Собственные типы данных

НОЦ МФТИ 2013

Введение

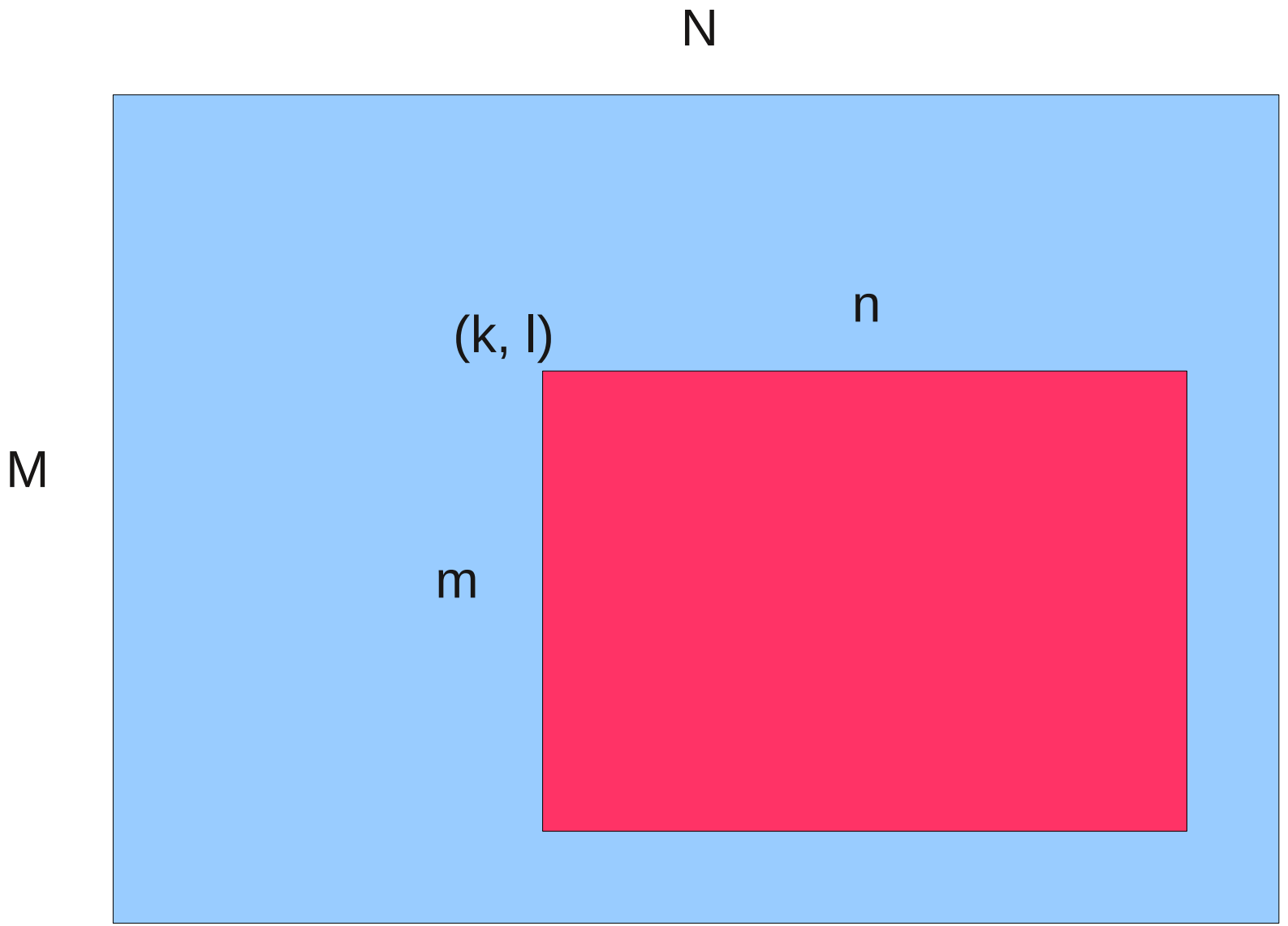
- Ранее мы имели дело с пересылкой массивов стандартных типов, расположенных в памяти последовательно
- Иногда требуется пересылка данных не находящихся в памяти последовательно или специальных структур
- Данная лекция дает обзор возможностей MPI решения таких задач

