**Основы параллельного программирования**

**Лабораторная работа № 1**

**Простые примеры параллельных программ**

**Цель** – дать представление о построении простых параллельных программ на языке параллельного программирования MPI; представление о параллельных программах, настраиваемых на размер вычислительной системы, как на параметр; практическое освоение функций парных и коллективных взаимодействий между ветвями параллельной программы.

**Методы распараллеливания и модели программ, поддерживаемые MPI**

Важнейшей особенностью MPI является то, что пользователь при написании своих параллельных программ не должен учитывать архитектурные особенности конкретных мультикомпьютеров, поскольку MPI предоставляет пользователю виртуальный мультикомпьютер с распределенной памятью и с виртуальной сетью связи между виртуальными компьютерами. Пользователь заказывает количество компьютеров, необходимых для решения его задачи, и определяет топологию связей между этими компьютерами. MPI реализует этот заказ на конкретной физической системе. Ограничением является объем оперативной памяти физического мультикомпютера. Таким образом пользователь работает в виртуальной среде, что обеспечивает переносимость его параллельных программ. Система MPI представляет собой библиотеку средств параллельного программирования для языков С и Fortran 77.

Одной из целей, преследуемых при решении задач на вычислительных системах, в том числе и на параллельных, – является эффективность. Эффективность параллельной программы существенно зависит от соотношения времени вычислений ко времени коммуникаций между компьютерами (при обмене данными). И чем меньше в процентном отношении доля времени, затраченного на коммуникации, в общем времени вычислений, тем больше эффективность. Для параллельных систем с передачей сообщений оптимальное соотношение между вычислениями и коммуникациями обеспечивают методы крупнозернистого распараллеливания, когда параллельные алгоритмы строятся из крупных и редко взаимодействующих блоков [2–8]. Задачи линейной алгебры, задачи, решаемые сеточными методами и многие другие, достаточно эффективно *распараллеливаются крупнозернистыми методами*.

MPMD - модель вычислений. MPI - программа представляет собой совокупность автономных процессов, функционирующих под управлением своих собственных программ и взаимодействующих посредством стандартного набора библиотечных процедур для передачи и приема сообщений. Таким образом, в самом общем случае MPI - программа реализует MPMD - модель программирования (Multiple program - Multiple Data).

SPMD - модель вычислений. Все процессы исполняют в общем случае различные ветви одной и той же программы. Такой подход обусловлен тем обстоятельством, что задача может быть достаточно естественным образом разбита на подзадачи, решаемые по одному алгоритму. На практике чаще всего встречается именно эта модель программирования (Single program - Multiple Data) [1,12].

Последнюю модель иначе можно назвать моделью *распараллеливания по данным.* Кратко, суть этого способа заключается в следующем. Исходные данные задачи распределяются по процессам (ветвям параллельного алгоритма), а алгоритм является одним и тем же во всех процессах, но действия этого алгоритма распределяются в соответствии с имеющимися в этих процессах *данными*. Распределение действий алгоритма заключается, например, в присвоении разных значений переменным одних и тех же циклов в разных ветвях, либо в исполнении в разных ветвях разного количества витков одних и тех же циклов и т.п. Другими словами, процесс в каждой ветви следует различными путями выполнения на той же самой программе.

Везде далее вместо "параллельная программа" будем писать сокращенно: п-программа.

ПРИМЕР 1.1

Каждая ветвь п-программы выводит на экран свой идентификационный номер и размер заказанной параллельной системы, т.е. количество виртуальных компьютеров, в каждый из которых загружается ветвь п-программы.

#include<stdio.h>

#include<mpi.h>

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

{ int size, rank;

MPI\_Init(&argc, &argv);

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

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

printf("SIZE = %d RANK = %d\n",size,rank);

MPI\_Finalize();

return(0);

}

Обратить внимание на:

1. #include<mpi.h> - для присоединения MPI–библиотеки.
2. main(int argc, char \*\*argv) – заголовок главной функции (запись указанных параметров обязательна).
3. MPI\_Init(&argc, &argv) – MPI–функция для инициализации MPI–библиотеки; записывается в программе один раз вначале программы.
4. MPI\_Finalize() – MPI–функция (без параметров) для завершения MPI–процессов; записывается в конце программы.

