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

Darwin Jacob Groskleg

Winter 2020

## Contents

# Part 1: Contiguous Derived Data Types

```
[ darwingroskleg@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  -lmpi  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 }
 darwingroskleg@starbuck ~/Dropbox/Documents/Terms/2020-01 - Winter/CSCI455/Lab7-Adv MPI Derived Data Types
```

Figure 1: Console output

### data-type-1-contiguous.c

```c
/* data-type-1-contiguous.c
 * ------------------------
 * Authors: Darwin Jacob Groskleg
 *
 * Contiguous Derived Data Type:
 *      a data type representing a row of an array and distribute a different
 *      row to all processes.
 */
#include <stdio.h>
#include <mpi.h>

#define DATA_SET_WIDTH 4

enum TaskRanks { SendTaskRank = 0 };

int main(int argc, char *argv[]) {
    float a[DATA_SET_WIDTH][DATA_SET_WIDTH] = {
        { 1.0,  2.0,  3.0,  4.0},
        { 5.0,  6.0,  7.0,  8.0},
        { 9.0, 10.0, 11.0, 12.0},
        {13.0, 14.0, 15.0, 16.0}
    };
    float b[DATA_SET_WIDTH];

    MPI_Init(&argc, &argv);
    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    int cluster_size;
    MPI_Comm_size(MPI_COMM_WORLD, &cluster_size);

    // create contiguous derived data type
    MPI_Datatype rowtype;   // required variable
    MPI_Type_contiguous(DATA_SET_WIDTH, MPI_FLOAT, &rowtype);
    MPI_Type_commit(&rowtype);

    if (cluster_size == DATA_SET_WIDTH) {
        int tag=1;
        if (rank == SendTaskRank) {
            for (int i=0; i<cluster_size; i++)
                MPI_Send(
                    &a[i][0],   // send the i'th row
                    1,          // rows to send
                    rowtype,
                    i,          // destination rank
                    tag,
                    MPI_COMM_WORLD);
        }

        // all tasks receive rowtype data from task 0
        MPI_Status stat;
        MPI_Recv(
            &b,
            1,              // rows received
            rowtype,
            SendTaskRank,   // from master rank
```

```
            tag,
            MPI_COMM_WORLD,
            &stat);
        printf("rank=%d  b={ %4.1f %4.1f %4.1f %4.1f }\n",
                    rank,   b[0], b[1], b[2], b[3]);
    }
    else
        printf("Must specify %d processors. Terminating.\n", DATA_SET_WIDTH);

    // free datatype when done using it
    MPI_Type_free(&rowtype);
    MPI_Finalize();
    return 0;
}
```

