.

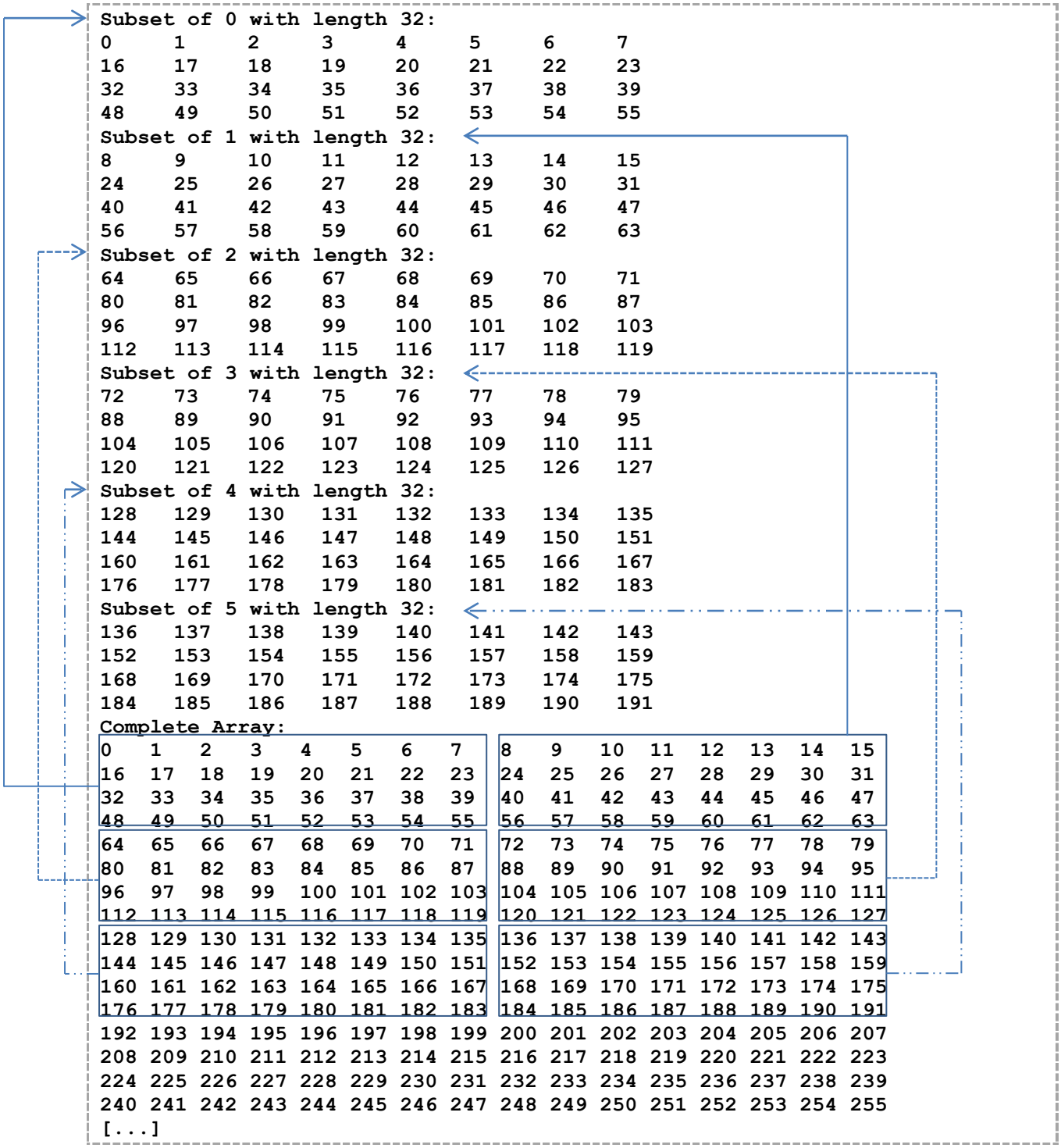
Solution:

My solution that I thought should work did not work on my workstation (Windows, Visual Studio 2008). Apparently the HPSC Pack from Microsoft lacks some implementations. After spending several hours trying to come up with some solution I just tested it on Trestles and it did work from the beginning.

