МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Кафедра «Вычислительная техника»

Дисциплина «Высокопроизводительные вычисления»

**Лабораторная работа №4.**

**Исследование кластерных реализаций алгоритма численного интегрирования**

Выполнил:

студент группы ИВТАПбд-31

Кондратьев П.С.

Проверил:

Негода В. Н.

Ульяновск, 2019

# Цель работы:

Исследование кластерных реализаций алгоритма численного интегрирования на компьютерах общего пользования. Исследование зависимости коэффициентов ускорения при варьировании числа потоков и времени работы от увеличения гранулярности.

Кластер основан на Message Passing Interface — программный интерфейс (API) для передачи информации, который позволяет обмениваться сообщениями между процессами, выполняющими одну задачу. Разработан Уильямом Гроуппом, Эвином Ласком (англ.) и другими.

MPI является наиболее распространённым стандартом интерфейса обмена данными в параллельном программировании, существуют его реализации для большого числа компьютерных платформ. Используется при разработке программ для кластеров и суперкомпьютеров. Основным средством коммуникации между процессами в MPI является передача сообщений друг другу.

Стандартизацией MPI занимается MPI Forum. В стандарте MPI описан интерфейс передачи сообщений, который должен поддерживаться как на платформе, так и в приложениях пользователя. В настоящее время существует большое количество бесплатных и коммерческих реализаций MPI. Существуют реализации для языков Фортран 77/90, Java, Си и C++.

В первую очередь MPI ориентирован на системы с распределенной памятью, то есть когда затраты на передачу данных велики, в то время как OpenMP ориентирован на системы с общей памятью (многоядерные с общим кешем). Обе технологии могут использоваться совместно, чтобы оптимально использовать в кластере многоядерные системы.

В MPI 1.1 (опубликован 12 июня 1995 года, первая реализация появилась в 2002 году) поддерживаются следующие функции:

* передача и получение сообщений между отдельными процессами;
* коллективные взаимодействия процессов;
* взаимодействия в группах процессов;
* реализация топологий процессов;

В MPI 2.0 (опубликован 18 июля 1997 года) дополнительно поддерживаются следующие функции:

* динамическое порождение процессов и управление процессами;
* односторонние коммуникации (Get/Put);
* параллельный ввод и вывод;
* расширенные коллективные операции (процессы могут выполнять коллективные операции не только внутри одного коммуникатора, но и в рамках нескольких коммуникаторов).

**Функционирование интерфейса**

Базовым механизмом связи между MPI процессами является передача и приём сообщений. Сообщение несёт в себе передаваемые данные и информацию, позволяющую принимающей стороне осуществлять их выборочный приём:

* отправитель — ранг (номер в группе) отправителя сообщения;
* получатель — ранг получателя;
* признак — может использоваться для разделения различных видов сообщений;
* коммуникатор — код группы процессов.

Операции приёма и передачи могут быть блокирующимися и неблокирующимися. Для неблокирующихся операций определены функции проверки готовности и ожидания выполнения операции.

Другим способом связи является удалённый доступ к памяти (RMA), позволяющий читать и изменять область памяти удалённого процесса. Локальный процесс может переносить область памяти удалённого процесса (внутри указанного процессами окна) в свою память и обратно, а также комбинировать данные, передаваемые в удалённый процесс с имеющимися в его памяти данными (например, путём суммирования). Все операции удалённого доступа к памяти не блокирующиеся, однако, до и после их выполнения необходимо вызывать блокирующиеся функции синхронизации.

**Ниже приведен примеры работы на тестирующем компьютере:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Потоки** | **100** | **1000** | **10000** | **100000** | **1000000** |
| **1** | 0,00000572 | 0,00006001 | 0,00052999 | 0,00522709 | 0,05445862 |
| **2**  **К, уск** | 0,00000528  1 | 0,00005621  1 | 0,00053859  1 | 0,005277791  1 | ,05552857  1 |
| **3**  **К, уск** | 0,00000620  1 | 0,00005221  1 | 0,00052047  1 | 0,00524733  1 | 0,05414915  1 |
| **4**  **К, уск** | 0,00000525  1 | 0,00007221  1 | 0,00052381  1 | 0,00522256  1 | 0,05221415  1 |
| **5**  **К, уск** | 0,00000468  1 | 0 ,00005627  1 | 0,00063409  1 | 0,00563741  1 | 0,05731940  1 |
| **6**  **К, уск** | 0,00000525  1 | 0,00010550  1 | 0,00073859  1 | 0,00551558  1 | 0,05224752  1 |
| **7**  **К, уск** | 0,00000548  1 | 0,00006342  1 | 0,00106645  1 | 0,00527382  1 | 0,05229402  1 |
| **8**  **К, уск** | 0,00000525  1 | 0,00005198  1 | 0,00053954  1 | 0,00523615  1 | 0,05231857  1 |

