Технологии параллельного программирования на C++

Семинар 10
MPI
Собственные типы данных

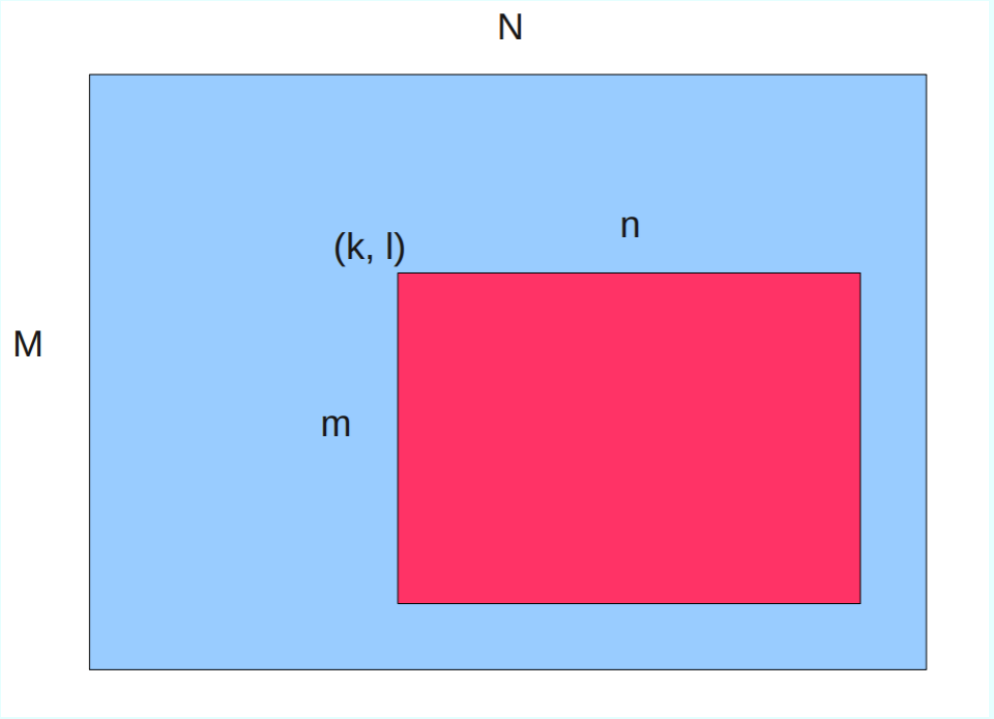
Пересылка разнотипных данных

- Упаковка данных
- Производные типы данных

Пример. Пересылка подматрицы

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.

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



Пример. Пересылка подматрицы. Способ 1

Можно использовать цикл:

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

Преимущество: простота реализации

Недостатки: множество сообщений

- Если заменить одно большое сообщение множеством маленьких, то производительность очень сильно уменьшится
- Если не часто встречается в программе, то такое решение может быть приемлемо

Пример. Пересылка подматрицы. Способ 2 Буферизация

Если данные расположены непоследовательно в памяти можно предварительно скопировать их в буфер:

```
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)
```

Преимущество: Посылка одного сообщения

Недостатки:

- Накладные расходы на память и на копирование данных
- Возможна пересылка только одного типа данных