Unfortunately this was not the only problem I faced using Windows. Another one was more strange – but the workaround did work quite nice in this case. Apparently *fread()* is implemented quite strangely, because integers bigger than 25 were displayed as -858993460. This happened while using *fwrite()* as well as the MPI equivalent. Therefore I decided to switch only to the MPI implementations – those worked quite well.

The solution consists of 4 functions: the main function where all calls are made, a function to print the arrays consisting of 8 (subset) or 16 (full) columns, as well as a function to read and to write the file. I tested the program using the output I got from running it. In order to have a structured output a buffer was added to the PrintSubset() function. Therefore only full blocks will be printed.

Program output:



The boxes have been added manually in order to explain the output and thus the correctness.

Code printout

```
1  #define MAX_FILENAME 255      /* Max Length of Filename     */
2  #define MAXLENGTH 32          /* Subset Maxlength 4 * 8     */
3  #define FILECONTENT 1000      /* Length of filecontent / 4*/
4  #include <stdio.h>
5  #include <string.h>
6  #include "mpi.h"
```

```
7  /* Prototypes */
8  void WriteFile(char fileName[], MPI_Status *status);
9  void PrintSubset(int my_rank, int buffer[], int length);
10 void ReadFile(char fileName[], MPI_Status *status);
11
12 int main(int argc, char* argv[])
13 {
14       int               my_rank;    /* rank of process            */
15       int               p;          /* number of processes        */
16       int               tag = 0;    /* tag for messages           */
17       int               my_name_len;/* length of my_name          */
18       MPI_Status        status;     /* return status for rec      */
19       MPI_File          fh;         /* MPI_FILE storage           */
20       int               gsizes[3];  /* array with lengths         */
21       int               gdistr[3];  /* array with distributions   */
22       int               gdargs[3];  /* array of dimensions        */
23       int               psizes[3];  /* array of sub-array-sizes   */
24       MPI_Datatype      rType;      /* new view of the file       */
25       char              fileName[MAX_FILENAME];
26       char              my_name[MPI_MAX_PROCESSOR_NAME];
27       int*              buffer;     /* the buffer for the file    */
28
29       /* set the filename - could also be done over args */
30       sprintf(&fileName, "rawdatafile");
31
32       /* set the darray parameter - dim [0] (eq. z) is obsol.    */
33       gsizes[0] = 1;
34       gsizes[1] = 12;
35       gsizes[2] = 16;
36       gdistr[0] = MPI_DISTRIBUTE_NONE;
37       gdistr[1] = MPI_DISTRIBUTE_BLOCK;
38       gdistr[2] = MPI_DISTRIBUTE_BLOCK;
39       gdargs[0] = MPI_DISTRIBUTE_NONE;
40       gdargs[1] = MPI_DISTRIBUTE_DFLT_DARG;
41       gdargs[2] = MPI_DISTRIBUTE_DFLT_DARG;
42       psizes[0] = 1;
43       psizes[1] = 3;
44       psizes[2] = 2;
45
46       /* allocate Buffer */
47       buffer = (int*)malloc(sizeof(int) * MAXLENGTH);
48
49       /* Start up MPI */
50       MPI_Init(&argc, &argv);
51
52       /* Find out process rank  */
53       MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
54
55       /* Find out number of processes */
56       MPI_Comm_size(MPI_COMM_WORLD, &p);
57
58       /* Modified from Pacheco -- get machine name */
59       MPI_Get_processor_name( my_name, &my_name_len );
60
61       /* Masterprocess creates the file */
62       if(my_rank == 0)
63             WriteFile(fileName, &status);
64
65       /* We hardcoded everything for 6(!) processes */
