ЛАБОРАТОРНАЯ РАБОТА №3

ИСЛЕДОВАНИЕ ВОЗМОЖНОСТЕЙ ФОРМИРОВАНИЯ ВИРТУАЛЬНЫХ ТОПОЛОГИЙ ВЫЧИСЛИТЕЛЬНЫХ КЛАСТЕРОВ

ЦЕЛЬ РАБОТЫ: исследовать возможности, предоставляемые MPI по формированию виртуальных топологий.

1. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Топология является дополнительным необязательным атрибутом, который может соответствовать интра-коммуникатору; топологии не могут быть добавлены к интер-коммуникатору. Виртуальная топология вычислительного кластера – это удобный механизм именования для процессов группы внутри коммуникатора. Как было сказано ранее, группа в MPI является коллекцией *n* процессов. Каждый процесс в группе имеет ранг от **`0`** до ***`n-1`***. Во многих параллельных приложениях линейное ранжирование процессов недостаточно отображает логическую коммуникационную модель взаимодействия процессов (которая обычно определяется проблемой геометрии топологии и используемым алгоритмом нумерации). Часто процессы организуются в топологические модели, такие как двух- или трех-мерные сетки. В общем виде логическая организация процессов описывается графом. Такая логическая организация процессов называется «виртуальной топологией». Существует четкое различие между виртуально топологией процессов и топологией физической аппаратуры. Виртуальная топология может быть использована системой для распределения процессов на физических процессорах, если это помогает улучшить коммуникационную производительность на данной машине. Описание виртуальной топологии, с другой стороны, зависит только от приложения и является машинно-независимой.

1.1. Виртуальные топологии

Коммуникационная модель взаимодействия процессов может быть представлена в виде графа. Узлы представляют собой процессы, ребра соединяют процессы, которые взаимодействуют друг с другом. MPI обеспечивает передачу сообщений между любой парой процессов в группе. После создания связей между процессами (в виртуальной топологии) отсутствуют специальные условия для открытия канала, поэтому недостающее ребро в определённом пользователем графе не запрещает соответствующую передачу между процессами. Данное состояние говорит только о том, что связь является игнорируемой в виртуальной топологии. Такая стратегия предполагает, что топология определяет способ именования путей коммуникации. Указание виртуальной топологии в виде графа является достаточным для всех приложений. Большая часть всех параллельных приложений использует топологию процессов, таких как кольцо, сетку с двумя или более измерениями, или торы. Эти структуры являются совершенно различными по количеству измерений и количеству процессов в каждом координатном направлении. Координаты процесса в координатной структуре начинают свое исчисление с **`0`**. Измерение по строкам является всегда более важным в координатной структуре. Это значит что отношение между рангами групп и координатами для четырех процессов в сетке (2*2) является следующим:

координата (0, 0): ранг 0
координата (0, 1): ранг 1
координата (1, 0): ранг 2
координата (1, 1): ранг 3

1.2. Конструкторы топологий

Для создания декартовой топологии используется функция:

```
int MPI_Cart_create(MPI_Comm comm_old, int ndims, int *dims,
int *periods,int reorder, MPI_Comm *comm_cart);
```

Атрибутами этой функции являются:

in **comm_old** - входной коммуникатор; in **ndims** - количество измерений декартовой сетки; in **dims** - целочисленный массив размера **`ndims`**, указывающий количество процессов в каждом измерении; in **periods** - логический массив размера **`ndims`**, указывающий является ли сетка периодической (**`true`**) или нет (**`false`**) в каждом измерении; in **reorder** - ранжирование может быть переупорядочено (**`true`**) или нет (**`false`**); out **comm_cart** - коммуникатор с новой декартовой топологией.

Функция возвращает дескриптор нового коммуникатора, к которому прикреплена информация о декартовой топологии. Если аргумент **`reorder`** равен значению **`false`**, тогда ранг каждого процесса в новой группе является идентичным его рангу в старой группе. В противном случае функция может переупорядочить процессы. Если общий размер декартовой сетки является меньше, чем размер группы коммуникатора **`comm`**, тогда некоторые процессы возвращаются как **`MPI_COMM_NULL`**, по аналогии с функцией **`MPI_Comm_split`**. Если аргумент **`ndims`** равен нулю, тогда создается декартовая нуль-мерная топология. Вызов является ошибочным, если он определяет сетку больше чем размер группы, или если аргумент **`ndims`** является отрицательным. Для декартовых топологий функция **`MPI_Dims_create`** помогает пользователю выбрать сбалансированное распределение процессов по каждому координатному направлению в зависимости от количества процессов в группе и от дополнительных ограничений, которые могут быть указаны

пользователем. Одним из использований является разбитие всех процессов на **n**-мерную топологию. Вид вызова функции следующий:
int MPI_Dims_create(int nnodes, int ndims, int *dims)
Атрибутами этой функции являются:
in **nnodes** - количество узлов в сетке; in **ndims** - количество декартовых измерений; inout **dims** - целочисленный массив размера **ndims** указывающий количество узлов в каждом измерении;

Элементы в массиве **dims** определяются для описания декартовой сетки с **ndims** измерениями и суммой **nnodes** узлов. Измерения устанавливаются так, чтобы быть как можно ближе друг к другу, используя подходящий алгоритм делимости. Если элемент **dims[i]**является положительным числом, функция не будет модифицировать количество узлов в измерении **i**; только те элементы, где значение **dims[i] = 0**, будут модифицированы вызывающей стороной.