## Part 2: Vector Derived Data Types

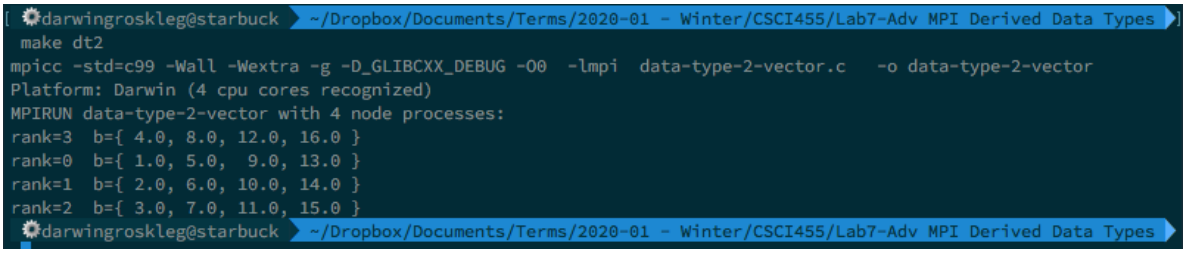


Figure 2: Console output

### data-type-2-vector.c

```c
/* data-type-2-vector.c
 * --------------------
 * Authors: Darwin Jacob Groskleg
 *
 * Vector Derived Data Type:
 *       a data type representing a column of an array and distribute
 *       different columns to all processes.
 */
#include <stdio.h>
#include <mpi.h>

#define DATA_SET_WIDTH 4

enum TaskRanks { SendTaskRank = 0 };

int main(int argc, char *argv[])  {
    float a[DATA_SET_WIDTH][DATA_SET_WIDTH] = {
        { 1.0,  2.0,  3.0,  4.0},
        { 5.0,  6.0,  7.0,  8.0},
        { 9.0, 10.0, 11.0, 12.0},
        {13.0, 14.0, 15.0, 16.0}
    };
    float b[DATA_SET_WIDTH];

    MPI_Init(&argc, &argv);
    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    int cluster_size;
    MPI_Comm_size(MPI_COMM_WORLD, &cluster_size);

    //if (rank == SendTaskRank) {
    //    printf("a={ %.1f",    *(a[0] + 0));
    //    for (int i=0; i<DATA_SET_WIDTH*DATA_SET_WIDTH; i++)
    //        printf(", %.1f", *(a[0] + i));
    //    printf(" }\n");
    //}
    //MPI_Barrier(MPI_COMM_WORLD);

    // create vector derived data type
    MPI_Datatype columntype;   // required variable
    int count       = DATA_SET_WIDTH;   // count of elements of type float
    int blocklength = 1;
    int stride      = DATA_SET_WIDTH;
    // strided contiguous data type
    MPI_Type_vector(count, blocklength, stride, MPI_FLOAT, &columntype);
    MPI_Type_commit(&columntype);

    if (cluster_size == DATA_SET_WIDTH) {
        int tag=1;
        MPI_Status stat;
        // task 0 sends one element of columntype to all tasks
        if (rank == SendTaskRank) {
            for (int i=0; i<cluster_size; i++)
                MPI_Send(
                    &a[0][i],   // send i'th column
```

```
                    1,          // columns being sent
                    columntype,
                    i,          // destination rank
                    tag,
                    MPI_COMM_WORLD);
        }

        // all tasks receive columntype data from task 0
        MPI_Recv(
            &b,
            DATA_SET_WIDTH,          // columns received
            MPI_FLOAT,
            SendTaskRank,
            tag,
            MPI_COMM_WORLD,
            &stat);
        printf("rank=%d  b={ %3.1f, %3.1f, %4.1f, %4.1f }\n",
                    rank,    b[0],  b[1],  b[2],  b[3]);
    }
    else
        printf("Must specify %d processors. Terminating.\n", DATA_SET_WIDTH);

    // free datatype when done using it
    MPI_Type_free(&columntype);
    MPI_Finalize();
    return 0;
}
```

