**Курсовой проект: Многопоточное вычисление определенного интеграла.**

 Выполнил студент группы 63604/1:

Павлов А.А.

Проверил:

Ле-Захаров А.А.

 СПБГПУ 2015

**1. Введение в MPI**

Целью настоящей работы является освоение работы с библиотекой MPI в среде разработки Visual C++ и исследование способов создания параллельных программ, работающих на нескольких компьютерах, соединенных локальной сетью и использующих обмен сообщениями для коммуникации между своими частями, расположенными на разных рабочих станциях сети.

Как правило, программирование для сетевых кластеров отличается от привычной модели программирования для многозадачных систем, построенных на базе одного или множества процессоров. Часто в таких системах необходимо обеспечить только синхронизацию процессов, и нет нужды задумываться об особых способах обмена информацией между ними, т.к. данные обычно располагаются в общей разделяемой памяти. В сети реализация такого механизма затруднительна из-за высоких накладных расходов, связанных с необходимостью предоставлять каждому процессу копию одной и той же разделяемой памяти для работы. Поэтому, обычно, при программировании для сетевых кластеров используется SPMD-технология (Single Program – Multiple Data, одна программа – множественные данные). Идея SPMD в том, чтобы поделить большой массив информации между одинаковыми процессами, которые будут вести обработку своей части данных (рис. 1).

Входные данные

Результаты

**. . .**

ДДО 1

ДПО 1

ДДО 2

ДДО N

ДПО 2

ДПО N

ДДО – данные для обработки

ДПО – данные после обработки

 **Рис. 1.** Схема взаимодействия частей SPMD-программы

В случае SPMD-подхода достаточно рассылать время от времени процессам блоки данных, которые требуют трудоемкой обработки, а затем собирать результаты их работы. Если время обработки блока данных одной машиной значительно больше, чем время пересылки этого блока по сети, то сетевая кластерная система становится очень эффективной.

Именно такой подход используется в MPI. Здесь всегда есть основной процесс, который производит распределение данных по другим машинам, а после окончания вычислений собирает результаты и показывает их пользователю. Обычно процесс-мастер после распределения данных также выполняет обработку их части, чтобы использовать ресурсы системы наиболее эффективно.

**2. Постановка задачи.**

Для освоения основных функций параллельного программирования решим задачу о нахождение значения определенного интеграла.

Напишем консольное приложение, вычисляющее определенный интеграл методом Гаусса (прямоугольников) с шагом 10-6

Ниже приведен код программы:

Подключим необходимые библиотеки

#include "stdafx.h"

#include "mpi.h"

#include "stdio.h"

#include <time.h>

#include <math.h>

Запишем отрезок интегрирования и шаг интегрирования

const double a = 0.0;//Нижний предел

const double b = 100.0;//Верхний предел

const double h = 0.000001;//Шаг интегрирования

Запишем интегрируемую функцию

double fnc(double x)//Интегрируемая функция

{

 return pow(x,5);

}

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

{

 int myrank, ranksize, i;

 clock\_t start, finish;

 MPI\_Init(&argc, &argv);//Инициализация MPI

 //Определяем свой номер в группе:

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

 //Определяем размер группы:

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

 double cur\_a, cur\_b, d\_ba, cur\_h;

 double \*sbuf = NULL;

 if (!myrank)

 {//Это процесс-мастер

 //Определяем размер диапазона для каждого процесса:

 d\_ba = (b - a) / ranksize;

 sbuf = new double[ranksize \* 3];

 cur\_a = a;

 cur\_h = h;

 for (i = 0; i<ranksize; i++)

 {

 cur\_b = cur\_a + d\_ba - h;

 sbuf[i \* 3] = cur\_a;

 sbuf[i \* 3 + 1] = cur\_b;

 sbuf[i \* 3 + 2] = h;

 cur\_a += d\_ba;

 }

 }

 double rbuf[3];

 start = clock();

 //Рассылка всем процессам, включая процесс-мастер

 //начальных данных для расчета:

 MPI\_Scatter(sbuf, 3, MPI\_DOUBLE, rbuf, 3, MPI\_DOUBLE, 0,

 MPI\_COMM\_WORLD);

 if (sbuf) delete[]sbuf;

 cur\_a = rbuf[0]; cur\_b = rbuf[1]; cur\_h = rbuf[2];

 //Расчет интеграла в своем диапазоне, выполняют все

 //процессы:

 double s = 0;

 printf("Process %d. A=%.4f B=%.4f h=%.10f\n",

 myrank, cur\_a, cur\_b, cur\_h);

 for (cur\_a += cur\_h; cur\_a <= cur\_b; cur\_a += cur\_h)

 s += cur\_h\*fnc(cur\_a);

 rbuf[0] = s;

 if (!myrank) sbuf = new double[ranksize];

 //Собираем значения интегралов от процессов:

 MPI\_Gather(rbuf, 1, MPI\_DOUBLE, sbuf, 1, MPI\_DOUBLE, 0,

 MPI\_COMM\_WORLD);

 if (!myrank)

 {//Это процесс-мастер

 //Суммирование интегралов, полученных каждым

 //процессом:

 for (i = 0, s = 0; i<ranksize; i++) s += sbuf[i];

 finish = clock();

 //Печать результата:

 printf("Integral value: %.4f\n", s);

 printf("Time: %.4f\n", (double)(finish - start) / CLOCKS\_PER\_SEC);

 delete[]sbuf;

 }

 MPI\_Finalize();//Завершение работы с MPI

 getchar();

 return 0;

}

Далее приведем значения интеграла и время которое было затрачено на расчет



Рис. 2 Приложение считает интеграл на 1 ядре



Рис. 3 Приложение считает интеграл на 2 ядрах

Получили зависимость времени расчета от числа потоков (табл. 1)

|  |  |
| --- | --- |
| Количество потоков | Время выполнения |
| 1 | 9.54  |
| 2 | 6.23 |

Таблица 1 Зависимость времени выполнения операции от числа используемых ядер

**Заключение.**

В данном курсовом проекте было реализовано вычисление определенного интеграла, с помощью библиотеки MPI, благодаря чему удалось существенно увеличить скорость вычисления значения интеграла.