Προηγμένες αρχιτεκτονικές υπολογιστών και προγραμματισμός παραλλήλων συστημάτων

LAB-08. MPI (Μέρος Δ') – Οι τύποι δεδομένων του MPI

MPI datatypes

- Σκοπός των MPI datatypes είναι η συνεργασία μεταξύ ετερογενών υπολογιστών και αρχιτεκτονικών.
- Η χρήση μη συνεχών θέσεων μνήμης. Δομές δεδομένων και πίνακες με μη μοναδιαίο βήμα.
- Ο χρήστης μπορεί να ορίσει δικά του datatypes κατά το χρόνο εκτέλεσης (runtime).
- Αυτά λέγονται παραγόμενα (derived) datatypes.

Τύποι δεδομένων

- Βασικοί (elementary):
  - Ορισμένοι στη γλώσσα MPI (πχ., `MPI_INT` ή `MPI_DOUBLE`).
- Διανυσματικοί (vector):
  - Σταθερή απόσταση μεταξύ στοιχείων = «βήμα» (stride).
- Συνεχείς (contiguous):
  - Διάνυσμα με βήμα 1.
- Hvector:
  - Διάνυσμα με βήμα σε bytes
- Δείκτες (Indexed):
- Πίνακες δεικτών (για scatter/gather)
  - Hindexed:
  - Indexed, με δείκτες σε bytes
  - Struct:
  - Γενικοί μικτοί τύποι (για C structs κλπ.)

Βασικοί τύποι C

Τύπος MPI	Τύπος C
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNISGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	
MPI_PACKED	

MPI_Type_vector

- Δημιουργία ενός διανυσματικού datatype με δοθέν βήμα stride.
- Ο νέος τύπος προκύπτει από αντιγραφή ενός υπάρχοντος τύπου δεδομένων (old_datatype) πολλές φορές μέσα σε ένα block.
- Ο νέος τύπος δεδομένων (new_datatype) που θα δημιουργηθεί δύναται να περιέχει πολλαπλά τέτοια blocks.

MPI_Type_vector

- **int MPI_Type_vector(int block_count, int blocklength, int stride, MPI_Datatype oldtype, MPI_Datatype * newtype)**
  - block_count: Το πλήθος των block που θα περιέχει ο νέος τύπος δεδομένων.
  - blocklength: Το μέγεθος κάθε block.
    - Όλα τα blocks θα έχουν το ίδιο μέγεθος.
  - stride: Το πλήθος των στοιχείων που μεσολαβούν μεταξύ των αρχών δύο διαδοχικών blocks.
  - oldtype: Ο παλιός datatype που θα αντιγραφεί.
  - newtype: Επιστρέφεται ο νέος τύπος δεδομένων.

MPI_Type_commit

- Υποβάλλει ένα datatype.
- **`int MPI_Type_commit(MPI_Datatype * dtype)`**
  - dtype: Ο τύπος δεδομένων που θα υποβληθεί.

Παράδειγμα MPI_Type_vector

28	29	30	31	32	33	34
21	22	23	24	25	26	27
14	15	16	17	18	19	20
7	8	9	10	11	12	13
0	1	2	3	4	5	6

Παράδειγμα MPI_Type_vector

- Για να ορίσουμε αυτή τη γραμμή στη C χρησιμοποιούμε
  - `MPI_Type_vector(count, blocklen, stride, oldtype, &newtyp);`
  - `MPI_Type_commit(&newtyp);`

- Ο ακριβής κώδικας είναι
  - `MPI_Type_vector(5, 1, 7, MPI_INT, &newtyp);`
  - `MPI_Type_commit(&newtyp);`

Παράδειγμα MPI_Type_vector

```
int main(int argc, char* argv[]) {
    int size, rank;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    if (size != 2) { MPI_Abort(MPI_COMM_WORLD, EXIT_FAILURE); }
    if (rank == 0) {
        MPI_Datatype newtype;
        MPI_Type_vector(5, 1, 7, MPI_INT, &newtype);
        MPI_Type_commit(&newtype);

        int array2d[5][7] = {28, 29, 30, 31, 32, 33, 34, 21, 22, 23, 24, 25, 26, 27,
        14, 15, 16, 17, 18, 19, 20, 7,  8,  9, 10, 11, 12, 13, 0, 1, 2, 3, 4, 5, 6};

        MPI_Send(&array2d[0][0], 1, newtype, 1, 0, MPI_COMM_WORLD);
    } else {
        int received[5];
        MPI_Recv(&received, 5, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    }
    MPI_Finalize();
    return 0;
}
```