Пример. Пересылка подматрицы. Способ 3 Буферизация в MPI

Если данные расположены непоследовательно в памяти можно предварительно скопировать их в буфер:

```
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

Преимущество: Могут быть упакованы разные типы данных

Недостатки:

- После упаковки данные могут занимать больше места из-за перевода их в другое представление

Формирование сообщений при помощи упаковки и распаковки данных

- Используется для пересылок наборов различных данных расположенных в памяти не последовательно
- Заполняет буфер правильным для MPI образом и дает аргументы для функций передачи сообщений (например, 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

Упаковка данных в буфер

- **int MPI_Pack (void *data, int count, MPI_Datatype type, void *buf, int bufsize, int *bufpos, MPI_Comm comm)**

  data – буфер памяти с элементами для упаковки,

  count – количество элементов в буфере,

  type – тип данных для упаковываемых элементов,

  buf - буфер памяти для упаковки,

  bufsize – размер буфера в байтах,

  bufpos – позиция для начала записи в буфер (в байтах от начала буфера),

  comm - коммуникатор для упакованного сообщения.

Определение необходимого размера буфера для упаковки

- **int MPI_Pack_size (int count, MPI_Datatype type, MPI_Comm comm, int *size)**

В параметре **size** указывает необходимый размер буфера для упаковки **count** элементов типа **type**

Распаковка данных в буфер

- **int MPI_Unpack (void *buf, int bufsize, int *bufpos, void *data, int count, MPI_Datatype type, MPI_Comm comm)**

  buf - буфер памяти с упакованными данными,

  bufsize – размер буфера в байтах,

  bufpos – позиция начала данных в буфере (в байтах от начала буфера),

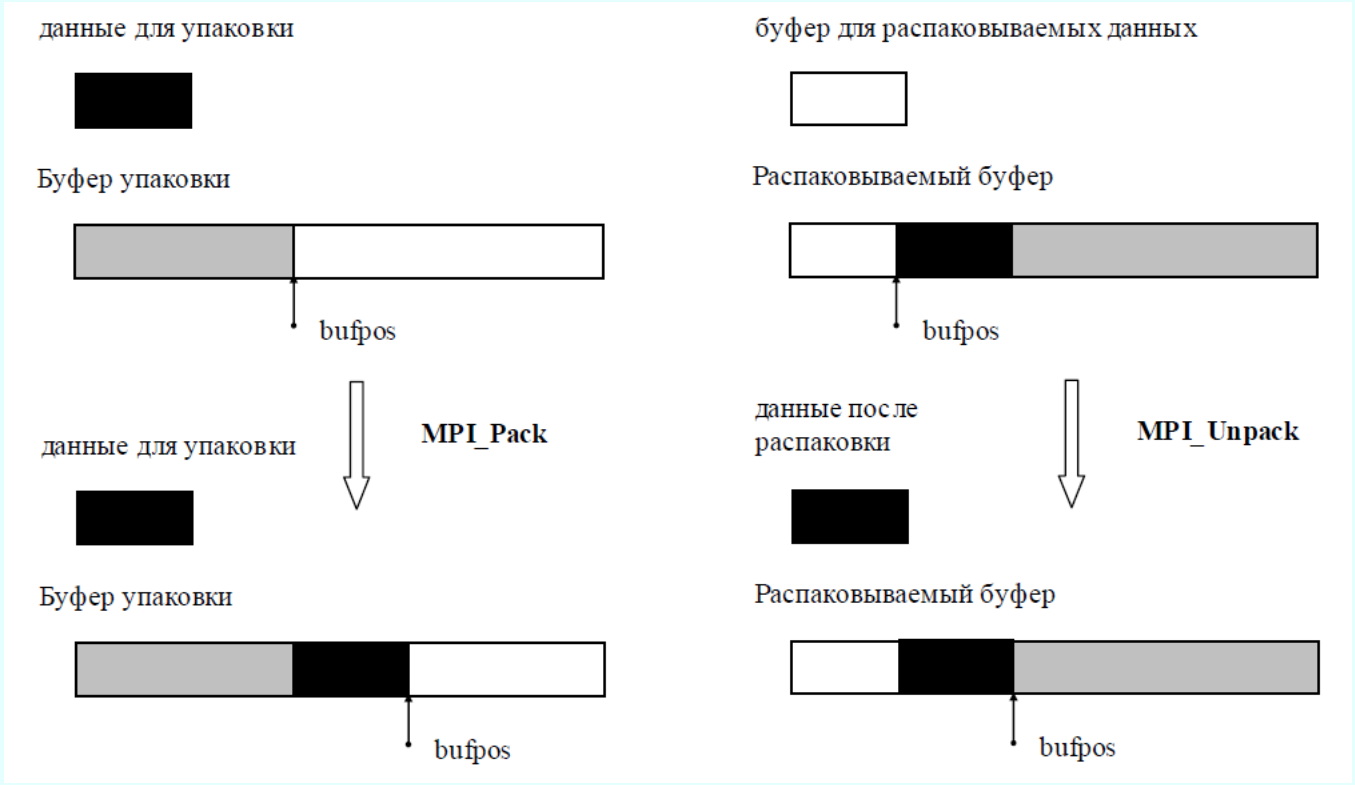
  data – буфер памяти для распаковываемых данных,

  count – количество элементов в буфере,

  type – тип распаковываемых данных,

  comm - коммуникатор для упакованного сообщения.

Упаковка и распаковка буфера



Производные типы данных

Способы конструирования производных типов:

- Непрерывный

  Позволяет определить непрерывный набор элементов существующего типа как новый производный тип

- Векторный

  Обеспечивает создание нового типа как набора элементов существующего типа, между элементами которого существуют регулярные промежутки памяти

- Индексный

  Отличается от векторного метода тем, что промежутки между элементами исходного типа могут иметь нерегулярный характер,

- Структурный

  Обеспечивает самое общее описание производного типа через явное указание карты создаваемого типа данных.

Непрерывный способ конструирования

- int **MPI_Type_contiguous**(int count, MPI_Data_type oldtype, MPI_Datatype *newtype)

  Новый тип newtype создается как count элементов исходного типа oldtype

