

# CSCI 455: Lab #7 — Advanced MPI: Derived Data Type

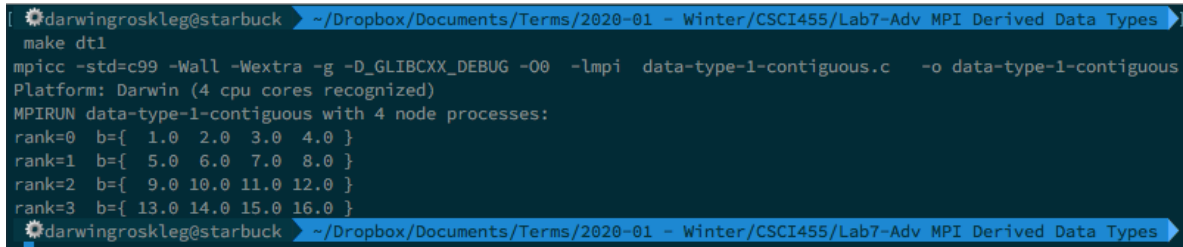
Darwin Jacob Groskleg

Winter 2020

## Contents

<b>Part 1: Contiguous Derived Data Types</b>	<b>1</b>
data-type-1-contiguous.c . . . . .	2
<b>Part 2: Vector Derived Data Types</b>	<b>4</b>
data-type-2-vector.c . . . . .	5
<b>Part 3: Indexed Derived Data Types</b>	<b>7</b>
data-type-3-indexed.c . . . . .	8
<b>Part 4: Struct Derived Data Types</b>	<b>10</b>
data-type-4-struct.c . . . . .	11

## Part 1: Contiguous Derived Data Types



```
[darwin@starbuck ~/Dropbox/Documents/Terms/2020-01 - Winter/CSCI455/Lab7-Adv MPI Derived Data Types]
make dt1
mpicc -std=c99 -Wall -Wextra -g -D_GLIBCXX_DEBUG -O0 -lm -o data-type-1-contiguous.c -o data-type-1-contiguous
Platform: Darwin (4 cpu cores recognized)
MPIRUN data-type-1-contiguous with 4 node processes:
rank=0 b={ 1.0 2.0 3.0 4.0 }
rank=1 b={ 5.0 6.0 7.0 8.0 }
rank=2 b={ 9.0 10.0 11.0 12.0 }
rank=3 b={ 13.0 14.0 15.0 16.0 }
```

Figure 1: Console output

**data-type-1-contiguous.c**

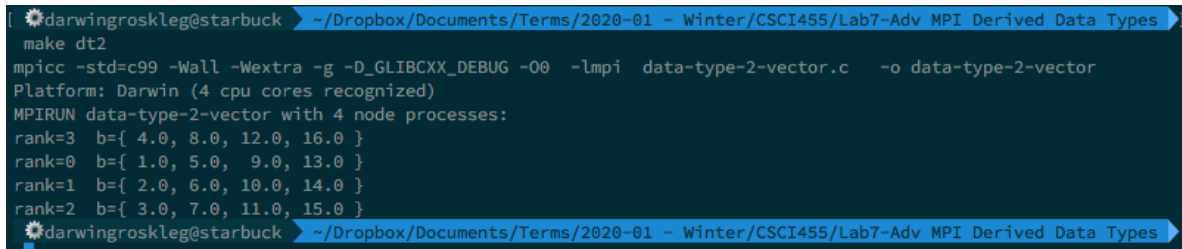
```

1  /* data-type-1-contiguous.c
2  * -----
3  * Authors: Darwin Jacob Groskleg
4  *
5  * Contiguous Derived Data Type:
6  *     a data type representing a row of an array and distribute a different
7  *     row to all processes.
8  */
9  #include <stdio.h>
10 #include <mpi.h>
11
12 #define DATA_SET_WIDTH 4
13
14 enum TaskRanks { SendTaskRank = 0 };
15
16 int main(int argc, char *argv[]) {
17     float a[DATA_SET_WIDTH][DATA_SET_WIDTH] = {
18         { 1.0, 2.0, 3.0, 4.0},
19         { 5.0, 6.0, 7.0, 8.0},
20         { 9.0, 10.0, 11.0, 12.0},
21         {13.0, 14.0, 15.0, 16.0}
22     };
23     float b[DATA_SET_WIDTH];
24
25     MPI_Init(&argc, &argv);
26     int rank;
27     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
28     int cluster_size;
29     MPI_Comm_size(MPI_COMM_WORLD, &cluster_size);
30
31     // create contiguous derived data type
32     MPI_Datatype rowtype; // required variable
33     MPI_Type_contiguous(DATA_SET_WIDTH, MPI_FLOAT, &rowtype);
34     MPI_Type_commit(&rowtype);
35
36     if (cluster_size == DATA_SET_WIDTH) {
37         int tag=1;
38         if (rank == SendTaskRank) {
39             for (int i=0; i<cluster_size; i++)
40                 MPI_Send(
41                     &a[i][0], // send the i'th row
42                     1, // rows to send
43                     rowtype,
44                     i, // destination rank
45                     tag,
46                     MPI_COMM_WORLD);
47         }
48
49         // all tasks receive rowtype data from task 0
50         MPI_Status stat;
51         MPI_Recv(
52             &b,
53             1, // rows received
54             rowtype,
55             SendTaskRank, // from master rank