Пример: пересылка подматрицы

- Имеется матрица в языке C в виде двумерного массива размером M x N
- Необходимо переслать подматрицу размера m x n
- Подматрица отсчитывается от ячейки с номером (k, l)
- Можно использовать цикл

```
for (i=0; i<n; ++i) {
   MPI_Send(&a[k+i][l], m, MPI_DOUBLE,  dest, tag,
   MPI_COMM_WORLD);
}
```

Пример: пересылка подматрицы



Пример: пересылка подматрицы

- Преимущества такого подхода — простота реализации
- Недостатки — множество сообщений
- Каждое сообщение, независимо от размера имеет оверхед
- Если заменить одно большое сообщение множеством маленьких, то производительность очень сильно уменьшится
- Если не часто встречается в программе, то такое решение может быть приемлемо
- Можно использовать буферизацию данных
- Существуют специальные инструменты в MPI для решения таких задач

Буферизация

- Если данные расположены непоследовательно в памяти можно предварительно скопировать их в буфер
- Для нашего примера:

```
p = &buffer;
for (i=0; i<n; ++i) {
   for(j=0; j<m; ++j) {
     *(p++) = a[i + k][j + l];
   }
}
MPI_Send(p, n*m, MPI_DOUBLE, dest, tag,
   MPI_COMM_WORLD)
```

Недостатки такого подхода

- Накладные расходы на память и на копирование данных
- Возможна пересылка только одного типа данных
- Ухудшает читаемость кода
- При попытках пересылать одни типы данных через другие возможны ошибки

Буферизация в MPI: MPI_Pack

- Используется для пересылок наборов различных данных расположенных в памяти не последовательно
- Заполняет буфер правильным для MPI образом и дает аргументы для MPI_Send
- Копирует данные в буфер и при необходимости транслирует их во внутреннее представление MPI
- После копирования данных в буфер можно пересылать их с типом MPI_PACKED

Пример: упаковка подматицы

```
count = 0
for (i=0; i<n; ++i) {
  MPI_Pack(&a[k+i][l], m, MPI_DOUBLE, buffer, bufsize,
  &count, MPI_COMM_WORLD);
}
MPI_Send(buffer, count, MPI_PACKED, dest, tag,
  MPI_COMM_WORLD);
```

- count изначально выставлен в 0, что говорит о начале заполнения буфера
- Каждый вызов обновляет значение count и конечное значение используется при пересылках

Буферизация в MPI

- MPI_PACKED — специальный тип данных, говорящий о том, что буфер упакован
- MPI_Unpack — используется для распаковки принятых данных
- После упаковки данные могут занимать больше места из-за перевода их в другое представление
- Для определения размера типа в упакованном виде можно использовать MPI_Pack_size

MPI_Pack_size

```
int MPI_Pack_size(  int incount,  MPI_Datatype datatype,
MPI_Comm comm,  int *size);
```

```
int bufsize = 0;
MPI_Pack_size(10, MPI_DOUBLE,
MPI_COMM_WORLD, &bufsize);
buffer = (char *) malloc((unsigned) bufsize);
```

Позволяет узнать размер данных после упаковки для выделения буфера

Производные типы

- Замена MPI_Pack
- Позволяет выполнять упаковку данных на-лету, без выделения дополнительной памяти
- Позволяет сэкономить процессорное время на упаковку данных в память и их трансляцию в понятную для MPI форму
- Позволяет читать и писать непосредственно в рабочие области памяти

Производные типы данных

- Contiguous — несколько копий указанного типа данных
- Vector — несколько копий данных с отступами между блоками
- Indexed — массивом задается мапинг на новый тип
- Struct — мапинг на области памяти, в частности на структуры языка C

Последовательность работы с типами

- Создание типа с помощью функций MPI:
  - MPI_Type_contiguous, MPI_Type_vector,
  - MPI_Type_struct, MPI_Type_indexed,
  - MPI_Type_hvector, MPI_Type_hindexed
- Определить тип (commit) для создания внутренних буфером MPI для данного типа
- Использование своего типа в функциях приема/передачи и т. д.
- Очистить тип данных после использования:
  - MPI_Type_Free(newtype, ierr)