```
int a[16];
MPI_Datatype intArr16;
MPI_Type_contiguous(16, MPI_INT, &intArr16);
MPI_Type_commit(&intArr16);
MPI_Send(a, 1, intArr16,…);
…
MPI_Type_free(&intArr16);
```

Векторный способ конструирования

- **int MPI_Type_vector (int count, int blocklen, int stride, MPI_Data_type oldtype, MPI_Datatype *newtype)**

  count – количество блоков

  blocklen – размер каждого блока,

  stride – количество элементов, расположенных между двумя соседними блоками

  oldtype - исходный тип данных,

  newtype - новый определяемый тип данных

Создание нового типа данных **newtype**, состоящего из **count** блоков по **blocklen** элементов базового типа данных **oldtype**. Следующий блок начинается через **stride** элементов типа **oldtype** после начала предыдущего. **Stride** может быть отрицательным

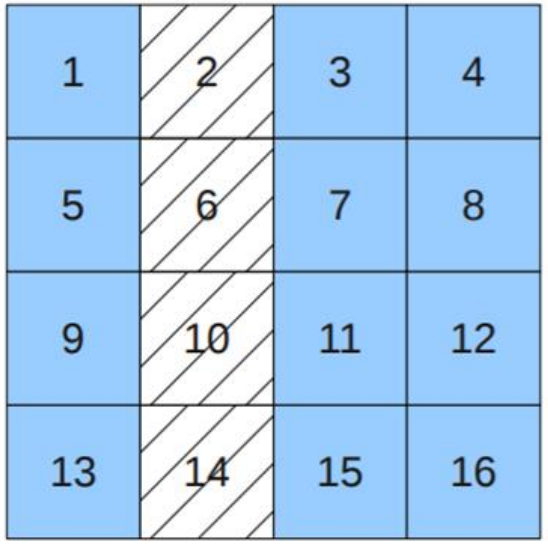
- **int MPI_Type_hvector (int count, int blocklen, MPI_Aint stride, MPI_Data_type oldtype, MPI_Datatype *newtype)**

Аналогично **MPI_Type_vector**, **stride** задается в байтах

Пример

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

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



Индексный способ конструирования

- **int MPI_Type_indexed (int count, int blocklens[], int indices[], MPI_Data_type oldtype, MPI_Datatype *newtype)**

  count – количество блоков,

  blocklens – количество элементов в каждов блоке,

  indices – смещение каждого блока от начала типа (в количестве элементов исходного типа),

  oldtype - исходный тип данных,

  newtype - новый определяемый тип данных.

Создание нового типа данных **newtype**, состоящего из **count** блоков по **blocklens** элементов базового типа данных **oldtype**. Следующий i-ый блок начинается через **indices[i]** элементов типа **oldtype** после начала предыдущего.

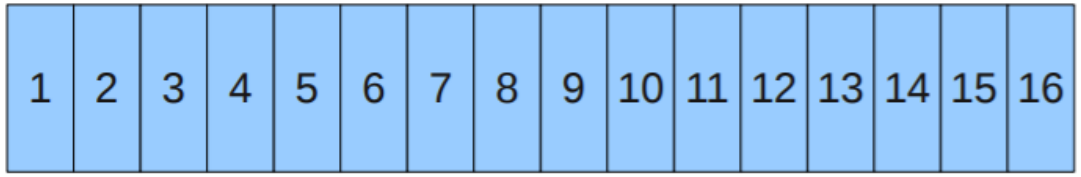
- **int MPI_Type_hindexed (int count, int blocklens[], MPI_Aint indices[], MPI_Data_type oldtype, MPI_Datatype *newtype)**

Аналогично **MPI_Type_indexed**, **indices[]** задается в байтах

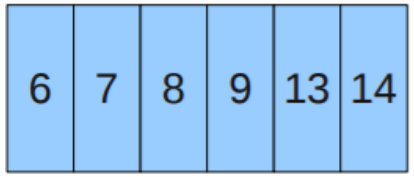
Пример

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];

Структурный способ конструирования

int MPI_Type_create_struct (int count, int blocklens[], MPI_Aint indices[], MPI_Data_type oldtypes[], MPI_Datatype *newtype)

count – количество блоков,

blocklens – количество элементов в каждов блоке,

indices – смещение каждого блока от начала типа (в байтах),

oldtypes - исходные типы данных в каждом блоке в отдельности,

newtype - новый определяемый тип данных.

Создание нового типа данных **newtype**, состоящего из **count** блоков по **blocklens** элементов типа данных **oldtype[]**. Следующий i-ый блок начинается через **indices[i]** байт после начала предыдущего.

Пример

- Пусть type1 есть {(double;0);(char;8)}

  B = (2, 1, 3)

  D = (0, 16, 26)

  T = (MPI_FLOAT, type1, MPI_CHAR)

- Вызов MPI_Type_create_struct (3,B,D, T, newtype) создаст следующий тип данных newtype:

{(float;0); (float; 4); (double;16); (char; 24) ; (char;26); (char; 27); (char; 28) }

Вспомогательные функции

- int **MPI_Type_size** (MPI_Datatype type, MPI_Aint *size)

  Получение размера типа в байтах (объема памяти, занимаемого одним элементом данного типа)

- int **MPI_Address** (void *location, MPI_Aint *address)

  Определения абсолютного байт-адреса размещения элементов в оперативной памяти

Объявление производных типов и их удаление

Перед использованием производный тип должен быть объявлен:

- int **MPI_Type_commit**(MPI_Datatype *type)

При завершении производный тип должен быть аннулирован:

- int **MPI_Type_free** (MPI_Datatype *type)

Задания

С помощью производных типов данных от одного процесса другому:

1. Передать диагональную матрицу
2. Передать нижнедиагональную матрицу
3. Передать четные столбцы