Лабораторная работа №4

Производные типы данных в MPI

Цель: изучить основные принципы использования производных типов данных в технологии MPI на примере использования в рамках языка C++.

Для получения **теоретических сведений** настоятельно рекомендуется при подготовке изучить материалы, представленные в списке литературы в конце разработки, а также прочие материалы по тематике лабораторной работы, представленные в открытых источниках.

Далее следует краткий конспект материала, приведенного в данных источниках, в конце включающий короткие примеры фрагментов программ.

1. Производные типы данных в MPI. Общие сведения.

Для обеспечения б**о**льших возможностей при определении состава передаваемых сообщений в MPI предусмотрен механизм **производных типов данных**.

В общем виде под **производным типом данных** в MPI можно понимать описание набора значений предусмотренного в MPI типа, причем в общем случае описываемые значения не обязательно непрерывно располагаются в памяти. Задание типа в MPI принято осуществлять при помощи **карты типа** (**type map**) в виде последовательности описаний входящих в тип значений, каждое отдельное значение описывается указанием типа и смещения адреса месторасположения от некоторого базового адреса, часть карты типа с указанием только типов значений именуется в MPI **сигнатурой типа**. Сигнатура типа описывает, какие базовые типы данных образуют некоторый производный тип данных MPI и, тем самым, управляет интерпретацией элементов данных при передаче или получении сообщений. Смещения карты типа определяют, где находятся значения данных.

Поясним рассмотренные понятия на следующем примере. Пусть в сообщение должны входить значения переменных:

```
double a; /* адрес 24 */
double b; /* адрес 40 */
int n; /* адрес 48 */
```

Тогда производный тип для описания таких данных должен иметь карту типа следующего вида:

```
{(MPI_DOUBLE,0), (MPI_DOUBLE,16), (MPI_INT,24)}
```

Дополнительно для производных типов данных в MPI используется следующий ряд новых понятий: **нижняя граница** типа, **верхняя граница** типа, **протяженность** типа.

Для получения значения протяженности и размера типа в MPI предусмотрены функции:

```
int MPI_Type_extent( MPI_Datatype type, MPI_Aint *extent );
int MPI_Type_size ( MPI_Datatype type, MPI_Aint *size );
```

Определение нижней и верхней границ типа может быть выполнено при помощи функций:

```
int MPI_Type_lb ( MPI_Datatype type, MPI_Aint *disp );
int MPI_Type_ub( MPI_Datatype type, MPI_Aint *disp );
```

Важной и необходимой при конструировании производных типов является функция получения адреса переменной:

```
int MPI_Address ( void *location, MPI_Aint *address );
```

2. Непрерывный способ конструирования позволяет определить непрерывный набор элементов существующего типа как новый производный тип.

При непрерывном способе конструирования производного типа данных в MPI используется функция:

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

Как следует из описания, новый тип **newtype** создается как **count** элементов исходного типа **oldtype**. Например, если исходный тип данных имеет карту типа

```
{ (MPI_INT,0),(MPI_DOUBLE,8) }
```

то вызов функции **MPI_Type_contiguous** с параметрами

```
MPI_Type_contiguous (2, oldtype,&newtype);
```

приведет к созданию типа данных с картой типа

```
{ (MPI_INT,0),(MPI_DOUBLE,8),(MPI_INT,16),(MPI_DOUBLE,24) }
```

В целом использование аргумента **count** в процедурах MPI равносильно использованию непрерывного типа данных такого же размера.

3. Векторный способ конструирования обеспечивает создание нового производного типа как набора элементов существующего типа, между элементами которого существуют регулярные промежутки по памяти, размер промежутков задается в числе элементов исходного типа, в то время, как в варианте H-векторного способа этот размер указывается в байтах.

При векторном способе конструирования производного типа данных в MPI используются функции:

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

где

count – количество блоков,
blocklen – размер каждого блока,
stride – количество элементов, расположенных между двумя соседними блоками
oldtype - исходный тип данных, newtype - новый определяемый тип данных.

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

Отличие способа конструирования, определяемого функцией **MPI_Type_hvector**, состоит лишь в том, что параметр **stride** для определения интервала между блоками задается в байтах, а не в элементах исходного типа данных.

При векторном способе новый производный тип создается как набор блоков из элементов исходного типа, при этом между блоками могут иметься регулярные промежутки по памяти.

В MPI есть возможность создания производных типов для описания подмассивов многомерных массивов при помощи функции (стандарт MPI-2):

```
int MPI_Type_create_subarray (int ndims, int *sizes, int *subsizes,
   int *starts, int order, MPI_Data_type oldtype, MPI_Datatype *newtype);
```

где

ndims – размерность массива;
sizes – количество элементов в каждой размерности исходного массива;
subsizes – количество элементов в каждой размерности определяемого подмассива;
starts – индексы начальных элементов в каждой размерности определяемого подмассива;
order - параметр для указания необходимости переупорядочения;
oldtype - тип данных элементов исходного массива;
newtype - новый тип данных для описания подмассива.

4. Индексный способ конструирования отличается от векторного метода тем, что промежутки между элементами исходного типа могут иметь нерегулярный характер

При индексном способе конструирования производного типа данных в MPI используются функции:

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

где

- 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.

При индексном способе новый производный тип создается как набор блоков разного размера из элементов исходного типа, при этом между блоками могут иметься разные промежутки по памяти. Данный способ конструирования отличается тем, что элементы **indices** для определения интервалов между блоками задаются в байтах, а не в элементах исходного типа данных.

Также существует функция **MPI_Type_create_indexed_block** индексного способа конструирования для определения типов с блоками одинакового размера (данная функция предусматривается стандартом MPI-2).

5. Структурный способ конструирования обеспечивает самое общее описание производного типа через явное указание карты создаваемого типа данных.