MPI_Contiguous

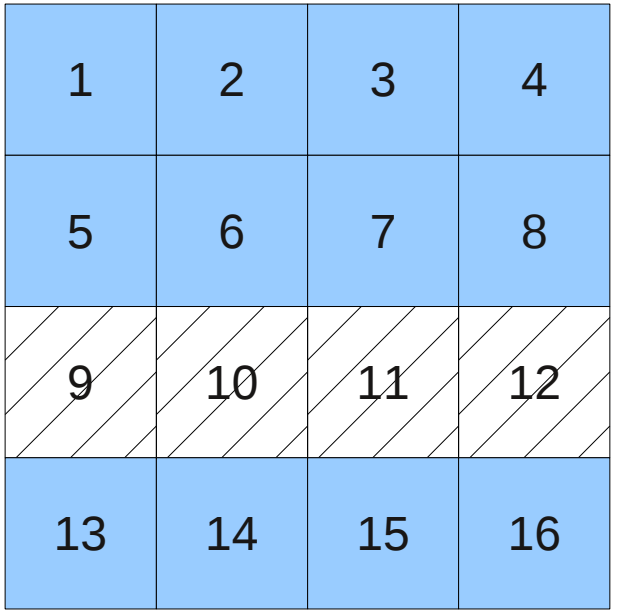
- Простейший тип, состоящий из наборов элементов заданного типа последовательно идущих в памяти
- C:

  int MPI_Type_contiguous (int count,MPI_datatype oldtype,MPI_Datatype *newtype)
- Fortran:
  - MPI_TYPE_CONTIGUOUS(COUNT,OLDTYPE,NEWTYPE,IERROR)
  - INTEGER COUNT, OLDTYPE, NEWTYPE, IERROR

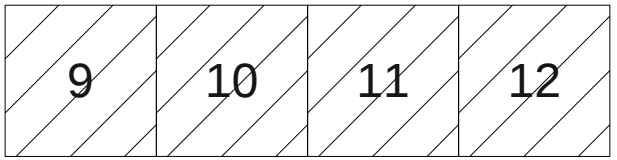
MPI_Type_contiguous

MPI_Type_contiguous (count,oldtype,&newtype)

count = 4;



a[4][4];



1 элемент rowtype &a[2][0]

Пример

```
#include <stdio.h>
#include<mpi.h>
int main(int argc, char *argv[]) {
        int rank;
        struct {
                int x;
                int y;
                int z;
        } point;
        MPI_Datatype ptype;
        MPI_Init(&argc,&argv);
        MPI_Comm_rank(MPI_COMM_WORLD,&rank);
        MPI_Type_contiguous(3,MPI_INT,&ptype);
        MPI_Type_commit(&ptype);
        if(rank==3){
                point.x=15; point.y=23; point.z=6;
                MPI_Send(&point,1,ptype,1,52,MPI_COMM_WORLD);
        } else if(rank==1) {
                MPI_Recv(&point,1,ptype,3,52,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
                printf("P:%d received coords are (%d,%d,%d) \n",rank,point.x,point.y,point.z);
        }
        MPI_Finalize();
}
```

```
P:1 received coords are (15,23,6)
```

MPI_Vector

- Пользователь задает расположение элементов старого типа в памяти
- C:

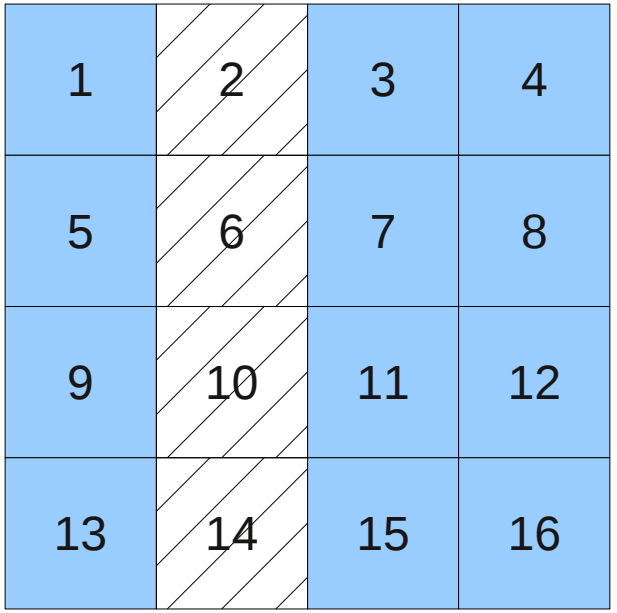
  int MPI_Type_vector(int count, int blocklength, int stride,MPI_Datatype oldtype, MPI_Datatype *newtype)
- Fortran:

