Средства и системы параллельного программирования

#9. Группы и виртуальные топологии MPI

Самостоятельная работа

Реализовать коллективную операцию, используя двуточечные операции MPI

1 вариант:

```
MY_MPI_Bcast (void *buffer, int count,
MPI_Datatype datatype, int root,
MPI_Comm comm)
```

2 вариант:

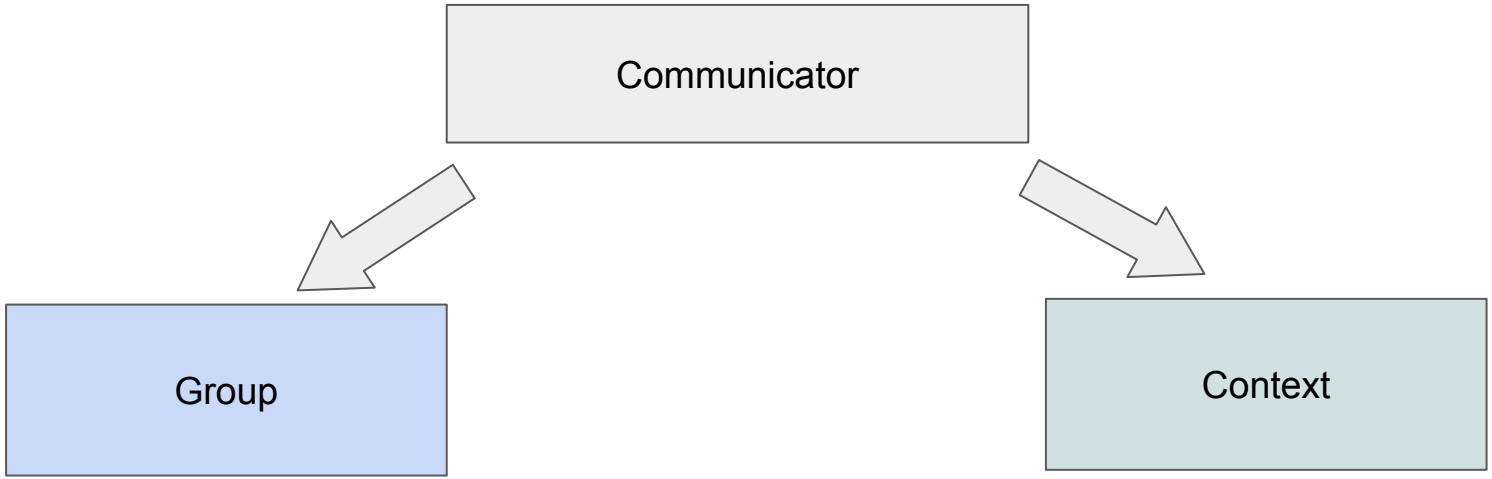
```
MY_MPI_Reduce (int* buffer,
int* result_buffer, int n,
int root, MPI_Comm comm)
```

Итоговое решение должно работать быстрее, чем за O(N) шагов, где N - число процессов, выполняющих операцию, а шаг - посылка одним процессом сообщения другому процессу. Передачи сообщений от разных процессов могут быть выполнены независимо и параллельно.

int MPI_Send(const void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)

int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status * status)

MPI Communicator



Predefined MPI communicators

MPI_COMM_WORLD

MPI_COMM_SELF

MPI_COMM_NULL

MPI Group

```
int MPI_Comm_group(MPI_Comm comm, MPI_Group *group);
```

Вернуть через group группу процессов, ассоциированную с коммуникатором comm

```
int MPI_Group_incl(MPI_Group group, int n, int *ranks,
MPI_Group *newgroup);
```

Создать новую группу newgroup путём выбора процессов из старой группы group. Нужно указать число процессов n и ранки ranks в старой нумерации (в нумерации group)

Создание коммуникатора через определение группы

```
int MPI_Comm_create( MPI_Comm comm, MPI_Group group,
MPI_Comm *newcomm );
```

Создание нового коммуникатора newcomm, ассоциированного с группой group, на основе старого comm.

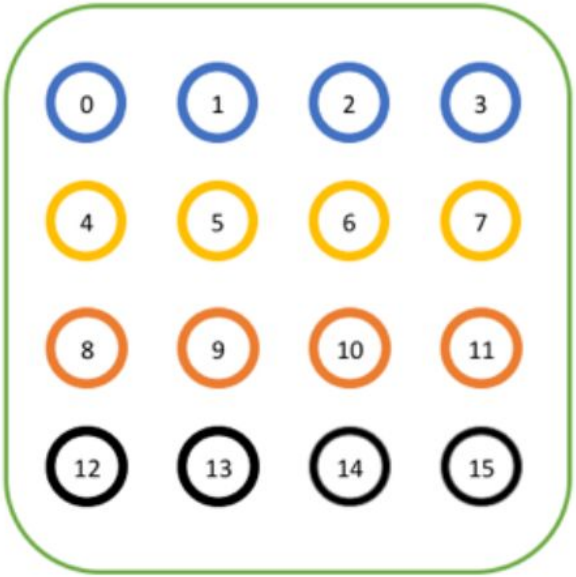
Создание коммуникатора разделением существующего