Как отмечалось ранее, структурный способ является самым общим методом конструирования производного типа данных при явном задании соответствующей карты типа. Использование такого способа производится при помощи функции:

```
int MPI_Type_struct (int count, int blocklens[], MPI_Aint indices[],
   MPI_Data_type oldtypes[], MPI_Datatype *newtype);
```

где

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.

Как следует из описания, структурный способ дополнительно к индексному методу позволяет указывать типы элементов для каждого блока в отдельности.

6. Объявление производных типов и их удаление

Функции конструирования позволяют определить производный тип данных. Дополнительно перед использованием созданный тип **должен быть объявлен** при помощи функции:

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

При завершении использования производный тип должен быть аннулирован при помощи функции:

```
int MPI_Type_free(MPI_Datatype *type)
```

7. Формирование сообщений при помощи упаковки и распаковки данных

В MPI предусмотрен и явный способ сборки и разборки сообщений, содержащих значения разных типов и располагаемых в разных областях памяти.

Для использования данного подхода должен быть определен буфер памяти достаточного размера для сборки сообщения. Входящие в состав сообщения данные должны быть **упакованы** в буфер при помощи функции:

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

где

data – буфер памяти с элементами для упаковки,
count – количество элементов в буфере,
type – тип данных для упаковываемых элементов,
buf - буфер памяти для упаковки,
buflen – размер буфера в байтах,
bufpos – позиция для начала записи в буфер (в байтах от начала буфера),
comm - коммуникатор для упакованного сообщения.

Функция MPI_Pack упаковывает **count** элементов из буфера **data** в буфер упаковки **buf**, начиная с позиции **bufpos**.

Начальное значение переменной **bufpos** должно быть сформировано до начала упаковки и далее устанавливается функцией **MPI_Pack**. Вызов функции **MPI_Pack** осуществляется последовательно для упаковки всех необходимых данных. Так, для ранее рассмотренного примера набора переменных **a**,**b** и **n**, для их упаковки необходимо выполнить:

```
bufpos = 0;
MPI_Pack(a,1,MPI_DOUBLE,buf,buflen,&bufpos,comm);
MPI_Pack(b,1,MPI_DOUBLE,buf,buflen,&bufpos,comm);
MPI_Pack(n,1,MPI_INT,buf,buflen,&bufpos,comm);
```

Для определения необходимого размера буфера для упаковки может быть использована функция:

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

которая в параметре **size** указывает необходимый размер буфера для упаковки **count** элементов типа **type**.

После упаковки всех необходимых данных подготовленный буфер может быть использован в функциях передачи данных с указанием типа **MPI_PACKED**.

После получения сообщения с типом **MPI_PACKED** данные могут быть распакованы при помощи функции:

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

где

buf - буфер памяти с упакованными данными,
buflen – 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.

Функция **MPI_Unpack** распаковывает начиная с позиции **bufpos** очередную порцию данных из буфера **buf** и помещает распакованные данные в буфер **data**. Начальное значение переменной **bufpos** должно быть сформировано до начала распаковки и далее устанавливается функцией

MPI_Unpack. Вызов функции **MPI_Unpack** осуществляется последовательно для распаковки всех упакованных данных, при этом порядок распаковки должен соответствовать порядку упаковки. Так, для ранее рассмотренного примера упаковки для распаковки упакованных данных необходимо выполнить:

```
bufpos = 0;
MPI_Pack(buf,buflen,&bufpos,a,1,MPI_DOUBLE,comm);
MPI_Pack(buf,buflen,&bufpos,b,1,MPI_DOUBLE,comm);
MPI_Pack(buf,buflen,&bufpos,n,1,MPI_INT,comm);
```

Данный способ может быть оправдан при сравнительно небольших размерах сообщений и при малом количестве повторений.

Задание. Реализуйте на основе технологии MPI многопоточную программу в соответствии с вариантом задания**.** Найдите возможные способы проверки корректности работы программы. Оцените их возможное достоинства и недостатки. Проверьте корректность работы программы при помощи одного из них. **Результаты занесите в отчет.**

Вариант	**Задание**
0	Реализуйте тип «комплексная матрица». Напишите программу, которая осуществляет умножение A матриц размером NxN.
1	Реализуйте тип «длинное целое». Напишите программу, которая осуществляет умножение A целых чисел заданной длины.
2	Реализуйте тип «полином». Напишите программу, которая осуществляет умножение A полиномов заданной степени.

Требования к сдаче работы

1. При подготовке изучить теоретический материал по тематике лабораторной работы, представленный в списке литературы ниже, выполнить представленные примеры, занести в отчёт результаты выполнения.
2. Продемонстрировать программный код для лабораторного задания.
3. Продемонстрировать выполнение лабораторных заданий (можно в виде скриншотов).
4. Ответить на контрольные вопросы.
5. Показать преподавателю отчет.

Литература

1. Спецификации стандарта Open MPI (версия 1.6, на английском языке): http://www.open-mpi.org/doc/v1.6/
2. Материалы, представленные на сайте intuit.ru в рамках курса «Intel Parallel Programming Professional (Introduction)»: http://old.intuit.ru/department/supercomputing/ppinteltt/5/
3. С.А. Лупин, М.А. Посыпкин Технологии параллельного программирования. – М.: ИД «ФОРУМ»: ИНФРА-М, 2011. – С. 12-96. *(Глава, посвященная MPI)*
4. Отладка приложений MPI в кластере HPC http://msdn.microsoft.com/ru-ru/library/dd560808(v=vs.100).aspx
5. http://www.parallel.ru/tech/tech_dev/mpi.html
6. http://msdn.microsoft.com/ru-ru/library/ee441265(v=vs.100).aspx#BKMK_debugMany