  CALL MPI_TYPE_VECTOR(COUNT, BLOCKLENGTH, STRIDE,OLDTYPE, NEWTYPE, IERROR)
- Новый тип состоит из count блоков каждый из которых состоит из blocklength элементов старого типа
- Отступы меду типами задаются stride

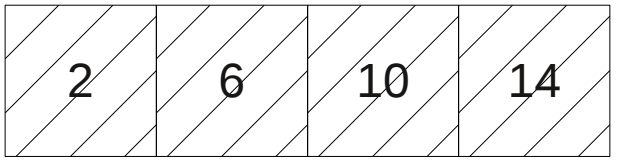
MPI_Type_vector

MPI_Type_vector (count,blocklength,stride,oldtype,&newtype)

count = 4;
blocklength = 1;
stride = 4;



a[4][4];



1 элемент newtype &a[0][1]

Пример

```
#include <mpi.h>
#include <math.h>
#include <stdio.h>
int main(int argc, char *argv[]) {
      int rank,i,j;
      MPI_Status status;
      double x[4][8];
      MPI_Datatype coltype;
      MPI_Init(&argc, &argv);
      MPI_Comm_rank(MPI_COMM_WORLD,&rank);
      MPI_Type_vector(4,1,8,MPI_DOUBLE,&coltype);
      MPI_Type_commit(&coltype);
      if(rank==3){
            for(i=0;i<4;++i)
            for(j=0;j<8;++j) x[i][j]=pow(10.0,i+1)+j;
            MPI_Send(&x[0][7],1,coltype,1,52,MPI_COMM_WORLD);
      } else if(rank==1) {
            MPI_Recv(&x[0][2],1,coltype,3,52,MPI_COMM_WORLD,&status);
            for(i=0;i<4;++i)printf("P:%d my x[%d][2]=%1f\n",rank,i,x[i][2]);
      }
      MPI_Finalize();
}
```

```
P:1 my x[0][2]=17.000000
P:1 my x[1][2]=107.000000
P:1 my x[2][2]=1007.000000
P:1 my x[3][2]=10007.000000
```

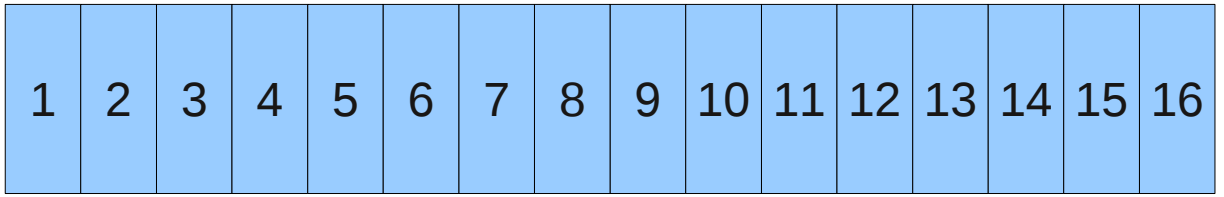
MPI_Indexed

- Позволяют задавать блоки различной длины через различные отступы
- int MPI_Type_indexed(int count,int blocklens[], int indices[],MPI_Datatype old_type, MPI_Datatype *newtype)
- count - число блоков, а также размер входных массивов
- blocklens — число элементов в каждом блоке
- indices — отступ у каждого следующего типа

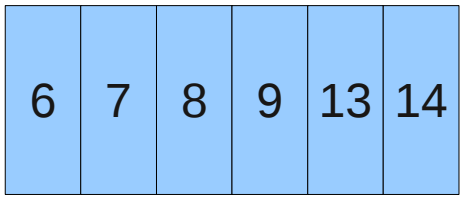
MPI_Type_indexed

MPI_Type_indexed (count,blocklens[],offsets[],old_type,&newtype)

count = 2; blocklengths[0] = 4; blocklengths[1] = 2;
displasements[0] = 5; displaysments[1] = 12;



a[16];



newtype a[0];

Пример

```
#include "mpi.h"
#include <stdio.h>
#define NELEMENTS 6

int main(int argc,char *argv[]) {
        int numtasks, rank, source=0, dest, tag=1, i;
        int blocklengths[2], displacements[2];
        float a[16] =
                {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[NELEMENTS];
        MPI_Status stat;
        MPI_Datatype indextype;
        MPI_Init(&argc,&argv);
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
        MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
        blocklengths[0] = 4;
        blocklengths[1] = 2;
        displacements[0] = 5;
        displacements[1] = 12;
        MPI_Type_indexed(2, blocklengths, displacements, MPI_FLOAT, &indextype);
        MPI_Type_commit(&indextype);
```