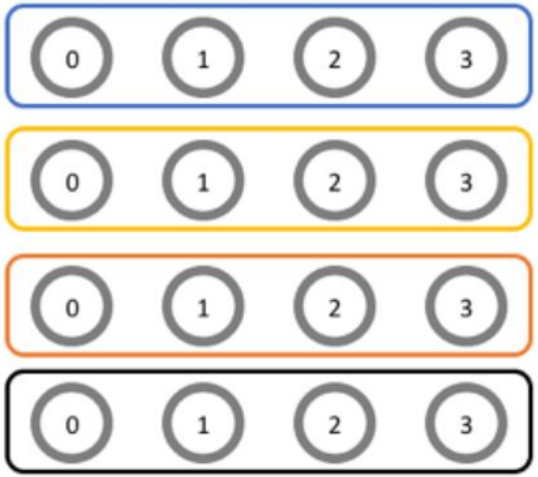
```
int MPI_Comm_split(MPI_Comm comm, int color, int key, MPI_Comm
*newcomm);
```

Создание нового коммуникатора newcomm на основе существующего коммуникатора comm.

Группа, ассоциированная с коммуниатором comm, разделяется на N групп, где N - число уникальных значений color среди всех процессов, вызвавших функцию. Внутри каждой группы производится своя нумерация, причём порядковый номер rank внутри группы выставляется в зависимости от key (у какого процесса больше значение key, тот в новой группе получит бОльший ранк).

У каждой группы newcomm свой и отличный от других групп.


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

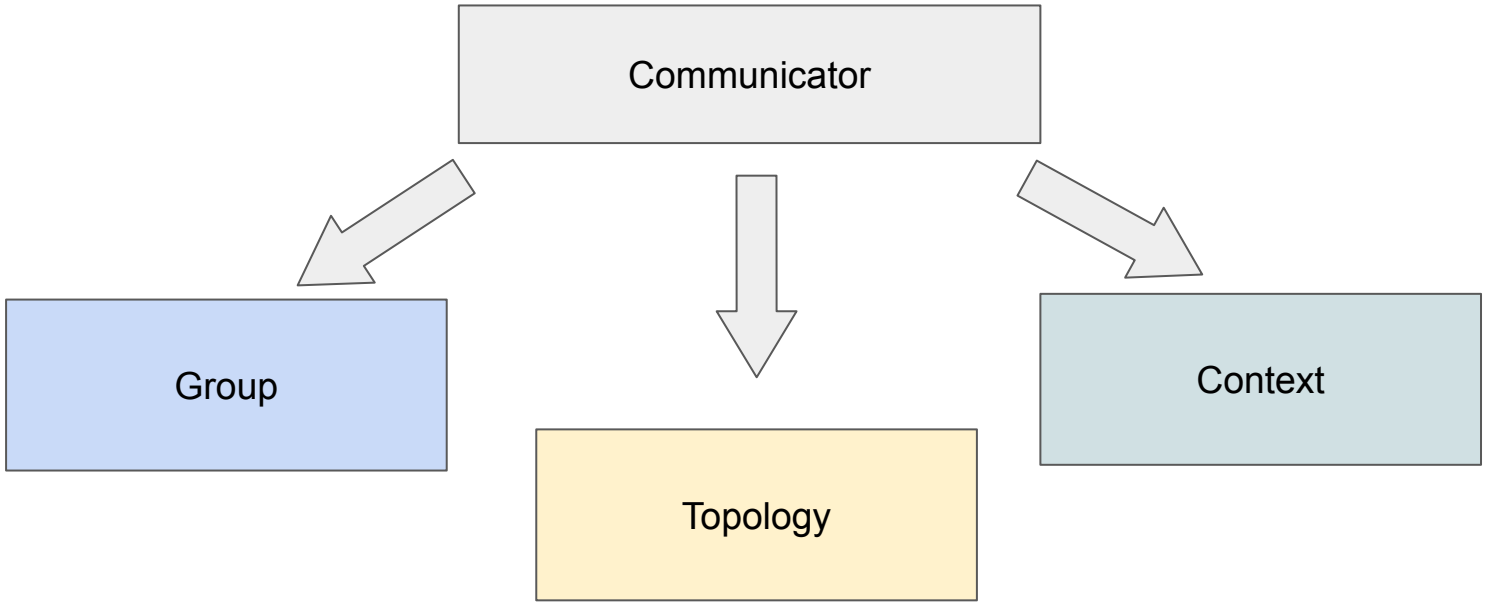

0
1
2
3
0
1
2
3
0
1
2
3
0
1
2
3

MPI Comm dup

```
int MPI_Comm_dup(MPI_Comm old_comm, MPI_Comm* new_comm);
```

Дублирование коммуникатора (сохраняется и контекст, и соответствующая коммуникатору группа)