MPI_Type_struct

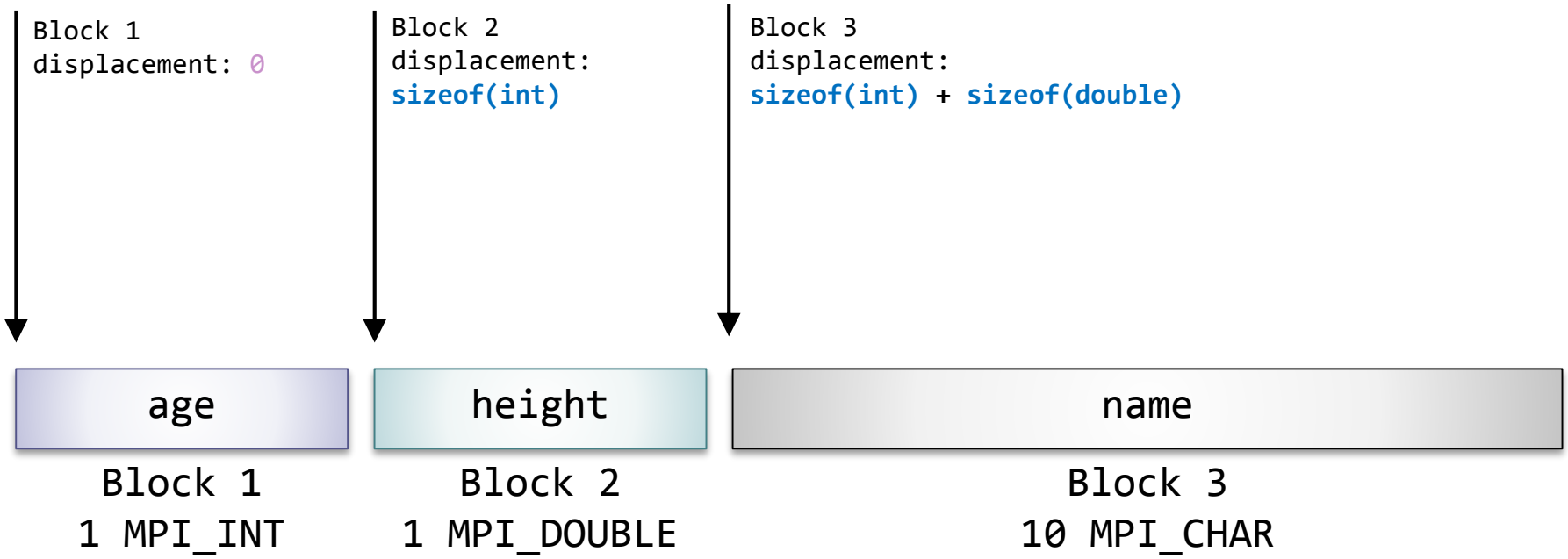
- Δημιουργία datatype τύπου δομής.
- **int MPI_Type_struct(int count, int* array_of_blocklengths, MPI_Aint * array_of_displacements, MPI_Datatype * array_of_types, MPI_Datatype * newtype)**
  - count: Το πλήθος των block που θα περιέχει ο νέος τύπος δεδομένων.
  - Επίσης, ταυτίζεται με το πλήθος των στοιχείων που περιέχονται στους πίνακες array_of_types, array_of_displacements και array_of_blocklengths

MPI_Type_struct

- `array_of_types`: Πίνακας στον οποίο δηλώνεται το πλήθος των στοιχείων σε κάθε block.
- `array_of_displacements`: Πίνακας στον οποίο δηλώνεται η μετατόπιση κάθε block (σε bytes).
- `array_of_blocklengths`: Πίνακας στον οποίο δηλώνεται ο τύπος δεδομένων των στοιχείων σε κάθε block.
- `newtype`: Επιστρέφεται ο νέος τύπος δεδομένων.

MPI_Type_struct - Παράδειγμα 1

```
struct person_t {
        int age;
        double height;
        char name[10];
};
```



MPI_Type_struct - Παράδειγμα 1

```
int main(int argc, char* argv[]) {
    int size, rank;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    if (size != 2) {
        MPI_Abort(MPI_COMM_WORLD, EXIT_FAILURE);
    }

    /// Create the datatype
    MPI_Datatype person_type;

    int lengths[3] = { 1, 1, 10 };
    MPI_Aint displacements[3] = { 0, sizeof(int), sizeof(int) + sizeof(double) };
    MPI_Datatype types[3] = { MPI_INT, MPI_DOUBLE, MPI_CHAR };

    MPI_Type_create_struct(3, lengths, displacements, types, &person_type);
    MPI_Type_commit(&person_type);
```

MPI_Type_struct - Παράδειγμα 1

```
    if (rank == 0) {
        struct person_t buffer;
        buffer.age = 20;
        buffer.height = 1.83;
        strncpy(buffer.name, "Tom", 9);
        buffer.name[9] = '\0';
        printf("MPI process %d sends person:\n\t- age = %d\n\t- height = %f\n\t-
              name = %s\n", rank, buffer.age, buffer.height, buffer.name);
        MPI_Send(&buffer, 1, person_type, 1, 0, MPI_COMM_WORLD);

    } else if (rank == 1) {
        struct person_t received;
        MPI_Recv(&received, 1, person_type, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        printf("MPI process %d received person:\n\t- age = %d\n\t- height = %f\n\t-
              name = %s\n", rank, received.age, received.height, received.name);
    }

    MPI_Finalize();

    return EXIT_SUCCESS;
}
```