Строки этих четырех пунктов в MPI–программе обязательны.

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size)- MPI–функция, записывающая в переменную size количество запущенных параллельных ветвей (см. п. Основные функции MPI).

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank)- MPI–функция, записывающая в переменную rank номер параллельной ветви (см. п. Основные функции MPI).

**ЗАДАНИЕ**

Скомпилировать и запустить программу примера 1 на 4-х и на 8-и компьютерах.

ПРИМЕР 1.2

Ветвь с номером 0 пересылает данные (в данном случае число 10) ветви с номером 3. Ветвь 3 печатает на экран свой номер и принятое число от нулевой ветви.

#include<stdio.h>

#include<mpi.h>

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

{ int size, rank, a, b;

MPI\_Status st;

MPI\_Init(&argc, &argv);

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

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

if(rank == 0)

{ a = 10;

MPI\_Send(&a,1,MPI\_INT,3,15,MPI\_COMM\_WORLD);

}

else

{ if(rank == 3)

{ MPI\_Recv(&b,1,MPI\_INT,0,15,MPI\_COMM\_WORLD,&st);

printf("Vetv= %d b= %d\n",rank,b);

}

}

MPI\_Finalize();

return(0);

}

**ЗАДАНИЕ**

Скомпилировать и запустить программу примера 2 на 4-х компьютерах.

ПРИМЕР 1.3

Ветвь с номером 0 пересылает данные (в данном случае число 10) последней ветви в множестве запущенных ветвей. Последняя ветвь печатает на экран свой номер и принятое число от нулевой ветви.

#include<stdio.h>

#include<mpi.h>

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

{ int size, rank, a, b;

MPI\_Status st;

MPI\_Init(&argc, &argv);

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

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

if(rank == 0)

{ a = 10;

MPI\_Send(&a,1,MPI\_INT,size-1,15,MPI\_COMM\_WORLD);

}

else

{ if(rank == size-1)

{ MPI\_Recv(&b,1,MPI\_INT,0,15,MPI\_COMM\_WORLD,&st);

printf("Vetv= %d b= %d\n",rank,b);

}

}

MPI\_Finalize();

return(0);

}

Обратить внимание на условие задачи и на то, что последний номер ветви в множестве запущенных ветвей равен size-1.

Это простой пример параллельной программы, которая настраивается на размер вычислительной системы, как на параметр. Т.е. программу без переделок и без перекомпиляции можно запускать на любом количестве компьютеров и она будет выдавать правильный результат. Основными параметрами в параллельной программе являются: 1) размер вычислительной системы - size; и 2) номер ветви - rank.

В данном случае этих параметров достаточно, что бы программа была настраиваемой автоматически на размер системы.

Желательно стремиться писать самонастраиваемые программы.

**ЗАДАНИЕ**

Скомпилировать и запустить программу примера 3 на 4-х и на 8-и компьютерах.

ПРИМЕР 1.4

Ветвь с номером 0 пересылает данные (в данном случае число 10) всем ветвям в множестве запущенных ветвей. Все ветви печатают на экран свой номер и принятое число от нулевой ветви.

Первый вариант программы.

#include<stdio.h>

#include<mpi.h>

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

{ int size, rank, a;

MPI\_Init(&argc, &argv);

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

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

if(rank == 0)

{ a = 10;

MPI\_Bcast(&a,1,MPI\_INT,0,MPI\_COMM\_WORLD);

}

else

{ MPI\_Bcast(&a,1,MPI\_INT,0,MPI\_COMM\_WORLD);

printf("Vetv= %d a= %d\n",rank,a);

}

MPI\_Finalize();

return(0);

}

Второй вариант программы.

#include<stdio.h>

#include<mpi.h>

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

{ int size, rank, a;

MPI\_Init(&argc, &argv);

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

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

if(rank == 0)

a = 10;

MPI\_Bcast(&a,1,MPI\_INT,0,MPI\_COMM\_WORLD);

printf("Vetv= %d a= %d\n",rank,a);

MPI\_Finalize();

return(0);

}

**ЗАДАНИЕ**

Скомпилировать и запустить обе программы примера 4 на 4-х и на 8-и компьютерах.