**Лабораторная работа №1**

**Передача и прием сообщений в MPI**

**Макеева Настя ПИН-31**

**Цель:** изучить основные принципы приема и передачи сообщений в технологии MPI на примере использования в рамках языка С++.

**Контрольные вопросы**

1. **В чем состоят основы технологии MPI?**

В вычислительных системах с распределенной памятью процессоры работают независимо друг от друга. Для организации параллельных вычислений необходимо уметь::

*-распределять* вычислительную нагрузку,

*-организовать* информационное взаимодействие (*передачу данных*) между процессорами.

**«Одна программа – много процессов». В рамках MPI для решения задачи разрабатывается одна программа, она запускается на выполнение одновременно на всех имеющихся процессорах**

**Такой способ организации параллельных вычислений обычно именуется как *модель "одна программа - множество процессов"* (*single program multiple processes или SPMP*) .** Процессы могут выполняться как на разных процессорах, так и на одном. Каждыйпроцесс параллельной программы порождается на основе копии одного и того жепрограммного кода (модель SPMP). Все процессы программы последовательноперенумерованы от 0 до p-1, где p есть общее количество процессов. Номер процессаименуется рангом процесса. Основу MPI составляют операции передачи сообщений. Среди предусмотренных в составе MPI функций различаются парные (point-to-point) операции между двумя процессами и коллективные (collective) коммуникационные действия для одновременного взаимодействия нескольких процессов.Процессы параллельной программы объединяются в группы. Под коммуникатором вMPI понимается служебный объект, объединяющий в своем составе группу процессови ряд дополнительных параметров (контекст), используемых при выполненииопераций передачи данных. Один и тот же процесс может принадлежать разнымгруппам и коммуникаторам. Все имеющиеся в программе процессы входят в составсоздаваемого по умолчанию коммуникатора с идентификатором MPI\_COMM\_WORLD.При необходимости передачи данных между процессами из разных групп необходимосоздавать глобальный коммуникатор (intercommunicator).

1. **В чем состоят основные преимущества и недостатки технологии MPI?**

Преимущества:

-MPI позволяет существенно снизить остроту проблемы переносимости параллельных программ между разными компьютерными системами.

-MPI содействует повышению эффективности параллельных вычислений - практически для каждого типа вычислительных систем существуют реализации библиотек MPI.

-MPI уменьшает сложность разработки параллельных программ:

-- большая часть основных операций передачи данных предусматривается стандартом MPI,

-- имеется большое количество библиотек параллельных методов, созданных с использованием MPI.

Недостатки:

-низкий уровень (программирование на MPI часто сравнивают с

программированием на ассемблере),

-необходимость детального управления распределением массивов и витков циклов между процессами, а также обменом сообщениями между процессами - высокая трудоемкость разработки программ;

-необходимость избыточной спецификации типов данных в передаваемых

сообщениях, а так же наличие жестких ограничений на типы

передаваемых данных;

-сложность написания программ, способных выполняться при произвольных размерах массивов и произвольном количестве процессов

1. **Что понимается под параллельной программой в рамках технологии MPI?**

Под **параллельной программой** в рамках MPI понимается множество одновременно выполняемых **процессов**. Процессы могут выполняться на разных процессорах, но на одном процессоре могут располагаться и несколько процессов (в этом случае их исполнение осуществляется в режиме разделения времени). В предельном случае для выполнения параллельной программы может использоваться один процессор – как правило, такой способ применяется для начальной проверки правильности параллельной программы.

1. **Как происходит инициализация и завершение MPI программ?**

**Первой вызываемой функцией** MPI должна быть функция:

int MPI\_Init ( int \*agrc, char \*\*\*argv );

для инициализации среды выполнения MPI-программы. Параметрами функции являются количество аргументов в командной строке и текст самой командной строки.

**Последней вызываемой функцией** MPI обязательно должна являться функция:

int MPI\_Finalize (void);

1. **Как происходит передача и прием сообщений MPI программе?**

Для **передачи сообщения** процесс-отправитель должен выполнить функцию:

int MPI\_Send(void \*buf, int count, MPI\_Datatype type, int dest,

int tag, MPI\_Comm comm),

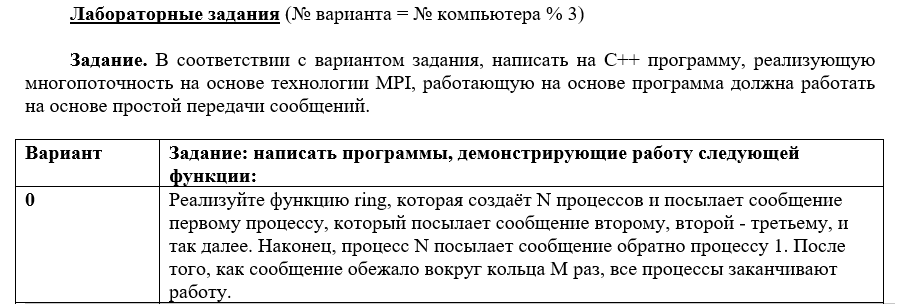
где buf – адрес буфера памяти, в котором располагаются данные отправляемого сообщения, count – количество элементов данных в сообщении, type - тип элементов данных пересылаемого сообщения, dest - ранг процесса, которому отправляется сообщение, tag - значение-тег, используемое для идентификации сообщений, comm - коммуникатор, в рамках которого выполняется передача данных.

Для **приема сообщения** процесс-получатель должен выполнить функцию:

int MPI\_Recv(void \*buf, int count, MPI\_Datatype type, int source,

int tag, MPI\_Comm comm, MPI\_Status \*status),

где buf, count, type – буфер памяти для приема сообщения, назначение каждого отдельного параметра соответствует описанию в MPI\_Send, - source - ранг процесса, от которого должен быть выполнен прием сообщения, tag - тег сообщения, которое должно быть принято для процесса, comm - коммуникатор, в рамках которого выполняется передача данных, status – указатель на структуру данных с информацией о результате выполнения операции приема данных.



#include <stdio.h>

#include "mpi.h"

#include "stdlib.h"

int main(int argc, char\* argv[])

{

int ProcNum, ProcRank, RingNum;

RingNum = atoi(argv[1]);

int recvFrom;

int sendTo;

char sendStr[30];

MPI\_Init(&argc, &argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &ProcNum);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &ProcRank);

if ( ProcRank == 0 )

{

char message[] = "message";

MPI\_Send( message, countof( message ), MPI\_CHAR, 1, 0, MPI\_COMM\_WORLD );

printf("process rank %d send message to 1 \n", ProcRank);

}

else

{

for ( int i = 0; i < RingNum; i++ )

{

char msg[30];

if ( i > 0 && ProcRank == 1 ) { recvFrom = ProcNum – 1 ; }

else { recvFrom = ProcRank - 1; }

MPI\_Recv( msg, countof( msg ), MPI\_CHAR, recvFrom, 0, MPI\_COMM\_WORLD,

MPI\_STATUSES\_IGNORE);

printf("processor number %d received message : \"%s\" from processor number %d for the %d time \n\n", ProcRank, msg, recvFrom, I );

if ( ProcRank == ProcNum - 1) sendTo = 1;

else sendTo = ProcRank + 1;

for ( int i = 0; i < 7; i++)

{

sendStr[i] = msg[i+1];

}

//отправляем сообщения

MPI\_Send(sendStr, countof( sendStr ), MPI\_CHAR, sendTo, 0, MPI\_COMM\_WORLD);

}

}

MPI\_Finalize();

return 0;

}

#include <string>

#include <vector>

#include <cstdlib>

#include <iostream>

#include <stdio.h>

#include "mpi.h"

#include "stdlib.h"

int main(int argc, char\* argv[])

{

int ProcNum, ProcRank, RingNum;

RingNum = atoi(argv[1]);

int recvFrom;

int sendTo;

char sendStr[30];

MPI\_Init(&argc, &argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &ProcNum);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &ProcRank);

for (int u = 0; u < 9; u++) {

if (ProcRank == 0)

{

char message[] = "message";

//MPI\_Send(message, \_countof(message), MPI\_CHAR, 1, 0, MPI\_COMM\_WORLD);

printf("process rank %d send message to 1 \n", ProcRank);

}

else

{

for (int i = 0; i < RingNum; i++)

{

char message[30];

if (i > 0 && ProcRank == 1) { recvFrom = ProcNum - 1; }

else { recvFrom = ProcRank - 1; }

MPI\_Recv(message, \_countof(message), MPI\_CHAR, recvFrom, 0, MPI\_COMM\_WORLD,

MPI\_STATUSES\_IGNORE);

printf("processor number %d received message : \"%s\" from processor number %d for the %d time \n\n", ProcRank, message, recvFrom, i);

if (ProcRank == ProcNum - 1) sendTo = 1;

else sendTo = ProcRank + 1;

for (int k = 0; k < 7; k++)

{

sendStr[k] = message[k + 1];

}

//отправляем сообщения

MPI\_Send(sendStr, \_countof(sendStr), MPI\_CHAR, sendTo, 0, MPI\_COMM\_WORLD);

RingNum--;

}

}

}

MPI\_Finalize();

return 0;

}