FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION
OF HIGHER EDUCATION
ITMO UNIVERSITY

Parallel algorithms for the analysis and synthesis of data on the assignments No.15, 16, 17

Performed by
Ivan Dubinin
J4132c

St. Petersburg
2021

Assignment 15

Understand the new functions in Assignment15.c.

Append part of code.

Listing of the program

See it in my github repo

Description

Program created n processes. All processes are distributed to 2 groups based on process rank (less or more/equal than n / 2). Then new communicator created for new group. Function MPI_Allreduce returns sum of old ranks for every new group. Then new rank is written in variable new_rank. In the end, cout call is added to output old rank, new rank and result of MPI_Allreduce work for every process.

Example of launch parameters and output

```
[pes@vandosik HW_MPI]$ mpic++ Assignment15.c -o task_15
[pes@vandosik HW_MPI]$ mpirun -n 8 --use-hwthread-cpus task_15 --mca opal_warn_on_missing_libcuda 0
rank = 0  newrank = 0  rbuf = 6
rank = 1  newrank = 1  rbuf = 6
rank = 2  newrank = 2  rbuf = 6
rank = 3  newrank = 3  rbuf = 6
rank = 4  newrank = 0  rbuf = 22
rank = 6  newrank = 2  rbuf = 22
rank = 5  newrank = 1  rbuf = 22
rank = 7  newrank = 3  rbuf = 22
[pes@vandosik HW_MPI]$
```

Assignment 16

In the MPI_Comm_split function (Assignment16.c), replace the color parameter with (rank% 2), (rank% 3), look at how many groups the processes are split into, depending on the specified attribute of division into groups.

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

• comm - parent communicator

• color - a sign of division into groups

• key - parameter defining numbering in new groups

• OUT newcomm - new communicator

The function splits the group associated with the parent communicator into non-overlapping subgroups, one for each value of the color subgroup attribute. Color must be non-negative. Each subgroup contains processes with the same color value. The key parameter controls the ordering within the new groups: a lower key value corresponds to a lower process ID value. If the key parameter is equal for multiple processes, the ordering is performed according to the order in the parent group.

Listing of the program

See it in my github repo

Description

Program created n processes. Firstly all processes were added to new group with inverse ranks. Then all processes are distributed to groups based on color argument in MPI_Comm_splitfunction: using (rank % 2) as color, all processes were splitted into 2 groups (group of processes with initially odd ranks and group of processes with initially even rank). Similarly,in (rank % 3) case all processes were splitted into 3 groups. Rank1, rank2, rank3 variables are initialized with process rank in the new group and the initial rank is stored in proc_rank variable. For synchronization purposes MPI_Barrier and sleep() calls were used.

Example of launch parameters and output

```
[pes@vandosik HW_MPI]$ mpic++ Assignment16.c -o task_16
[pes@vandosik HW_MPI]$ mpirun -n 8 --use-hwthread-cpus task_16 --mca opal_warn_on_missing_libcuda 0
Setting color to (1)
proc_rank = 5, rank1 = 2
proc_rank = 7, rank1 = 0
proc_rank = 0, rank1 = 7
proc_rank = 3, rank1 = 4
proc_rank = 1, rank1 = 6
proc_rank = 2, rank1 = 5
proc_rank = 4, rank1 = 3
proc_rank = 6, rank1 = 1
Setting color to (proc_rank % 2)
proc_rank = 2, rank2 = 2
proc_rank = 7, rank2 = 0
proc_rank = 4, rank2 = 1
proc_rank = 5, rank2 = 1
proc_rank = 0, rank2 = 3
proc_rank = 3, rank2 = 2
proc_rank = 1, rank2 = 3
proc_rank = 6, rank2 = 0
Setting color to (proc_rank % 3)
proc_rank = 1, rank3 = 2
proc_rank = 5, rank3 = 0
proc_rank = 2, rank3 = 1
proc_rank = 7, rank3 = 0
proc_rank = 4, rank3 = 1
proc_rank = 0, rank3 = 2
proc_rank = 6, rank3 = 0
proc_rank = 3, rank3 = 1
[pes@vandosik HW_MPI]$
```

Assignment 17

Understand the new functions in Assignment17.c. and explain program execution.

Display the values of the process number and arrays a[i], b[i], before packing and distribution, and after. See how broadcasting works.

Listing of the program

See it in my github repo

Description

Program creates n processes. Each process creates two arrays of integers and chars. Both arrays are packed into one continuous memory block using MPI_Pack. Then that pack transferred from root process (with rank 0) to other processes using MPI_Bcast (if this function called on root process, data from buffer is broad casted to all processes, otherwise message is received into buffer). Using MPI_Unpack function continuous block of memory is transformed in arrays of integers and chars. For non-root processes received messages overwrites original arrays.

Example of launch parameters and output

```
[pes@vandosik HW_MPI]$ mpirun -n 8 --use-hwthread-cpus task_17 --mca opal_warn_on_missing_libcuda 0
Before distribution
Process: 3: a: [4,4,4,4,4,4,4,4,4,4]; b: [b,b,b,b,b,b,b,b,b,b]
Process: 4: a: [5,5,5,5,5,5,5,5,5,5]; b: [b,b,b,b,b,b,b,b,b,b]
Process: 5: a: [6,6,6,6,6,6,6,6,6,6]; b: [b,b,b,b,b,b,b,b,b,b]
Process: 6: a: [7,7,7,7,7,7,7,7,7,7]; b: [b,b,b,b,b,b,b,b,b,b]
Process: 7: a: [8,8,8,8,8,8,8,8,8,8]; b: [b,b,b,b,b,b,b,b,b,b]
Process: 0: a: [1,1,1,1,1,1,1,1,1,1]; b: [a,a,a,a,a,a,a,a,a,a]
Process: 1: a: [2,2,2,2,2,2,2,2,2,2]; b: [b,b,b,b,b,b,b,b,b,b]
Process: 2: a: [3,3,3,3,3,3,3,3,3,3]; b: [b,b,b,b,b,b,b,b,b,b]
After distribution
Process: 0: a: [1,1,1,1,1,1,1,1,1,1]; b: [a,a,a,a,a,a,a,a,a,a]
Process: 1: a: [1,1,1,1,1,1,1,1,1,1]; b: [a,a,a,a,a,a,a,a,a,a]
Process: 7: a: [1,1,1,1,1,1,1,1,1,1]; b: [a,a,a,a,a,a,a,a,a,a]
Process: 3: a: [1,1,1,1,1,1,1,1,1,1]; b: [a,a,a,a,a,a,a,a,a,a]
Process: 4: a: [1,1,1,1,1,1,1,1,1,1]; b: [a,a,a,a,a,a,a,a,a,a]
Process: 5: a: [1,1,1,1,1,1,1,1,1,1]; b: [a,a,a,a,a,a,a,a,a,a]
Process: 6: a: [1,1,1,1,1,1,1,1,1,1]; b: [a,a,a,a,a,a,a,a,a,a]
Process: 2: a: [1,1,1,1,1,1,1,1,1,1]; b: [a,a,a,a,a,a,a,a,a,a]
[pes@vandosik HW_MPI]$
```