Пример

```
      if (rank == 0) {
       for (i=1; i<numtasks; i++)
                MPI_Send(a, 1, indextype, i, tag, MPI_COMM_WORLD);
       } else {
            MPI_Recv(b, NELEMENTS, MPI_FLOAT, source, tag, MPI_COMM_WORLD,
&stat);
      printf("rank= %d  b= %3.1f %3.1f %3.1f %3.1f %3.1f %3.1f\n",
      rank,b[0],b[1],b[2],b[3],b[4],b[5]);
  }
MPI_Type_free(&indextype);
MPI_Finalize();
}
```

```
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
rank= 3  b= 6.0 7.0 8.0 9.0 13.0 14.0
```

MPI_Extent

- Используется при операциях над типами в MPI
- Показывает на сколько байт тип распределен в памяти
- При этом фактический размер типа может быть меньше
- C:

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

- Fortran:

```
CALL MPI_TYPE_EXTENT (DATATYPE, EXTENT,
IERROR)
```

MPI_Struct

- Используется для задания гетерогенных типов
- Наиболее общий тип
- Используется со структурами языка C

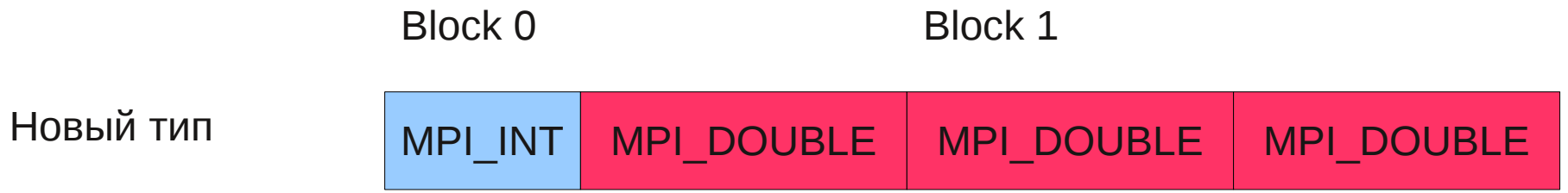
```
int MPI_Type_struct (int count, int
*array_of_blocklengths, MPI_Aint
*array_of_displacements, MPI_Datatype
*array_of_types, MPI_Datatype *newtype)
```

MPI_Struct

MPI_INT

MPI_DOUBLE

```
struct {
        int type;
        double x, y, z;
} point;
```



- count = 2
- array_of_blocklengths = {1,3}
- array_of_types = {MPI_INT, MPI_DOUBLE}
- array_of_displacements = {0, extent(MPI_INT)}

Пример

```
#include <stdio.h>
#include<mpi.h>
int main(int argc, char *argv[]) {
	int rank,i;
	MPI_Status status;
	struct {
		int num;
		float x;
		double data[4];
	} a;
	int blocklengths[3]={1,1,4};
	MPI_Datatype types[3]={MPI_INT, MPI_FLOAT, MPI_DOUBLE};
	MPI_Aint displacements[3];
	MPI_Datatype restype;
	MPI_Aint intex, floatex;
	MPI_Init(&argc, &argv);
	MPI_Comm_rank(MPI_COMM_WORLD, &rank);
	MPI_Type_extent(MPI_INT, &intex);MPI_Type_extent(MPI_FLOAT,
&floatex);
	displacements[0] = (MPI_Aint) 0; displacemtnts[1] = intex;
	displacements[2] = intex+floatex;
	MPI_Type_struct(3, blocklengths, displacements, types, &restype);
```

Пример

```
    MPI_Type_commit(&restype);
    If (rank==3){
        a.num=6; a.x=3.14; for(i=0;i 4;++i) a.data[i]=(double) i;
        MPI_Send(&a,1,restype,1,52,MPI_COMM_WORLD);
    } else if(rank==1) {
        MPI_Recv(&a,1,restype,3,52,MPI_COMM_WORLD,&status);
        printf("P:%d my a is %d %f %lf %lf %lf %lf\n",
        rank,a.num,a.x,a.data[0],a.data[1],a.data[2],a.data[3]);
    }
    MPI_Finalize();
}
```

```
P:1 my a is 6 3.140000 0.000000 1.000000 2.000000 3.000002
```

Вопросы.