Лабораторная работа №6

Виртуальные топологии

Цель: изучить основные принципы использования виртуальных топологий 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. Виртуальные топологии. Общие сведения

Под **топологией** вычислительной системы обычно понимается структура узлов сети и линий связи между этими узлами. Топология может быть представлена в виде графа, в котором вершины есть процессоры (процессы) системы, а дуги соответствуют имеющимся линиям (каналам) связи.

В MPI поддерживаются два вида топологий - **прямоугольная решетка** произвольной размерности (**декартова топология**) и **топология графа** любого произвольного вида. Следует отметить, что имеющиеся в MPI функции обеспечивают лишь получение новых логических систем адресации процессов, соответствующих формируемым виртуальным топологиям. Выполнение же всех коммуникационных операций должно осуществляться, как и ранее, при помощи обычных функций передачи данных с использованием исходных рангов процессов.

2. Декартовы топологии (решетки)

Декартовы топологии, в которых множество процессов представляется в виде прямоугольной **решетки**, а для указания процессов используется декартова система координат, широко применяются во многих задачах для описания структуры имеющихся информационных зависимостей.

Для создания декартовой топологии (решетки) в MPI предназначена функция:

```
int MPI_Cart_create(MPI_Comm oldcomm, int ndims, int *dims, int *periods,
   int reorder, MPI_Comm *cartcomm)
```

где:

oldcomm - исходный коммуникатор,

ndims - размерность декартовой решетки,

dims - массив длины ndims, задает количество процессов в каждом измерении решетки,

periods - массив длины ndims, определяет, является ли решетка периодической вдоль каждого измерения,

reorder - параметр допустимости изменения нумерации процессов,

cartcomm – создаваемый коммуникатор с декартовой топологией процессов.

Операция создания топологии является коллективной и, тем самым, должна выполняться всеми процессами исходного коммуникатора.

Рассмотрим пример создания двухмерной решетки **4x4**, в которой строки и столбцы имеют кольцевую структуру (за последним процессом следует первый процесс):

```
// создание двухмерной решетки 4x4
MPI_Comm GridComm;
int dims[2], periods[2], reorder = 1;
dims[0] = dims[1] = 4;
periods[0] = periods[1] = 1;
MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods, reoreder, &GridComm);
```

Для определения декартовых координат процесса по его рангу можно воспользоваться функцией:

```
int MPI_Card_coords(MPI_Comm comm, int rank, int ndims, int *coords)
```

где

comm – коммуникатор с топологией решетки,
rank - ранг процесса, для которого определяются декартовы координаты,
ndims - размерность решетки,
coords - возвращаемые функцией декартовы координаты процесса.

Обратное действие – определение ранга процесса по его декартовым координатам – обеспечивается при помощи функции:

```
int MPI_Cart_rank(MPI_Comm comm, int *coords, int *rank)
```

где

comm – коммуникатор с топологией решетки,
coords - декартовы координаты процесса,
rank - возвращаемый функцией ранг процесса.

Процедура разбиения решетки на подрешетки меньшей размерности обеспечивается при помощи функции:

```
int MPI_Card_sub(MPI_Comm comm, int *subdims, MPI_Comm *newcomm)
```

где

comm - исходный коммуникатор с топологией решетки,
subdims – массив для указания, какие измерения должны остаться в создаваемой подрешетке,
newcomm - создаваемый коммуникатор с подрешеткой.

Для пояснения функции **MPI_Cart_sub** дополним ранее рассмотренный пример создания двухмерной решетки и определим коммуникаторы с декартовой топологией для каждой строки и столбца решетки в отдельности:

```
// создание коммуникаторов для каждой строки и столбца решетки
MPI_Comm RowComm, ColComm;
int subdims[2];
// создание коммуникаторов для строк
subdims[0] = 0; // фиксации измерения
subdims[1] = 1; // наличие данного измерения в подрешетке
MPI_Cart_sub(GridComm, subdims, &RowComm);
// создание коммуникаторов для столбцов
subdims[0] = 1;
subdims[1] = 0;
MPI_Cart_sub(GridComm, subdims, &ColComm);
```

В приведенном примере для решетки размером 4x4 создаются 8 коммуникаторов, по одному для каждой строки и столбца решетки. Для каждого процесса определяемые коммуникаторы **RowComm** и **ColComm** соответствуют строке и столбцу процессов, к которым данный процесс принадлежит.

Дополнительная функция **MPI_Cart_shift** обеспечивает поддержку процедуры последовательной передачи данных по одному из измерений решетки (**операция сдвига данных** - см. раздел 3). В зависимости от периодичности измерения решетки, по которому выполняется сдвиг, различаются два типа данной операции:

- Циклический сдвиг на k элементов вдоль измерения решетки – в этой операции данные от процесса **i** пересылаются процессу **(i+k) mod dim**, где **dim** есть размер измерения, вдоль которого производится сдвиг,

- **Линейный сдвиг** на k позиций вдоль измерения решетки – в этом варианте операции данные от процессора **i** пересылаются процессору **i+k** (если таковой существует).

Функция **MPI_Cart_shift** обеспечивает получение рангов процессов, с которыми текущий процесс (процесс, вызвавший функцию **MPI_Cart_shift**) должен выполнить обмен данными:

```
int MPI_Card_shift(MPI_Comm comm, int dir, int disp, int *source, int *dst);
```

где

comm – коммуникатор с топологией решетки,
dir - номер измерения, по которому выполняется сдвиг,
disp - величина сдвига (<0 – сдвиг к началу измерения),
source – ранг процесса, от которого должны быть получены данные,
dst - ранг процесса которому должны быть отправлены данные.

Следует отметить, что функция **MPI_Cart_shift** только определяет ранги процессов, между которыми должен быть выполнен обмен данными в ходе операции сдвига. Непосредственная передачами данных, может быть выполнена, например, при помощи функции MPI_Sendrecv.

3. Топологии графа

Для создания коммуникатора с топологией типа граф в MPI предназначена функция:

```
int MPI_Graph_create(MPI_Comm oldcomm, int nnodes, int *index, int *edges,
   int reorder, MPI_Comm *graphcomm)
```

где

oldcomm - исходный коммуникатор,
nnodes - количество вершин графа,
index - количество исходящих дуг для каждой вершины,
edges - последовательный список дуг графа,
reorder - параметр допустимости изменения нумерации процессов,
cartcomm – создаваемый коммуникатор с топологией типа граф.

Количество соседних процессов, в которых от проверяемого процесса есть выходящие дуги, может быть получено при помощи функции:

```
int MPI_Graph_neighbors_count(MPI_Comm comm,int rank, int *nneighbors);
```

Получение рангов соседних вершин обеспечивается функцией:

```
int MPI_Graph_neighbors(MPI_Comm comm,int rank,int mneighbors, int *neighbors);
```

где **mneighbors** есть размер массива **neighbors**.

Лабораторные задания

Задание. Реализуйте на основе технологии MPI многопоточную программу умножения длинных целых чисел методом Шёнхаге — Штрассена, используя виртуальные топологии. Сравните результаты работы с программами умножения длинных целых, реализованными в Л.Р. №№ 4 и 5. **Результаты занесите в отчет.**

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