66       if(p == 6)
67       {
```

```
68              /* Create the own datatype darray */
69              MPI_Type_create_darray(p, my_rank, 3, gsizes, gdistr,
70                              gdargs, psizes, MPI_ORDER_C, MPI_INT, &rType);
71
72              /* Wait for the file to be written and commit datatype */
73              MPI_Type_commit(&rType);
74
75              /* Open the file and create the file type - used as sync */
76              MPI_File_open(MPI_COMM_WORLD, &fileName, MPI_MODE_RDONLY,
77                              MPI_INFO_NULL, &fh);
78
79              /* Set the view to read in the elements */
80              MPI_File_set_view(fh, 0, MPI_INT, rType, "native",
81                              MPI_INFO_NULL);
82
83              /* Read the file using the set view above */
84              MPI_File_read(fh, buffer, MAXLENGTH, MPI_INT, &status);
85
86              /* Close the file */
87              MPI_File_close(&fh);
88
89              /* Print the subarray that has been read out */
90              PrintSubset(my_rank, buffer, MAXLENGTH);
91      }
92      /* Print the whole file i.O. to see file content */
93      if(my_rank == 0)
94              ReadFile(fileName, &status);
95
96      /* Shut down MPI */
97      MPI_Finalize();
98
99      /* Clear the buffer */
100     free(buffer);
101
102     return 0;
103 } /* main */
104
105 void WriteFile(char fileName[], MPI_Status *status)
106 {
107     int k;
108     MPI_File ufh;
109
110     /* open the file in read+write and create mode (only 1 process) */
111     MPI_File_open(MPI_COMM_SELF, fileName, MPI_MODE_CREATE |
112                     MPI_MODE_RDWR, MPI_INFO_NULL, &ufh);
113
114     /* write the data */
115     for(k = 0; k < FILECONTENT; k++)
116             MPI_File_write(ufh, &k, 1, MPI_INT, status);
117
118     /* close the file */
119     MPI_File_close(&ufh);
120 } /* WriteFile */
121
122 void ReadFile(char fileName[], MPI_Status *status)
123 {
124     int i;
125     /* set the Buffer */
126     int* buffer = (int*)malloc(sizeof(int) * FILECONTENT);
127     MPI_File ufh;
128
```

```
129         /* open the file in read mode (only 1 process - that's why _SELF) */
130         MPI_File_open(MPI_COMM_SELF, fileName, MPI_MODE_RDONLY,
131                       MPI_INFO_NULL, &ufh);
132
133         /* read in the data */
134         for(i = 0; i < FILECONTENT; i++)
135               MPI_File_read(ufh, &buffer[i], 1, MPI_INT, status);
136
137         /* close the file */
138         MPI_File_close(&ufh);
139
140         /* print the whole array - rank: -1 */
141         PrintSubset(-1, buffer, FILECONTENT);
142
143         /* free the memory */
144         free(buffer);
145   } /* ReadFile */
146
147   void PrintSubset(int my_rank, int *buffer, int length)
148   {
149         int i, j;
150         /* set the Buffer */
151         char* buff = (char*)malloc(sizeof(char) * 10000);
152
153         if(my_rank != -1) /* Special case for whole array */
154         {
155               j = 8;
156               sprintf(buff, "Subset of %d with length %d:\n",
157                           my_rank, length);
158         }
159         else /* General case for the subset */
160         {
161               j = 16;
162               sprintf(buff, "Complete Array:\n");
163         }
164
165         /* Generate the message */
166         for(i = 0; i < length; i++)
167               sprintf(buff, "%s%d\t%s", buff, buffer[i],
168                           (i % j == j - 1) ? "\n" : "");
169
170         /* Print the message */
171         printf("%s", buff);
172
173         /* Clear the memory */
174         free(buff);
175   } /* PrintSubset */
```