```

```
56         tag,  
57         MPI_COMM_WORLD,  
58         &stat);  
59     printf("rank=%d  b={ %4.1f %4.1f %4.1f %4.1f }\n",  
60           rank,  b[0], b[1], b[2], b[3]);  
61 }  
62 else  
63     printf("Must specify %d processors. Terminating.\n", DATA_SET_WIDTH);  
64  
65     // free datatype when done using it  
66     MPI_Type_free(&rowtype);  
67     MPI_Finalize();  
68     return 0;  
69 }
```

## Part 2: Vector Derived Data Types

A terminal window with a dark blue background and light blue text. The prompt is 'darwinroskleg@starbuck' followed by a blue arrow pointing to the right. The path is '~/Dropbox/Documents/Terms/2020-01 - Winter/CSCI455/Lab7-Adv MPI Derived Data Types'. The user enters 'make dt2'. The output shows the compilation command 'mpicc -std=c99 -Wall -Wextra -g -D\_GLIBCXX\_DEBUG -O0 -lmpi data-type-2-vector.c -o data-type-2-vector', the platform 'Darwin (4 cpu cores recognized)', and the MPIRUN command 'MPIRUN data-type-2-vector with 4 node processes:'. The output then shows four ranks with their respective vector values: rank=3 b={ 4.0, 8.0, 12.0, 16.0 }, rank=0 b={ 1.0, 5.0, 9.0, 13.0 }, rank=1 b={ 2.0, 6.0, 10.0, 14.0 }, and rank=2 b={ 3.0, 7.0, 11.0, 15.0 }.

```
[ darwingroskleg@starbuck ~/Dropbox/Documents/Terms/2020-01 - Winter/CSCI455/Lab7-Adv MPI Derived Data Types ]
make dt2
mpicc -std=c99 -Wall -Wextra -g -D_GLIBCXX_DEBUG -O0 -lmpi data-type-2-vector.c -o data-type-2-vector
Platform: Darwin (4 cpu cores recognized)
MPIRUN data-type-2-vector with 4 node processes:
rank=3 b={ 4.0, 8.0, 12.0, 16.0 }
rank=0 b={ 1.0, 5.0, 9.0, 13.0 }
rank=1 b={ 2.0, 6.0, 10.0, 14.0 }
rank=2 b={ 3.0, 7.0, 11.0, 15.0 }
[ darwingroskleg@starbuck ~/Dropbox/Documents/Terms/2020-01 - Winter/CSCI455/Lab7-Adv MPI Derived Data Types ]
```

