

Средства и системы параллельного программирования

#10. Производные типы данных MPI

Структуры в Си

```
struct S1 {  
    char c;  
    int i;  
    double v;  
};
```

Какой размер структуры (sizeof(S1)) ?

offsetof macro

```
#define offsetof(a,b) ((int)(amp(((a*)(0))->b)))
```

Создание “структурного” типа данных в MPI

```
int MPI_Type_create_struct(int count,  
const int array_of_blocklengths[], const MPI_Aint  
array_of_displacements[], const MPI_Datatype array_of_types[],  
MPI_Datatype *newtype)
```

`count` - число блоков для отправки

`array_of_blocklengths` - длина каждого блока [count]

`array_of_displacements` - смещение каждого блока [count]

`array_of_types` - типы данных для каждого блока [count]

`newtype` - указатель на новый, производный тип

Смещение типа в MPI

MPI_Aint - специальный тип данных MPI для представления адреса

`int MPI_Get_address(const void *location, MPI_Aint *address)` -

записать в `address` адрес места в памяти `location`

`int MPI_Aint_diff(MPI_Aint addr_1, MPI_Aint addr_2)` -

получить смещение `addr_1` относительно `addr_2`

Сценарий работы с производными типами

```
int MPI_Type_commit(MPI_Datatype* datatype);
```

перед использованием производного типа данных в коммуникациях (Send,Recv, и.т.д) обязательно зарегистрировать тип `datatype`, созданный ранее с помощью команд создания типов

```
int MPI_Type_free(MPI_Datatype *datatype)
```

освобождение типа `datatype`, больше нет возможности его использовать в коммуникациях

Вывод размера типа

```
int MPI_Type_size(MPI_Datatype datatype, int *size)
```

Положить в *size* значение размера (в байтах), занимаемый производным типом данных *datatype*.

*Стоит воспринимать *size* как размер сообщения при передаче этого типа.*

Протяжённость типа

```
int MPI_Type_extent(MPI_Datatype datatype, MPI_Aint *  
extent)
```

Вывести в `extent` протяжённость типа `datatype`

`Extent` стоит воспринимать как размер структуры в С-представлении

Карта типа

Typemap = $\{(type_0, disp_0), \dots, (type_{n-1}, disp_{n-1})\}$

Lower bound = $\min_j disp_j$

Upper bound = $\max_j (disp_j + sizeof(type_j)) + eps$

Extent = Upper bound - Lower bound

“Продолжительный” тип данных

```
int MPI_Type_contiguous(int count, MPI_Datatype old_datatype,  
MPI_Datatype* new_datatype);
```

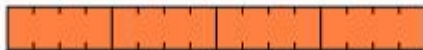
Создание нового типа данных `new_datatype`, получаемого путём взятия подряд идущих в памяти `count` элементов типа данных `old_datatype`

`oldtype = MPI_REAL`



`count = 4`

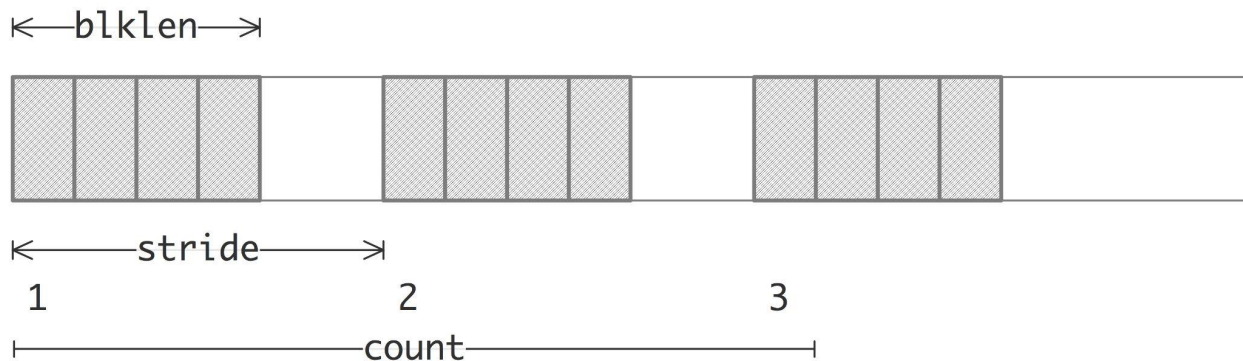
`newtype`



Векторный тип данных

```
int MPI_Type_vector(int count, int blocklength, int stride,  
MPI_Datatype oldtype, MPI_Datatype *newtype)
```

