Системы и средства параллельного программирование

2016 – 2017 уч. г.

Лектор: доцент Н.Н.Попова,

Лекция 4

Тема: **«MPI. Базовые операции передачи данных»**

10 октября 2016 г.

Основные понятия MPI

- Процессы объединяются в ***группы***.
- Группе приписывается ряд свойств (как связаны друг с другом и некоторые другие). Получаем ***коммуникаторы***
- Процесс идентифицируется своим номером в группе, привязанной к конкретному коммуникатору.
- При запуске параллельной программы создается специальный коммуникатор с именем ***MPI_COMM_WORLD***
- **Все** обращения к MPI функциям содержат коммуникатор, как параметр.

Понятие коммуникатора MPI

- **Все** обращения к MPI функциям содержат коммуникатор, как параметр.
- Наиболее часто используемый коммуникатор `MPI_COMM_WORLD`
  - определяется при вызове `MPI_Init`
  - содержит ВСЕ процессы программы

Типы данных MPI

- Данные в сообщении описываются тройкой: (***address, count, datatype***)
- ***datatype*** (типы данных MPI)

Signed
- MPI_CHAR
- MPI_SHORT
- MPI_INT
- MPI_LONG

Unsigned
- MPI_UNSIGNED_CHAR
- MPI_UNSIGNED_SHORT
- MPI_UNSIGNED
- MPI_UNSIGNED_LONG

- MPI_FLOAT
- MPI_DOUBLE
- MPI_LONG_DOUBLE

Базовые MPI-типы данных (C)

MPI datatype	C datatype
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double

Специальные типы MPI

- MPI_Comm
- MPI_Status
- MPI_datatype

Понятие тэга

- Сообщение сопровождается определяемым пользователем признаком – целым числом – ***тэгом*** для идентификации принимаемого сообщения
- Теги сообщений у отправителя и получателя должны быть согласованы. Можно указать в качестве значения тэга константу **`MPI_ANY_TAG`**.
- Некоторые не-MPI системы передачи сообщений называют тэг типом сообщения. MPI вводит понятие тэга, чтобы не путать это понятие с типом данных MPI.

C: MPI helloworld.c

```
#include <stdio.h>
#include <mpi.h>
int main(int argc, char **argv){
    MPI_Init(&argc, &argv);
    printf("Hello, MPI world\n");
    MPI_Finalize();
return 0; }
```

Формат MPI-функций

C (case sensitive):

```
error = MPI_Xxxxx(parameter,...);
MPI_Xxxxx(parameter,...);
```

C++ (case sensitive):

```
error = MPI::Xxxxx(parameter,...);
MPI::Xxxxx(parameter,...);
```

Основные группы функций MPI

- Определение среды
- Передачи «точка-точка»
- Коллективные операции
- Производные типы данных
- Группы процессов
- Виртуальные топологии
- Односоторонние передачи данных
- Параллельный ввод-вывод
- Динамическое создание процессов
- Средства профилирования

Функции определения среды

```
int MPI_Init(int *argc, char ***argv)
```
должна первым вызовом, вызывается только один раз

```
int MPI_Comm_size(MPI_Comm comm, int *size)
```
число процессов в коммуникаторе

```
int MPI_Comm_rank(MPI_Comm comm, int *rank)
```
номер процесса в коммуникаторе (нумерация с 0)

```
int MPI_Finalize()
```
завершает работу процесса

```
int MPI_Abort (MPI_Comm_size(MPI_Comm comm,
int*errorcode)
```
завершает работу программы

Инициализация MPI

MPI_Init должна первым вызовом, вызывается только один раз

C:

```
int MPI_Init(int *argc, char ***argv)
```

http://www.mcs.anl.gov/research/projects/mpi/www/www3/MPI_Init.html

Обработка ошибок MPI-функций

Определяется константой ***MPI_SUCCESS***

```
I
int error;
……
error = MPI_Init(&argc, &argv));
If (error != MPI_SUCCESS)
{
 fprintf (stderr, “ MPI_Init error \n”);
return 1;

}
```

MPI_Comm_size
Количество процессов в коммуникаторе

- **Размер коммуникатора**

```
int MPI_Comm_size(MPI_Comm comm, int *size)
```

Результат – число процессов

http://www-unix.mcs.anl.gov/mpi/www/www3/MPI_Comm_size.html

MPI_Comm_rank
номер процесса (process rank)

- Process ID в коммуникаторе
  - Начинается с 0 до (*n*-1), где *n* – число процессов
- Используется для определения номера процесса-отправителя и получателя

```
int MPI_Comm_rank(MPI_Comm comm, int *rank)
```

Результат – номер процесса

Завершение MPI-процессов

- Никаких вызовов MPI функций после

C:

int MPI_Finalize()

int MPI_Abort (MPI_Comm_size(MPI_Comm comm, int*errorcode)

Если какой-либо из процессов не выполняет MPI_Finalize, программа зависает.