Figure 2: Console output

**data-type-2-vector.c**

```

1  /* data-type-2-vector.c
2  * -----
3  * Authors: Darwin Jacob Groskleg
4  *
5  * Vector Derived Data Type:
6  *     a data type representing a column of an array and distribute
7  *     different columns to all processes.
8  */
9  #include <stdio.h>
10 #include <mpi.h>
11
12 #define DATA_SET_WIDTH 4
13
14 enum TaskRanks { SendTaskRank = 0 };
15
16 int main(int argc, char *argv[]) {
17     float a[DATA_SET_WIDTH][DATA_SET_WIDTH] = {
18         { 1.0, 2.0, 3.0, 4.0},
19         { 5.0, 6.0, 7.0, 8.0},
20         { 9.0, 10.0, 11.0, 12.0},
21         {13.0, 14.0, 15.0, 16.0}
22     };
23     float b[DATA_SET_WIDTH];
24
25     MPI_Init(&argc, &argv);
26     int rank;
27     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
28     int cluster_size;
29     MPI_Comm_size(MPI_COMM_WORLD, &cluster_size);
30
31     //if (rank == SendTaskRank) {
32     //     printf("a={ %.1f", *(a[0] + 0));
33     //     for (int i=0; i<DATA_SET_WIDTH*DATA_SET_WIDTH; i++)
34     //         printf(", %.1f", *(a[0] + i));
35     //     printf(" }\n");
36     //}
37     //MPI_Barrier(MPI_COMM_WORLD);
38
39     // create vector derived data type
40     MPI_Datatype columntype; // required variable
41     int count = DATA_SET_WIDTH; // count of elements of type float
42     int blocklength = 1;
43     int stride = DATA_SET_WIDTH;
44     // strided contiguous data type
45     MPI_Type_vector(count, blocklength, stride, MPI_FLOAT, &columntype);
46     MPI_Type_commit(&columntype);
47
48     if (cluster_size == DATA_SET_WIDTH) {
49         int tag=1;
50         MPI_Status stat;
51         // task 0 sends one element of columntype to all tasks
52         if (rank == SendTaskRank) {
53             for (int i=0; i<cluster_size; i++)
54                 MPI_Send(
55                     &a[0][i], // send i'th column

```

```
56         1,           // columns being sent
57         columntype,
58         i,           // destination rank
59         tag,
60         MPI_COMM_WORLD);
61     }
62
63     // all tasks receive columntype data from task 0
64     MPI_Recv(
65         &b,
66         DATA_SET_WIDTH,           // columns received
67         MPI_FLOAT,
68         SendTaskRank,
69         tag,
70         MPI_COMM_WORLD,
71         &stat);
72     printf("rank=%d  b={ %3.1f, %3.1f, %4.1f, %4.1f }\n",
73           rank,  b[0],  b[1],  b[2],  b[3]);
74 }
75 else
76     printf("Must specify %d processors. Terminating.\n", DATA_SET_WIDTH);
77
78 // free datatype when done using it
79 MPI_Type_free(&columntype);
80 MPI_Finalize();
81 return 0;
82 }
```

## Part 3: Indexed Derived Data Types

```
[darwin@starbuck ~/Dropbox/Documents/Terms/2020-01 - Winter/CSCI455/Lab7-Adv MPI Derived Data Types]
make dt3
mpicc -std=c99 -Wall -Wextra -g -D_GLIBCXX_DEBUG -O0 -lmpi data-type-3-indexed.c -o data-type-3-indexed
Platform: Darwin (4 cpu cores recognized)
MPIRUN data-type-3-indexed with 4 node processes:
rank=3 b={ 6.0 7.0 8.0 9.0 13.0 14.0 }
rank=0 b={ 6.0 7.0 8.0 9.0 13.0 14.0 }
rank=1 b={ 6.0 7.0 8.0 9.0 13.0 14.0 }
rank=2 b={ 6.0 7.0 8.0 9.0 13.0 14.0 }
[darwin@starbuck ~/Dropbox/Documents/Terms/2020-01 - Winter/CSCI455/Lab7-Adv MPI Derived Data Types]
```