MPI Virtual topology



MPI Virtual topology

Топологии предоставляют дополнительный механизм перенумерации процессов коммуникатора с целью более удобной реализации алгоритмов

Топологии в MPI виртуальные, не имеют ничего общего с топологиями сетей вычислительной машины/кластера

Топологии в MPI могут быть графовыми и декартовыми

Создание декартовой топологии

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

Создание декартовой топологии из процессов коммуникатора comm_old

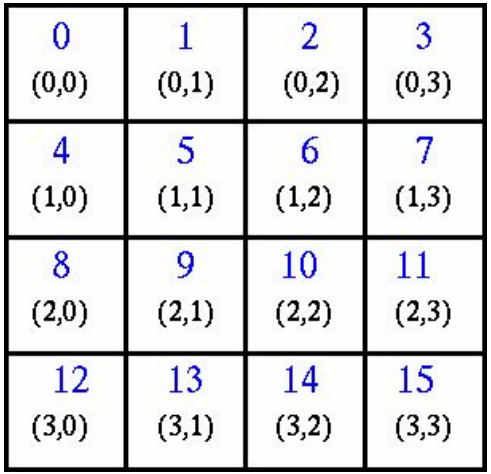
ndims - число измерений декартовой решётки

*dims - число процессов по каждому измерению [ndims]

*periods - периодичность каждого измерения [ndims]

reorder - позволить MPI перенумеровать процессы

comm_cart - новый коммуникатор, через который далее нужно обмениваться сообщениями



Определение места процесса в новой топологии

int MPI_Cart_coords(MPI_Comm comm, int rank, int dimension_number, int* coords);

Получить координаты coords[] процесса rank в декартовой решётке

int MPI_Cart_rank(MPI_Comm comm, const int coords[], int *rank)

По заданным координатам coords[] процесса получить его ранк rank

Определение размера по каждому измерению

int MPI_Dims_create(int nnodes, int ndims, int dims[])

Определение “оптимального” разбиения всех nnodes процессов в ndims-мерную решётку, получение “оптимальных” размеров dims[]

Определение соседей процесса

```
int MPI_Cart_shift(MPI_Comm comm, int direction, int displ,
int *source, int *dest);
```

Виртуально сдвинуть топологию по измерению direction на displ элементов-процессов (знак определяет направление сдвига)

В dest будет находиться ранк процесса, в который мы придём из нынешнего ранка, выполнив сдвиг

В source будет находиться ранк процесса, из которого в процессе сдвига мы придём к нынешнему процессу

Производные топологии на основе декартовой

```
int MPI_Cart_sub(MPI_Comm comm, const int remain_dims[],
MPI_Comm *newcomm)
```

Аналог функции MPI_Comm_split, только принимает в качестве входного коммуникатора comm, определённый на декартовой решётке.

Новые коммуникаторы newcomm создаются исходя из логики, что те измерения, которые должны “остаться” в новом коммуникаторе, определены в remain_dims[] и имеют ненулевое вхождение

```
/* Create 2D Cartesian topology for processes */
MPI_Cart_create(MPI_COMM_WORLD, ndim, dims, period, reorder, &comm2D);
MPI_Comm_rank(comm2D, &id2D);
MPI_Cart_coords(comm2D, id2D, ndim, coords2D);
/* Create 1D row subgrids */
belongs[0] = 0;
belongs[1] = 1; // this dimension belongs to subgrid
MPI_Cart_sub(comm2D, belongs, &commrow);
/* Create 1D column subgrids */
belongs[0] = 1; // this dimension belongs to subgrid
belongs[1] = 0;
MPI_Cart_sub(comm2D, belongs, &commcol);
```

0,0 (0)	**0, 1 (1)**
1, 0 (2)	**1, 1 (3)**
2, 0 (4)	**2, 1 (5)**

2D Cartesian Grid

0, 0 (0) **0 (0)**	**0, 1 (1)** **1 (1)**
1, 0 (2) 0 (0)	1, 1, (3) 1 (1)
2, 0 (4) 0 (0)	2, 1 (5) 1 (1)