Отрицательные входные значения **dims[i]** являются ошибочными. Ошибка будет происходить, если аргумент **nnodes** не является произведением:

$$\prod_{i,dims[i]\neq 0} dims[i].$$

Функция **MPI_Dims_create** является локальной.

Таблица 1.1

Пример использования функции **MPI_Dims_create**

Dims до вызова	вызов функции	**Dims** на выходе
(0,0)	**MPI_Dims_create(6,2,dims)**	(3, 2)
(0, 0)	**MPI_Dims_create(7,2,dims)**	(7, 1)
(0, 3, 0)	**MPI_Dims_Create(6,3,dims)**	(2, 3, 1)
(0, 3, 0)	**MPI_Dims_create(7,3,dims)**	Ошиб. вызов

Для создания топологии графа используется следующая функция:
int MPI_Graph_create(MPI_Comm comm_old, int nnodes, int *index, int *edges, int reorder, MPI_Comm *comm_graph);
Атрибутами этой функции являются:
in **comm_old** - входной коммуникатор; in **nnodes** - количество узлов в графе; in **index** - целочисленный массив, описывающий степени узлов; in **edges** - целочисленный массив, описывающий ребра графа; in **reorder** - ранжирование может быть переупорядочено (**true**) или нет (**false**); out **comm_graph** - коммуникатор с добавленной топологией графа.

Функция возвращает дескриптор нового коммуникатора, к которому прикреплена информация о топологии графа. Если аргумент **reorder** равен значению **false**, тогда ранг каждого процесса в новой группе является идентичным его рангу в старой группе. Если размер (аргумент **nnodes)** графа меньше чем размер группы коммуникатора **comm**, тогда некоторые процессы будут возвращены как **MPI_COMM_NULL**, по аналогии с **MPI_Cart_create**. Если граф пустой, т.е. аргумент **nnodes = 0**, тогда значение **MPI_COMM_NULL** возвращается во всех процессах. Вызов является ошибочным, если он указывает граф, который является больше, чем размер группы входного коммуникатора. Три параметра **nnodes**, **index** и **edges** определяют структуру графа. Аргумент **nnodes** является количеством узлов графа. Узлы нумеруются от **0** до **nnodes-1**. При этом **i**-ый элемент массива **index** сохраняет общее число соседей первых **i** узлов графа. Список соседей узлов **0**, **1**, …, **nnodes-1** сохраняется в последовательности массива **edges**. Массив **edges** является развернутым представление списка ребер. Общее количество элементов в аргументе **index** является равным **nnodes,**общее количество элементов в аргументе **edges** является равным количеству ребер графа.

Пример: Предположим, что существует четыре процесса 0, 1, 2, 3 со следующей матрицей смежности:

процессы	соседи
0	1, 3
1	0
2	3
3	0, 2

Тогда, входные аргументы следующие:
nnodes = 4
index = 2, 3, 4, 6
edges = 1, 3, 0, 3, 0, 2

Поэтому в С элемент **index[0]** является степенью нулевого узла, а **index[i] – index[i-1]** является степенью узла **i**, где **i=1**, …, **nnodes-1**; список соседей нулевого узла сохраняется в элементе **edges[j]**, для **0**≤ **j** ≤ **index[0]-1** исписок соседей узлов **i (**где**i**>**0)**, сохраняются в элементе **edges[j]**, для**index[i-1]** ≤ **j**≤ **index[i]-1**.

1.3. Реализация топологических запросов

Если топология была определена одной из предыдущих функции, тогда информация о топологии может быть просмотрена, используя функции запросов, которые являются локальными вызовами:

intMPI_Topo_test(MPI_Commcomm, int *status) **Атрибутами этой функции являются:**
in **comm** – коммуникатор; out **status** - тип топологии коммуникатора **comm;**

Функция возвращает тип топологии, которая назначена коммуникатору.
Выходное значение аргумента **status** является одним из следующих:

MPI_GRAPH	топология графа
MPI_CART	декартова топология
MPI_DIST_GRAPH	распределенная топология графа
MPI_UNDEFINED	топологии нет

Функции **MPI_Graphdims_get** и **MPI_Graph_get** получают информацию о топологии графа, которая была ассоциирована с коммуникатором через **MPI_Cart_create**:
int MPI_Graphdims_get(MPI_Comm comm, int *nnodes, int *nedges);
Атрибутами этой функции являются:
in **comm** - коммуникатор с топологией графа; out **nnodes** - количество узлов в графе; out **nedges** -количество ребер в графе;

Функции **MPI_Cartdim_get** и **MPI_Cart_get** возвращают информацию о декартовой топологии, которая была ассоциирована с коммуникатором, используя функцию **MPI_Cart_create**:
int MPI_Cartdim_get(MPI_Comm comm, int *ndims);
Атрибутами этой функции являются:
in **comm** - коммуникатор с декартовой структурой; out **ndims** - количество измерений декартовой структуры.

