Санкт-Петербургский государственный университет
кафедра вычислительной физики

С.А.Немнюгин

Лабораторная работа 0.4
Производные типы в MPI

Методические материалы к курсу «Средства программирования для многопроцессорных вычислительных систем»

Intel Multicore Curriculum Initiative

Санкт-Петербург
2007

Производные типы

Производные типы данных создаются во время выполнения программы. Создание типа — двухступенчатый процесс, который состоит из двух шагов:

1. конструирование типа;
2. регистрация типа.

После завершения работы с производным типом, он аннулируется. При этом все производные от него типы остаются и могут использоваться дальше, пока и они не будут уничтожены. Последовательность удаления может быть любой.

Производные типы данных создаются из базовых типов с помощью подпрограмм–конструкторов. Операции создания производных типов могут применяться рекурсивно.

Производный тип данных в MPI характеризуется последовательностью базовых типов и набором целочисленных значений смещения. Смещения отсчитываются относительно начала буфера обмена и определяют те элементы данных, которые будут участвовать в обмене. Смещения могут принимать как положительные, так и отрицательные значения. Не требуется также, чтобы они были упорядочены (по возрастанию или по убыванию). Порядок элементов в производном типе может отличаться от исходного. Один элемент данных может появляться в новом типе многократно. Элементы могут и располагаться с разрывами и перекрываться между собой. Последовательность пар (тип, смещение) называется *картой типа*.

Подпрограмма `MPI_Type_struct` (см. Методическое пособие) является наиболее общим конструктором типа в MPI - программист может использовать полное описание каждого элемента типа. Если пересылаемые данные содержат подмножество элементов массива, такая детальная информация не нужна, поскольку у всех элементов один и тот же базовый тип. MPI содержит три конструктора, которые можно использовать в такой ситуации: `MPI_Type_contiguous`, `MPI_Type_vector` и `MPI_Type_indexed`. Первый из них создает производный тип, элементы которого являются непрерывно расположенными элементами массива. Второй создает тип, элементы которого расположены на одинаковых расстояниях друг от друга, а третий создает тип, содержащий произвольные элементы.

"Векторный" тип создается конструктором `MPI_Type_vector`. Схема расположения данных в новом типе представлена на рис. 1.

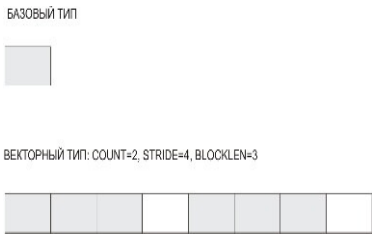

Рис. 1. Схема векторного типа данных

Лабораторная работа

В заданиях лабораторной работы 0.4 предлагается дописать предлагаемые фрагменты программ на языке С, написанные с использованием процедур MPICH 1.2.7. Пропущенные фрагменты обозначены многоточием.
Необходимый для выполнения данной лабораторной работы справочный материал можно найти на стр. 55 – 60 *методического пособия* «Средства программирования для многопроцессорных вычислительных систем».

Задание 1

В программе на языке Fortran имеется трехмерный массив `arr`. Используя приведенный ниже шаблон, дополните программу таким образом, чтобы в ней определялся производный тип, соответствующий: а) горизонтальному сечению массива; б) вертикальному сечению массива. Затем сечение должно пересылаться от процесса с рангом 0 процессу с рангом 1. Откомпилировать и запустить программу.

```
program main_mpi
include 'mpif.h'
parameter (n = 50)
real arr(n, n, n), b(...)
integer slice, sizeofreal
integer rank, ierr, status(MPI_STATUS_SIZE)
integer tag, cnt, vcount, blocklen, stride, count
cnt = …
tag = 0
vcount = …
blocklen = …
call MPI_Init(ierr)
call MPI_Comm_rank(MPI_COMM_WORLD, rank, ierr)
if (rank.eq.0) then
call MPI_Type_extent(MPI_REAL, sizeofreal, ierr)
stride = …
call MPI_Type_vector(…)
call MPI_Type_commit(slice, ierr)
call MPI_Send(arr(…), cnt, slice, 1, tag, MPI_COMM_WORLD, ierr)
else if (rank.eq.1) then
count = …
call  MPI_Recv(b,  count,  MPI_REAL,  0,  tag,  MPI_COMM_WORLD,
status, ierr)
print *, b
end if
call MPI_Finalize(ierr)
end
```

Задание 2

В программе на языке С задаются типы членов производного типа, затем количество элементов каждого типа. После этого вычисляются адреса членов типа `indata` и определяются смещения трех членов производного типа относительно адреса первого, для которого смещение равно 0. Затем определяется производный тип. Аргументы подпрограмм `MPI_Type_struct` и `MPI_Type_commit`, а также некоторые другие фрагменты пропущены. Добавить эти фрагменты, откомпилировать и запустить программу.

```
#include "mpi.h"
#include <stdio.h>
 struct newtype {
 float a;
 float b;
 int n;
 };

int main(int argc,char *argv[])
{
 int myrank;
 MPI_Datatype NEW_MESSAGE_TYPE;
 int block_lengths[3];
 MPI_Aint displacements[3];
 MPI_Aint addresses[4];
 MPI_Datatype typelist[3];
 int blocks_number;
 struct newtype indata;
 int tag = 0;
 MPI_Status status;

…
 MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

 typelist[0] = MPI_FLOAT;
 typelist[1] = MPI_FLOAT;
 typelist[2] = MPI_INT;
 block_lengths[0] = block_lengths[1] = block_lengths[2] = 1;
 MPI_Address(&indata, &addresses[0]);
 MPI_Address(&(indata.a), &addresses[1]);
 MPI_Address(&(indata.b), &addresses[2]);
 MPI_Address(&(indata.n), &addresses[3]);
 displacements[0] = addresses[1] – addresses[0];
 displacements[1] = addresses[2] – addresses[0];
 displacements[2] = addresses[3] – addresses[0];
 blocks_number = 3;
 MPI_Type_struct(…);
 MPI_Type_commit(…);

 if (myrank == 0)
{
```

```
 indata.a = 3.14159;
 indata.b = 2.71828;
 indata.n = 2002;

 MPI_Send(&indata, 1,NEW_MESSAGE_TYPE, 1, tag, MPI_COMM_WORLD);
 printf("Process %i send: %f %f %i\n", myrank, indata.a,
indata.b, indata.n);
}
else
{
 MPI_Recv(&indata, 1, NEW_MESSAGE_TYPE, 0, tag, MPI_COMM_WORLD,
&status);
 printf("Process %i received: %f %f %i, status %s\n", myrank,
indata.a, indata.b, indata.n, status.MPI_ERROR);
}

…

 MPI_Finalize();
 return 0;
}
```