Figure 3: Console output



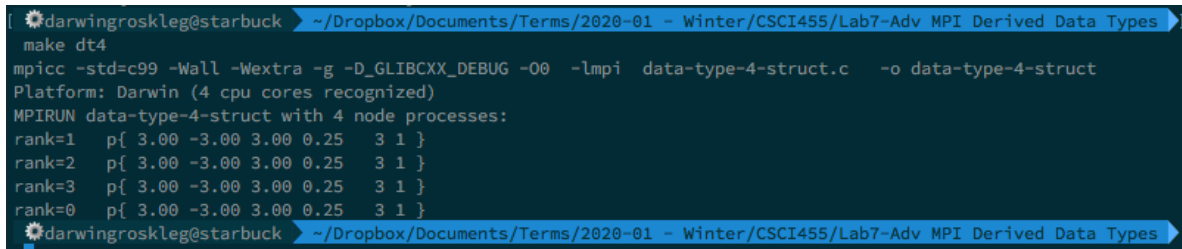
**data-type-3-indexed.c**

```

1  /* data-type-3-indexed.c
2  * -----
3  * Authors: Darwin Jacob Groskleg
4  *
5  * Index Derived Data Type:
6  *     a datatype to extract variable portions of an array and then distribute
7  *     to them to all tasks.
8  */
9  #include <stdio.h>
10 #include "mpi.h"
11
12 #define A_SET_SIZE 16
13 #define B_SET_SIZE 6      // subset of A
14 #define BLOCKCOUNT 2
15
16 enum TaskRanks { SendTaskRank = 0 };
17
18 int main(int argc, char *argv[]) {
19     float a[A_SET_SIZE] = {
20         1.0, 2.0, 3.0, 4.0,
21         5.0, 6.0, 7.0, 8.0,
22         9.0, 10.0, 11.0, 12.0,
23         13.0, 14.0, 15.0, 16.0
24     };
25     float b[B_SET_SIZE];    // subset of A
26
27     MPI_Init(&argc, &argv);
28     int rank;
29     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
30     int cluster_size;
31     MPI_Comm_size(MPI_COMM_WORLD, &cluster_size);
32
33     const int blockcount = BLOCKCOUNT;
34     int blocklengths[BLOCKCOUNT] = { 4, 2 };
35     int displacements[BLOCKCOUNT] = { 5, 12 };
36     blocklengths[0] = 4;
37     blocklengths[1] = 2;
38     displacements[0] = 5;
39     displacements[1] = 12;
40
41     // create indexed derived data type
42     MPI_Datatype indexedtype;
43     MPI_Type_indexed(blockcount, blocklengths, displacements, MPI_FLOAT,
44                     &indexedtype);
45     MPI_Type_commit(&indexedtype);
46
47     int tag=1;
48     if (rank == SendTaskRank) {
49         for (int dest_rank=0; dest_rank<cluster_size; dest_rank++)
50             // task 0 sends one element of indexedtype to all tasks
51             MPI_Send(
52                 &a,
53                 1, // count of indexedtype
54                 indexedtype,
55 
```

```
56         dest_rank,  
57         tag,  
58         MPI_COMM_WORLD);  
59     }  
60  
61     // all tasks receive indexedtype data from task 0  
62     MPI_Status stat;  
63     MPI_Recv(&b,  
64             B_SET_SIZE,      // target buffer size  
65             MPI_FLOAT,  
66             SendTaskRank,  
67             tag,  
68             MPI_COMM_WORLD,  
69             &stat);  
70     printf("rank=%d  b={ %3.1f %3.1f %3.1f %3.1f %3.1f %3.1f }\n",  
71           rank,  b[0], b[1], b[2], b[3], b[4], b[5]);  
72  
73     // free datatype when done using it  
74     MPI_Type_free(&indexedtype);  
75     MPI_Finalize();  
76     return 0;  
77 }
```

## Part 4: Struct Derived Data Types



```
[darwin@starbuck ~/Dropbox/Documents/Terms/2020-01 - Winter/CSCI455/Lab7-Adv MPI Derived Data Types]$ make dt4
mpicc -std=c99 -Wall -Wextra -g -D_GLIBCXX_DEBUG -O0 -lmpi data-type-4-struct.c -o data-type-4-struct
Platform: Darwin (4 cpu cores recognized)
MPIRUN data-type-4-struct with 4 node processes:
rank=1  p{ 3.00 -3.00 3.00 0.25  3 1 }
rank=2  p{ 3.00 -3.00 3.00 0.25  3 1 }
rank=3  p{ 3.00 -3.00 3.00 0.25  3 1 }
rank=0  p{ 3.00 -3.00 3.00 0.25  3 1 }
```

The screenshot shows a terminal window with a dark background and light blue text. The prompt is `[darwin@starbuck ~/Dropbox/Documents/Terms/2020-01 - Winter/CSCI455/Lab7-Adv MPI Derived Data Types]`. The user enters `make dt4`, which triggers the compilation of `data-type-4-struct.c` into `data-type-4-struct` using `mpicc` with various flags. The output indicates the platform is Darwin with 4 CPU cores recognized. Then, `MPIRUN` is used to execute `data-type-4-struct` with 4 node processes. The output shows four ranks (0, 1, 2, 3) each printing a struct `p` containing five values: `3.00`, `-3.00`, `3.00`, `0.25`, and `3 1`.