Если аргумент **comm** ассоциирован с нуль-мерной декартовой топологией, функция **MPI_Cartdim_get** возвратит аргумент **ndims=0**, функция **MPI_Cart_get** при этом оставит все выходные аргументы неизмененными:
int MPI_Cart_get(MPI_Comm comm, int maxdims, int *dims, int *periods,int *coords). **Атрибутами этой функции являются:**
in **comm** - коммуникатор с декартовой структурой; in **maxdims** - длина вектора **dims**, **period** и **cords** в вызывающей программе; out **dims** - количество процессов для каждого декартового измерения; out **periods** - периодичность для каждого декартового измерения; out **cords** - координаты вызывающего процесса в декартовой структуре;

Для того, чтобы на основе координат получить ранг процесса, используется следующая функция:
int MPI_Cart_rank(MPI_Comm comm, int *coords, int *rank)
Атрибутами этой функции являются:
in **comm** - коммуникатор с декартовой структурой; in **cords** - целочисленный массив (размера **ndims**) определяющий декартовы координаты процесса; out **rank** - ранг указанного процесса;

Для группы процессов с декартовой структурой функция **MPI_Cart_rank** переведет логические координаты процесса в ранг процесса, как они использовались бы процедурами точка-точка.
Для обратного отображения, перевод ранга в координаты, используется функция, синтаксис которой следующий:
Int MPI_Cart_coords (MPI_Comm comm, int rank, int maxdims, int *coords);
Атрибутами этой функции являются:
in **comm** - коммуникатор с декартовой структурой; in **rank** - ранг процесса внутри группы **comm; i**n **maxdims** - длина вектора **cords** в вызывающей программе; out **cords** - целочисленный массив (размера **ndims**) содержащий декартовы координаты указанного процесса.

Если аргумент **comm** ассоциирован снуль мерной декартовой топологией, аргумент **cords** будет неизменен.

Функции **MPI_Graph_neighbors_count** и **MPI_Graph_neighbors** обеспечивают информацию о смежности для общей топологии графа:
int MPI_Graph_neighbors_count(MPI_Comm comm, int rank, int *nneighbors);
Атрибутами этой функции являются:
in **comm** - коммуникатор с топологией графа; in **rank** - ранг процесса в группе **comm;** out **nneighbors** - количество соседей указанного процесса.
int MPI_Graph_neighbors(MPI_Comm comm, int rank, int maxneighbors, int *neighbors);
Атрибутами этой функции являются:
in comm - коммуникатор с топологией графа; in rank - ранг процесса в группе **comm;** in maxneighbors -размер массива соседей; out neighbors - ранги процессов, которые являются соседями указанного процесса;

Возвращаемое количество и массив соседей для запрашиваемого ранга будет как включать всех соседей, так и отражать такой же порядок ребер, как было определено оригинальным вызовом функции **MPI_Graph_create**. Точнее данные функции будут возвращать значения, соответствующие аргументам **index** и **edges**, переданным в функцию **MPI_Graph_create.** Количество соседей возвращенное функцией **MPI_Graph_neighbors_count** будет равно **index[rank]–index[rank–1]**. Массив **neighbors**, возвращенный функцией **MPI_Graph_neighbors** будет от **edges[index[rank – 1]]** до **edges[index[rank – 1]]**.

1. Задание на работу

Задание выбирается в соответствии с вариантом, назначенным преподавателем. Алгоритмы перемножения матриц, которые необходимо реализовать в вариантах, находятся в приложении В.

2.1. Вариант №1

Необходимо реализовать алгоритм перемножения матриц ленточным способом с распределением столбцов.

2.2. Вариант №2

Необходимо реализовать алгоритм перемножения матриц ленточным способом с распределением строк.

2.3. Вариант №3

Необходимо реализовать алгоритм перемножения матриц по методу Фокса.

3. Контрольные вопросы

3.1.Обязана ли виртуальная топология повторять физическую топологию целевого компьютера?
3.2.Любой ли коммуникатор может обладать виртуальной топологией?
3.3.Может ли процесс входить одновременно в декартову топологию и в топологию графа?
3.4.Как определить, с какими процессами в топологии графа связан данный процесс?