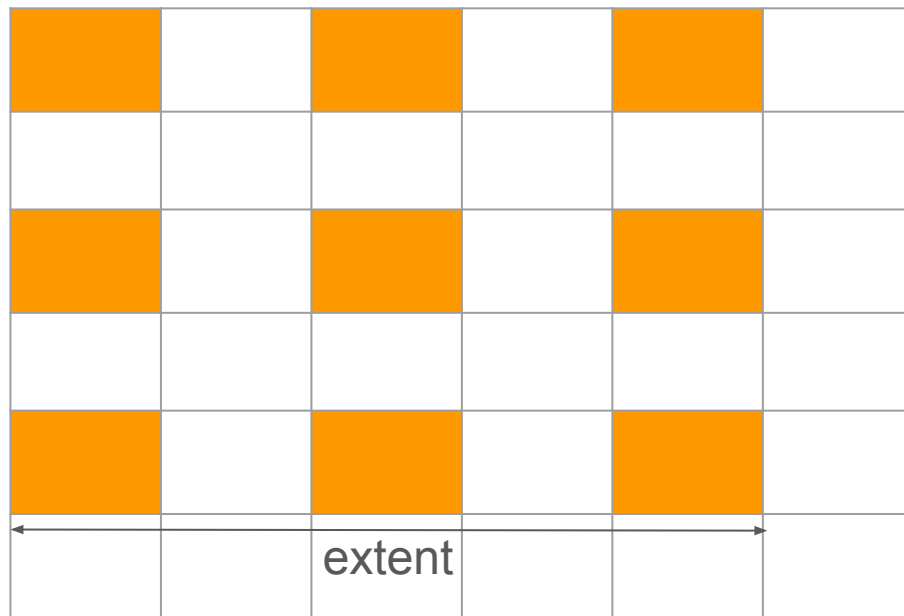
Создание нового типа данных `newtype`, который получается из `count` блоков размера `blocklength` каждый, причём начала соседних блоков находятся на расстоянии `stride` элементов типа `oldtype` друг от друга



Векторный тип данных (2)

```
int MPI_Type_create_hvector(int count,  
                             int blocklength,  
                             MPI_Aint stride,  
                             MPI_Datatype oldtype,  
                             MPI_Datatype *newtype)
```

То же самое, но *stride* задан не по количеству элементов, а в байтах



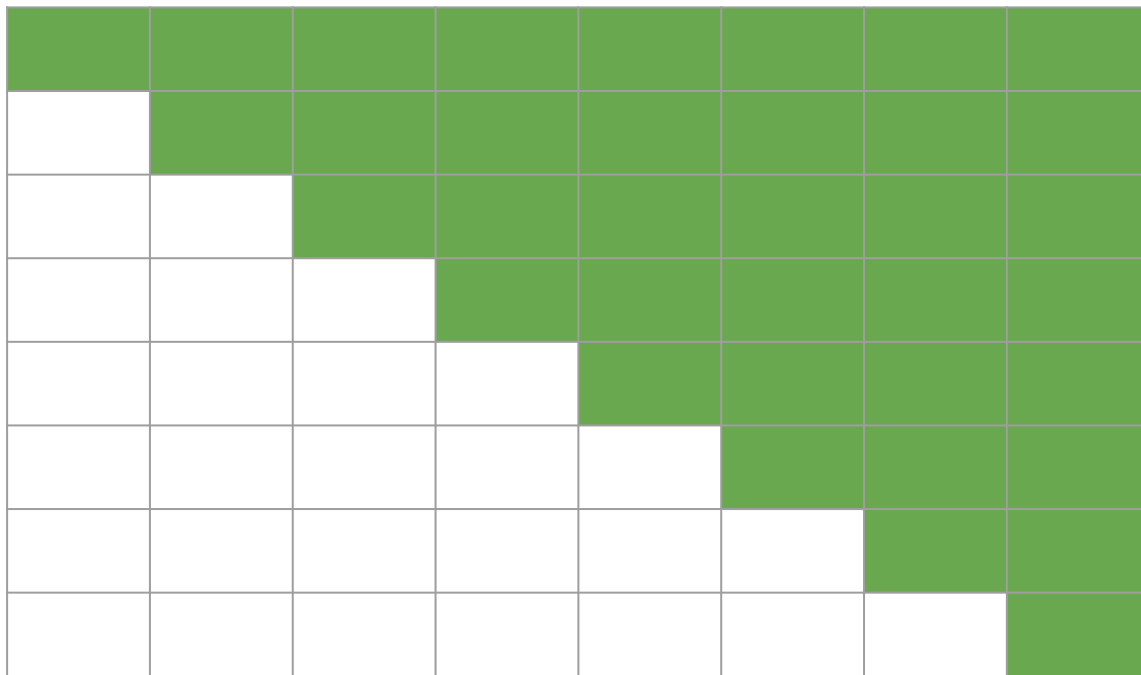
Hvector полезен в случаях, когда невозможно получить правило формирования stride