Hello, MPI world! (2)

```
#include <stdio.h>
#include "mpi.h"
int main(int argc, char **argv){
   int rank, size;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    printf("Hello, MPI world! I am %d of %d\n“,rank,size);
    MPI_Finalize();
    return 0; }
```

Пример : скалярное произведение векторов. Алгоритм.

Алгоритм:

- Считаем, что элементы векторов A и B равномерно распределены по процессам. Число элементов векторов кратно числу процессов.
- Начальные значения векторов определяются каждым процессом
- Схема алгоритма: главный-подчиненный (master-slave)
- Главный (master):

  - обработка входных данных (чтение количества элементов векторов из командной строки)

  - рассылка данных по подчиненным

  - прием результатов (локальных сумм) от подчиненных

  - вычисление общей суммы
- Завершение работы

Пример : скалярное произведение. Главный процесс.

- Главный (master):

  - обработка входных данных (чтение количества элементов векторов из командной строки)

  - рассылка данных по подчиненным

  for (i=1; i<Nproc; i++)

  MPI_Send( )

  - прием результатов (локальных сумм) от подчиненных

  for (i=1; i<Nproc; i++)

  MPI_Recv( )

  - вычисление общей суммы

- Завершение работы

  MPI_Finalize()

Пример : скалярное произведение. Подчиненные процессы.

Подчиненные процессы (slaves)

- Прием информации о числе элементов от Master

  MPI_Recv ( )
- Создание локальных массивов в динамической памяти
- Определение начальных значений элементов векторов
- Вычисление локальных сумм
- Пересылка вычисленных значений процессу Master

  MPI_Send ( )
- Завершение работы

  MPI_Finalize ()

Пример : скалярное произведение. Реализация

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
void  dataRead(double *a, double *b, int n);
double localSum(double *v,int n);
/*****************************************/
int main(int argc, char *argv[]){
  int nProc, myRank;
  int root =0;
  int tag =0;
  int i, N, N_local;
  double partialSum, totalSum=0.0;
  double *a_local, *b_local;
  double timeStart,timeFinish;

  MPI_Status status;
```

Пример : скалярное произведение. Реализация. Master.

```
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &myRank);
MPI_Comm_size(MPI_COMM_WORLD, &nProc);
if (myRank == root){
  timeStart = -MPI_Wtime();
  N = (int)(atof(argv[1]));
  N_local= N/nProc;
  for (i=1;i<nProc;i++)
    MPI_Send(&N_local, 1, MPI_INT, i, tag, MPI_COMM_WORLD);

   for (i=1;i<nProc;i++){
    MPI_Recv(&partialSum, 1, MPI_DOUBLE, i, MPI_ANY_SOURCE,
 tag,MPI_COMM_WORLD,&status);
 totalSum+=partialSum;
  }
  timeFinish = MPI_Wtime();
  printf(" Sum of vector =%e\n",totalSum);
  printf(" Program Run time = %d\n", timeStart+timeFinish);
}
```

Пример : скалярное произведение. Реализация

```
else {
    MPI_Recv(&N_local, 1, MPI_INT, root, tag, MPI_COMM_WORLD,
   &status);
    a_local=(double *) malloc(N_local*sizeof(double));
   b_local=(double *) malloc(N_local*sizeof(double));
    dataRead(a_local, N_local);
    dataRead(b_local, N_local);
    partialSum=localSum(a_local,b_local,N_local);

   MPI_Send(&partialSum,1,MPI_DOUBLE,root,tag,MPI_COMM_WORLD)
   ;
    free(a_local); free(b_local);
  }
  MPI_Finalize();
   return 0;
}
```

Пример : скалярное произведение. Реализация