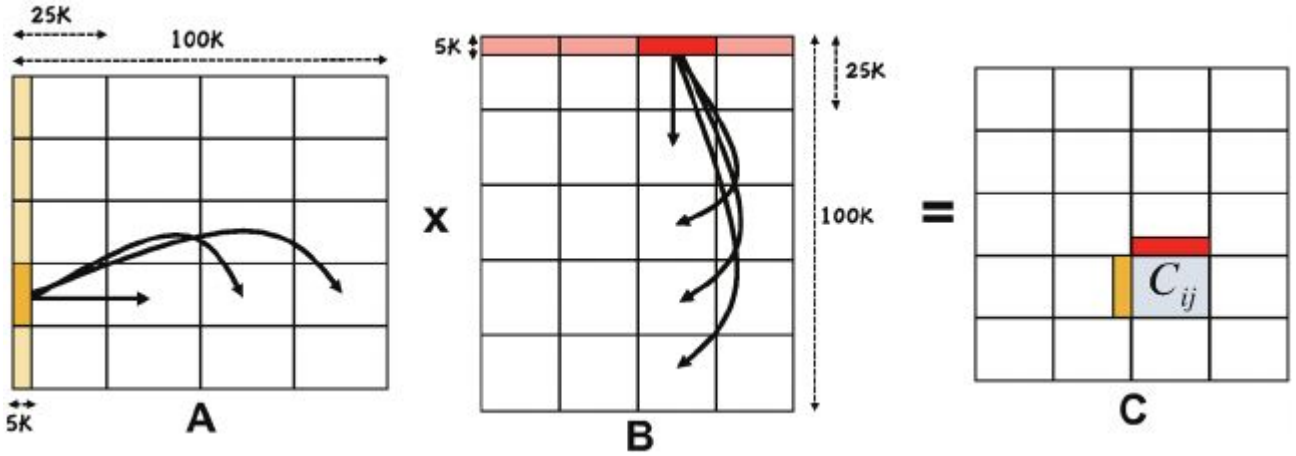
3 row subgrids

0, 0 (0) **0 (0)**	**0, 1 (1)** **0 (0)**
1, 0 (2) 1 (1)	1, 1 (3) 1 (1)
2, 0 (4) 2 (2)	2, 1 (5) 2 (2)

2 column subgrids

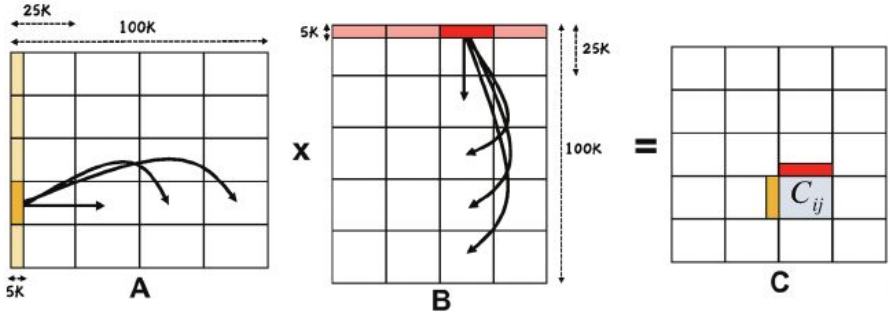
Задание

Алгоритм SUMMA (Scalable Universal Matrix Multiplication Algorithm)



Van De Geijn, Robert A., and Jerrell Watts. "SUMMA: Scalable universal matrix multiplication algorithm." *Concurrency: Practice and Experience* 9, no. 4 (1997): 255-274.

Задание




```
for k=0 to n-1 … или n/b-1 где b – размер блока
    … = # cols in A(i,k) and # rows in B(k,j)
    for all i = 1 to pr … in parallel
        owner of A(i,k) broadcasts it to whole processor row
    for all j = 1 to pc … in parallel
        owner of B(k,j) broadcasts it to whole processor column
    Receive A(i,k) into Acol
    Receive B(k,j) into Brow
    C_myproc = C_myproc + Acol * Brow
```

Задание

Требования:

1. Реализовать алгоритм SUMMA, используя процессные топологии
2. Матрицы предполагаем квадратными, N*N, тип данных - INT или FLOAT
3. Каждый процесс хранит в памяти только свою часть матриц A, B, C (+ приходящие данные)
4. Можете выбрать один из двух типов допущений:
    a.
        i. P - полный квадрат, т.е процессная топология - квадрат
        ii. Протестировать при P = 1, 4, 9, 16
        iii. Программа должна работать при произвольном размере блока b (b < N/sqrt(P)), N/sqrt(P) делится на b
    b.
        i. Программа должна работать при произвольной процессной топологии
        ii. Протестировать при P = 1–16
        iii. b = 1
5. На небольших размерах матрицы убедиться, что алгоритм выдаёт верный результат (сравнимый с последовательным ijk)
6. Провести запуски на Polus (mpisubmit.pl), нарисовать T(P), составить небольшой отчёт

Дедлайн: 9.12, 16.12