Индексированный тип

```
int MPI_Type_indexed(int block_count, const int  
block_lengths[], const int displacements[], MPI_Datatype  
old_datatype, MPI_Datatype* new_datatype);
```

Создать новый тип `new_datatype`, составленный из `block_count` блоков, причём `blocki` состоит из `block_lengthsi` элементов типа `old_datatype` и отстоят от начала типа на расстояние `displacementsi` элементов типа `old_datatype`

Пример: верхнетреугольная матрица 8*8



“Изменение размера” нового типа

```
int MPI_Type_create_resized(MPI_Datatype oldtype, MPI_Aint lb, MPI_Aint extent,  
MPI_Datatype *newtype)
```

На основе старого типа `oldtype` выставить в новом типе `newtype` нижнюю границу `lb` и протяжённость типа `extent`

Зачем?

В коллективных операциях новый элемент для отправки след.процессу берётся только следом за UB. Иногда нужно отправить данные, лежащие между LB и UB, поэтому нужно явно прописать меньший extent (см.пример 09)

Задание

Модифицировать задание 7 (уравнение Лапласа, $f = 0$) для 3D-области

$$\frac{-u_{i-1,j,k} + 2u_{ijk} - u_{i+1,j,k}}{h_x^2} + \frac{-u_{i,j-1,k} + 2u_{ijk} - u_{i,j+1,k}}{h_y^2} + \frac{-u_{i,j,k-1} + 2u_{ijk} - u_{i,j,k+1}}{h_z^2} = f(x_i, y_j, z_k)$$

Или, используя формулировку из 7 задания:

$$f_{i,j,k}^{(t+1)} = \frac{1}{6} (f_{i,j,k-1}^{(t)} + f_{i,j,k+1}^{(t)} + f_{i,j-1,k}^{(t)} + f_{i,j+1,k}^{(t)} + f_{i-1,j,k}^{(t)} + f_{i+1,j,k}^{(t)})$$

Задание (2)

Метод решения:

Построить сетку (на каждом процессе выделить массив размера, равного числу элементов в подобласти). Будем использовать **блочные** 3D-подобласти (каждый процесс работает над своей частью сетки $\{N_x, N_y, N_z\}$, причём $1 \leq N_x \leq N$, $1 \leq N_y \leq N$, $1 \leq N_z \leq N$, где N - размер всей сетки).

Рекомендуется использовать виртуальные топологии для организации обменов

Инициализировать начальное значение f случайным значением в каждой области сетки

До предустановленного числа итераций n_iter выполнять вычисления согласно методу Якоби

На последней итерации посчитать общую норму разности между решениями на двух соседних шагах времени.

Задание (3)

Требования к решению:

Запрещается хранить массив, соответствующий полной сетке, на одном процессе (за исключением запуска на 1 процессе)

Для коммуникации разрешается использовать любые виды обменов, главное требование - **пользоваться производными типами данных** для отправки сообщений. Для приёма (recv) сообщений разрешается воспринимать сообщение как набор из Q подряд идущих элементов

Можно предполагать, что размер сетки N - степень двойки. Сетка кубическая ($N*N*N$)

Произвести запуски на **Polus** ([через mpisubmit.pl](https://mpisubmit.pl) !)

Для фиксированного большого размера сетки произвести запуски при числе процессов $P = \{1, 2, 4, 6, 8, 12, 16\}$, нарисовать $T(P)$. На 8, 12, 16 процессах программа должна работать для случаев, когда число процессов в каждом измерении > 1 ($8 = 2*2*2$, $12 = 2*3*2$, $16 = 2*4*2$)

Дедлайн: 23.12, 24.12 **(ИЗМЕНЕНО)**