```
/*****************************************************/
void dataRead(double *v, int n){
int i;
for (i=0; i<n; i++)
  v[i]= (double)i;
}  /*dataRead*/

/*****************************************************/
double localSum(double *a,double *b, int n){
int i;
double sum=0.0;
for (i=0; i<n; i++)
   sum+=a[i]*b[i];
return sum;
}/*localSum*/
```

Трансляция программы система Regatta (IBM pSeries 690)

%mpicc –o dot_pr dot_pr.c

Справочная информация по опциям компилятора

% mpicc – help

Запуск программы на счет

- Обычно:

%mpirun –np 4 dot_pr 40000

- Запуск на Regatta под управлением системы управления заданиями LoadLeveler

%mpisubmit –w 01:00 –n 8 dot_pr

- одна минута

- 8 процессов

Запуск программы на счет

mpisubmit [<параметры mpisubmit>] <имя задачи – название исполняемого файла> [<параметры задачи>]

Параметры mpisubmit:

-w *лимит счетного времени предполагаемое время счета задания в формате чч:мм:сс или сс, или мм:сс; (По умолчанию 10 минут)*

-n *число процессоров, максимально 16, по умолч. 1*

-m *почтовый адрес*

<username>@regatta.cmc.msu.ru

На данный адрес будет послана информация по завершению задачи

-stdout *файл для потока вывода*

Имя_задания.nnnn.out

в каталог, из которого происходила постановка задания в очередь.

-stderr *файл для потока ошибок*

имя_задания.nnnn.err

в каталог, из которого происходила постановка задания в очередь.

-stdin *файл для потока ввода*

БАЗОВАЯ ИНСТРУКЦИЯ ПО РАБОТЕ на ВС Regatta

- Выход на систему:
  локальная машина>ssh <логин>@regatta.cs.msu.su
- Копировать файл с локального компьютера на Regatta:
  локальная машина>scp file <логин>@regatta.cs.msu.su:~/file
- Компиляция MPI-программы (на языке C, C++ и Fortran90 соответственно):
  >mpicc prog.c -o prog
  >mpiCC prog.cpp -o prog
  >mpif90 prog.F -o prog
- Постановка программы в очередь задач с лимитом выполнения 15 минут на четыре процессора с параметром командной строки parameter:
  >mpisubmit -w 15:00 -n 4 prog parameter
- Просмотреть состояние очереди:
  >llq
- Удалить задачу с ID regatta.1111 из очереди задач:
  >llcancel regatta.1111
- Полное руководство по ссылке: http://regatta.cmc.msu.ru

Основа 2-точечных обменов

*int MPI_Send(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**)*

Информация о завершившемся приеме сообщения

- Возвращается функцией **`MPI_Recv`** через `параметр status`

- Содержит:
  - Source: ***`status.MPI_SOURCE`***
  - Tag: ***`status.MPI_TAG`***
  - Count: ***`MPI_Get_count`***

MPI_Probe

- Int MPI_Probe (**int** *source*, **int** *tag*, **MPI_Comm** *comm*, **MPI_Status** **status* **);**

Замер времени MPI_Wtime

- Время замеряется в секундах
- Выделяется интервал в программе

```
double MPI_Wtime(void);
```

Пример.

```
double start, finish, elapsed, time ;
start=-MPI_Wtime;
MPI_Send(…);
finish = MPI_Wtime();
time= start+finish;
```

Функции MPI передачи «точка-точка»

Point-to-Point Communication Routines		
MPI_Bsend	MPI_Bsend_init	MPI_Buffer_attach
MPI_Buffer_detach	MPI_Cancel	MPI_Get_count
MPI_Get_elements	MPI_Ibsend	MPI_Iprobe
MPI_Irecv	MPI_Irsend	MPI_Isend
MPI_Issend	MPI_Probe	MPI_Recv
MPI_Recv_init	MPI_Request_free	MPI_Rsend
MPI_Rsend_init	MPI_Send	MPI_Send_init
MPI_Sendrecv	MPI_Sendrecv_replace	MPI_Ssend
MPI_Ssend_init	MPI_Start	MPI_Startall
MPI_Test	MPI_Test_cancelled	MPI_Testall
MPI_Testany	MPI_Testsome	MPI_Wait
MPI_Waitall	MPI_Waitany	MPI_Waitsome