MPI_Type_struct - Παράδειγμα 2

```
struct cmd_t {
     char display[50];
     int maxiter;
     double xmin, ymin;
     double xmax, ymax;
     int width;
     int height;
} cmd;

int main(int argc, char* argv[]) {
   int size, rank;
   MPI_Init(&argc, &argv);
   MPI_Comm_size(MPI_COMM_WORLD, &size);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);

   if (size != 2) {
      MPI_Abort(MPI_COMM_WORLD, EXIT_FAILURE);
   }
```

MPI_Type_struct - Παράδειγμα 2

```
/// Create the datatype
MPI_Datatype cmd_type;
int lengths[4] = { 50, 1, 4, 2 };
MPI_Aint displacements[4];

MPI_Address( &cmd.display, &displacements[0] );
MPI_Address( &cmd.maxiter, &displacements[1] );
MPI_Address( &cmd.xmin, &displacements[2] );
MPI_Address( &cmd.width, &displacements[3] );
for (int i = 3; i >= 0; i--) {
    displacements[i] -= displacements[0];
}

MPI_Datatype types[4]={MPI_CHAR, MPI_INT, MPI_DOUBLE, MPI_INT};
MPI_Type_create_struct(4, lengths, displacements, types,
    &cmd_type);
MPI_Type_commit(&cmd_type);
```

MPI_Type_struct - Παράδειγμα 2

```
    /// Get my rank and do the corresponding job
    if (rank == 0) {
        /// Send the message
        strncpy(cmd.display, "Tom & Jerry", 49);
        cmd.display[49] = '\0';
        cmd.maxiter;
        cmd.xmin=1.5; cmd.ymin=2.5; cmd.xmax = 3.5; cmd.ymax = 5.5;
        cmd.width = 2; cmd.height = 3;

        MPI_Send(&cmd, 1, cmd_type, 1, 0, MPI_COMM_WORLD);
    } else if (rank == 1) {
        struct cmd_t received;
        MPI_Recv(&received, 1, cmd_type, 0, 0, MPI_COMM_WORLD,
                 MPI_STATUS_IGNORE);
    }

    MPI_Finalize();
    return EXIT_SUCCESS;
}
```

Interleaving

- Παραδείγματα αποστολής:
  - 0-3, 8-11, 16-19, 24-27 στη διεργασία 0 και
  - 4-7, 12-15, 20-23, 28-31 στη διεργασία 1.

0	8	16	24	32
1	9	17	25	33
2	10	18	26	34
3	11	19	27	35
4	12	20	28	36
5	13	21	29	37
6	14	22	30	38
7	15	23	31	39

Interleaving (2)

- Αρχικά ορίζεται διάνυσμα για τα δεδομένα της διεργασίας ο.
  - `MPI_Type_vector(4,4,8, MPI_DOUBLE, &vec);`

- Εν συνεχεία, κατασκευάζεται δομή από το προαναφερθέν διάνυσμα και το διάνυσμα padding.

```
lengths[0] = 1;   lengths[1] = 1;
types[0]   = vec; types[1]   = MPI_UB;
displ[0]   = 0;   displ[1]  = sizeof(double);
MPI_Type_struct(2,blen,displ, types, &block);
```

MPI_Scatterv

- Τεμαχίζει ένα buffer και τον μοιράζει στις διεργασίες ενός communicator.
- **int MPI_Scatterv (const void *sendbuf, const int *sendcounts, const int *displs, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)**

MPI_Scatterv

- `sendbuf`: διεύθυνση του send buffer.
  `sendcounts`: πίνακας που ορίζει το πλήθος των στοιχείων που θα στείλουμε σε κάθε διεργασία.
  `displs`: πίνακας του οποίου το στοιχείο `i` ορίζει τη μετατόπιση του δεδομένου που θα σταλεί στη διεργασία `i`.
  `sendtype`: datatype των στοιχείων του `sendbuf`.
  `recvcount`: = πλήθος στοιχείων στον receive buffer.
  `recvtype`: = datatype των στοιχείων του receive buffer.
  `root`: rank της διεργασίας αποστολέα.
  comm: communicator.
- `recvbuf`: διεύθυνση του receive buffer.

Interleaving με την MPI_Scatterv

- scdispl[0] = 0;
- scdispl[1] = 4;
- scdispl[2] = 32;
- scdispl[3] = 36;
- MPI_Scatterv(sendbuf, sendcounts, scdispl, block, recvbuf, nx * ny, MPI_DOUBLE, 0, MPI_COMM_WORLD);