Figure 4: Console output

**data-type-4-struct.c**

```

1  /* data-type-4-struct.c
2  * -----
3  * Authors: Darwin Jacob Groskleg
4  *
5  * Struct Derived Data Type:
6  *     a data type that represents a particle and distribute an array of such
7  *     particles to all processes.
8  */
9  #include <stdio.h>
10 #include "mpi.h"
11
12 #define NELEM 25
13
14 enum TaskRanks { SendTaskRank = 0 };
15
16 int main(int argc, char *argv[]) {
17     typedef struct {
18         float x, y, z;
19         float velocity;
20
21         int n, type;
22     } Particle;
23
24     MPI_Init(&argc,&argv);
25     int rank;
26     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
27     int cluster_size;
28     MPI_Comm_size(MPI_COMM_WORLD, &cluster_size);
29
30     // required variables
31     int composed_type_count = 2; // float, int
32     MPI_Datatype composed_types[2];
33     int blockcounts[2]; // block by type
34
35     // MPI_Aint type used to be consistent with syntax of
36     // MPI_Type_extent routine
37     MPI_Aint offsets[2];
38     MPI_Aint extent;
39
40     // Descriptions of the 4 MPI_FLOAT fields:
41     //     x, y, z, velocity
42     offsets[0] = 0;
43     composed_types[0] = MPI_FLOAT;
44     blockcounts[0] = 4;
45
46     // Descriptions of the 2 MPI_INT fields:
47     //     n, type
48     // Must first figure offset by getting size of MPI_FLOAT
49     // DEPRECATED CALL:
50     //     MPI_Type_extent(MPI_FLOAT, &extent);
51     MPI_Type_get_extent(MPI_FLOAT, &extent, &extent);
52     offsets[1] = 4 * extent;
53     composed_types[1] = MPI_INT;
54     blockcounts[1] = 2;
55

```

```

56 // define structured type and commit it
57 MPI_Datatype particletype;
58 // NOTE MPI_Type_struct is deprecated in MPI3 !
59 //MPI_Type_struct(composed_type_count, blockcounts, offsets, composed_types,
60 //                &particletype);
61 MPI_Type_create_struct(composed_type_count, blockcounts, offsets,
62                       composed_types, &particletype);
63 MPI_Type_commit(&particletype);
64
65
66 Particle particles[NELEM]; // starting set
67 Particle p[NELEM];
68
69 // task 0 initializes the particle array and then sends it to each task
70 int tag=1;
71 MPI_Status stat;
72 if (rank == SendTaskRank) {
73     for (int i=0; i<NELEM; i++) {
74         particles[i].x      = i * 1.0;
75         particles[i].y      = i * -1.0;
76         particles[i].z      = i * 1.0;
77         particles[i].velocity = 0.25;
78         particles[i].n       = i;
79         particles[i].type    = i % 2;
80     }
81     for (int dest_rank=0; dest_rank<cluster_size; dest_rank++)
82         MPI_Send(particles, NELEM, particletype, dest_rank, tag,
83                 MPI_COMM_WORLD);
84 }
85
86 // all tasks receive particletype data
87 MPI_Recv(p, NELEM, particletype, SendTaskRank, tag, MPI_COMM_WORLD, &stat);
88
89 printf("rank=%d  p{ %.2f %.2f %.2f %.2f  %d %d }\n",
90        rank,  p[3].x, p[3].y, p[3].z, p[3].velocity, p[3].n, p[3].type);
91
92 // free datatype when done using it
93 MPI_Type_free(&particletype);
94 MPI_Finalize();
95 return 0;
96 }

```