## Part 3: Indexed Derived Data Types

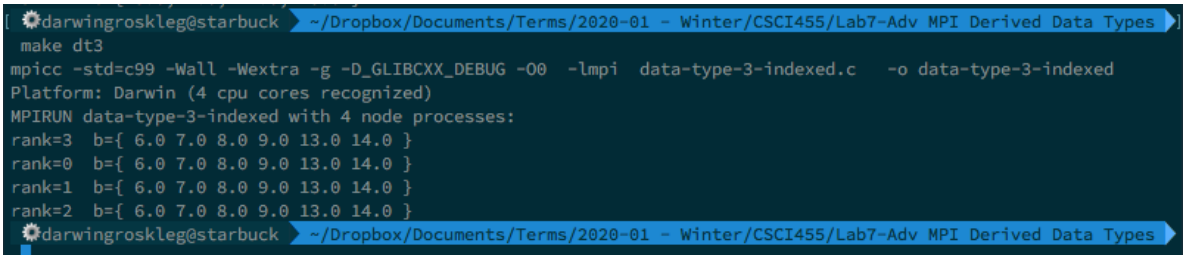


Figure 3: Console output

### data-type-3-indexed.c

```c
/* data-type-3-indexed.c
 * ---------------------
 * Authors: Darwin Jacob Groskleg
 *
 * Index Derived Data Type:
 *      a datatype to extract variable portions of an array and then distribute
 *      to them to all tasks.
 */
#include <stdio.h>
#include "mpi.h"

#define A_SET_SIZE 16
#define B_SET_SIZE 6     // subset of A
#define BLOCKCOUNT 2

enum TaskRanks { SendTaskRank = 0 };

int main(int argc, char *argv[])  {
    float a[A_SET_SIZE] = {
         1.0,  2.0,  3.0,  4.0,
         5.0,  6.0,  7.0,  8.0,
         9.0, 10.0, 11.0, 12.0,
        13.0, 14.0, 15.0, 16.0
    };
    float b[B_SET_SIZE];     // subset of A


    MPI_Init(&argc, &argv);
    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    int cluster_size;
    MPI_Comm_size(MPI_COMM_WORLD, &cluster_size);

    const int blockcount = BLOCKCOUNT;
    int  blocklengths[BLOCKCOUNT] = { 4,  2 };
    int displacements[BLOCKCOUNT] = { 5, 12 };
    blocklengths[0] = 4;
    blocklengths[1] = 2;
    displacements[0] = 5;
    displacements[1] = 12;

    // create indexed derived data type
    MPI_Datatype indexedtype;
    MPI_Type_indexed(blockcount, blocklengths, displacements, MPI_FLOAT,
                     &indexedtype);
    MPI_Type_commit(&indexedtype);

    int tag=1;
    if (rank == SendTaskRank) {
        for (int dest_rank=0; dest_rank<cluster_size; dest_rank++)
            // task 0 sends one element of indexedtype to all tasks
            MPI_Send(
                &a,
                1,      // count of indexedtype
                indexedtype,
```

```
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

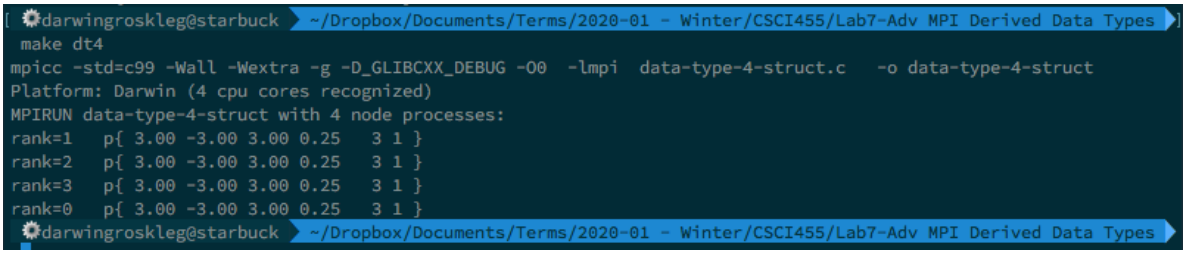


Figure 4: Console output

### data-type-4-struct.c

```c
/* data-type-4-struct.c
 * --------------------
 * Authors: Darwin Jacob Groskleg
 *
 * Struct Derived Data Type:
 *      a data type that represents a particle and distribute an array of such
 *      particles to all processes.
 */
#include <stdio.h>
#include "mpi.h"

#define NELEM 25

enum TaskRanks { SendTaskRank = 0 };

int main(int argc, char *argv[])  {
    typedef struct {
        float x, y, z;
        float velocity;

        int  n, type;
    } Particle;

    MPI_Init(&argc,&argv);
    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    int cluster_size;
    MPI_Comm_size(MPI_COMM_WORLD, &cluster_size);

    // required variables
    int composed_type_count = 2; // float, int
    MPI_Datatype composed_types[2];
    int blockcounts[2]; // block by type

    // MPI_Aint type used to be consistent with syntax of
    // MPI_Type_extent routine
    MPI_Aint    offsets[2];
    MPI_Aint    extent;

    // Descriptions of the 4 MPI_FLOAT fields:
    //      x, y, z, velocity
    offsets[0] = 0;
    composed_types[0] = MPI_FLOAT;
    blockcounts[0] = 4;

    // Descriptions of the 2 MPI_INT fields:
    //      n, type
    // Must first figure offset by getting size of MPI_FLOAT
    // DEPRECATED CALL:
    //      MPI_Type_extent(MPI_FLOAT, &extent);
    MPI_Type_get_extent(MPI_FLOAT, &extent, &extent);
    offsets[1] = 4 * extent;
    composed_types[1] = MPI_INT;
    blockcounts[1] = 2;

```

```
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{ %3.2f %3.2f %3.2f %3.2f   %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  }
```