**Рис. 1.** Результаты замеров времени (c) вычисления

**Рис. 2.** Графики коэффициентов ускорения при варьировании потоков

**Ниже приведены примеры работы на кластере Beowulf:**

Кластер Beowulf состоит из 4 компьютеров (1 из них – является консолью управления), 3 из которых является рабочими узлами. Все они имеют операционную систему Debian.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Потоки** | **100** | **1000** | **10000** | **100000** | **1000000** |
| **1** | 0,00000540 | 0,00005227 | 0,00052217 | 0,00524121 | 0,05235737 |
| **2**  **К, уск** | 0,00010064  0,5 | 0,00006270  1 | 0,000359351  1,49 | 0,00271885  1,93 | 0,02628649  2,09 |
| **3**  **К, уск** | 0,00012649  0,4 | 0,00005221  1 | 0,00028518  1,82 | 0,00192027  2,74 | 0,01757644  3,17 |
| **4**  **К, уск** | 0,00043522  0,013 | 0,00044615  0,16 | 0,00056545  1,07 | 0,00174583  3,02 | 0,01352575  4,25 |
| **5**  **К, уск** | 0,00043613  0,04 | 0,00044578  0,11 | 0,00060566  1,2 | 0,00151801  3,64 | 0,01351727  4,24 |
| **6**  **К, уск** | 0,00043954  0,01 | 0,00044713  0,43 | 0,00052569  1,14 | 0,00135274  4,07 | 0,01919363  2,08 |
| **7**  **К, уск** | 0,00043363  0,01 | 0,00044067  0,11 | 0,00050818  1,02 | 0,00132078  3,99 | 0,00825686  6,76 |
| **8**  **К, уск** | 0,00048632  0,01 | 0,00044199  0,13 | 0,00050246  1,24 | 0,00117275  5,21 | 0,00876077  7,65 |

**Рис. 3.** Результаты замеров времени (c) вычисления

**Рис. 4.** Графики коэффициентов ускорения при варьировании потоков

**Код программы:**

#include "/usr/include/mpi/mpi.h"

#include <stdio.h>

#include <math.h>

#include <clocale>

#include <iostream>

#include <clocale>

using namespace std;

static int KRepeat = 10;

static int IRepeat = 100;

double h;

double f(double x)

{

return (exp(x) + exp(-x)) / 2;

}

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

{

setlocale(LC\_CTYPE, "rus");

int myid, numprocs, i;

double myy, y, h, sum, x;

int namelen;

char processor\_name[MPI\_MAX\_PROCESSOR\_NAME];

double Integral = 0.0, Res = 0.0;

double tmin = 100000.0, t;

int num\_hs;

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

MPI\_Comm\_size(MPI\_COMM\_WORLD, &numprocs); // Получить общее число процессов

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &myid); // Получить номер текущего процесса

for (num\_hs = 1000000; num\_hs >= 100; num\_hs /= 10)

{

printf("Granulyarnost: %d\n", num\_hs);

sum = 0.0;

h = 1.0 / (double)num\_hs;

int n = num\_hs - 1

double t1, t2;

// чисто последовательное исполнение

t1 = MPI\_Wtime();

sum = 0.5 \* (f(1.0) + f(2.0));

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

{

x = 1 + (double)i \* h;

sum = sum + f(x);

}

y = h \* sum;

t2 = MPI\_Wtime();

printf("Bez rassparall: t = %4.8f c;\n", t2 - t1);

Res = 0.0;

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

{

tmin = 100000.0;

Integral = 0.0;

t1 = MPI\_Wtime();

for (int i1 = 0; i1 < IRepeat; i1++)

{

sum = 0.5 \* (f(1.0) + f(2.0));

// Рассылка интервалов интегрирования процессам

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

for (i = myid + 1; i <= n; i += numprocs)

{

x = 1 + (double)i \* h;

sum = sum + f(x);

}

myy = h \* sum;

// Сбор результатов со всех процессов и сложение

MPI\_Reduce(&myy, &y, 1, MPI\_DOUBLE, MPI\_SUM, 0, MPI\_COMM\_WORLD);

Integral += y;

}

t2 = MPI\_Wtime();

t = (double)(t2 - t1) / IRepeat;

if (t < tmin)

tmin = t;

Integral /= IRepeat;

Res += Integral;

}

Res /= KRepeat;

if (myid == 0)

printf("kol potokov(%d): t = %4.8f c;\n", numprocs, tmin);

}

MPI\_Finalize();// Освобождение подистемы MPI

return 0;

}

# 

# Приложение 1.

Характеристика, тестируемого ЭВМ;



