МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**“УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”**

Факультет информационных систем и технологий

Кафедра “ Вычислительная техника”

Дисциплина “Высокопроизводительные вычисления”

**Лабораторная работа №4**

Исследование кластерных реализаций алгоритма численного интегрирования

Вариант 3

Выполнил:

студент гр. ИВТАПбд-31

Вершинин Д. В.

Проверил:

Негода В.Н.

Ульяновск, 2019

Для данной лабораторной работы необходимо запустить код, вычисляющий определенный интеграл из лабораторной работы 3 на кластере.

Кластер состоит из 5 компьютеров:

10.3.20.240 – консоль управления

10.3.20.241

10.3.20.242

10.3.20.243

10.3.20.247 – рабочие узлы

Все рабочие узлы должны быть включены и загружены операционной системой Debian. Для запуска программы заходим в консоль управления и помещаем файл с исходным кодом в домашнем каталоге. Для компиляции программы используем компилятор g++. Для поддержки библиотеки OpenMP добавим к инструкции опцию -fopenmp.

*g++ /srv/beowulf/Source3.cpp -fopenmp*

Запустим полученный двоичный код программы с помощью mpiexec.

*mpiexec -f ~/hosts -iface eth0 -n 14 /srv/beowulf/a.out*

С помощью опции -n можно регулировать количество используемых процессоров.

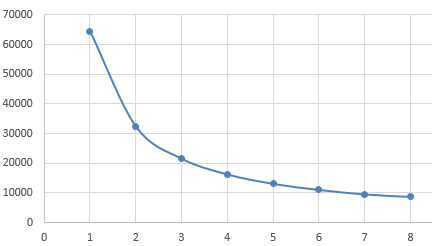
Вывод программы:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Потоки | Гранулярность | | | | |
| 100 | 1000 | 10000 | 100000 | 1000000 |
| 1 | 6.63 | 64.82 | 645.26 | 6450.65 | 64524.28 |
| 2 | 98.3 | 127.89 | 418.63 | 3329.3 | 32480.45 |
| 3 | 108.74 | 128.41 | 323.75 | 2263.89 | 21628.62 |
| 4 | 192.91 | 207.61 | 352.94 | 1805.5 | 16320.59 |
| 5 | 190.2 | 203.06 | 320.22 | 1491.88 | 13155.1 |
| 6 | 196.68 | 208.71 | 304.17 | 1272.24 | 11125.5 |
| 7 | 193.36 | 201.97 | 284.92 | 1213.99 | 9618.63 |
| 8 | 202.24 | 210.12 | 292.88 | 1102.86 | 8766.06 |

Таблица ускорений (Speed up).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Потоки | Гранулярность | | | | |
| 100 | 1000 | 10000 | 100000 | 1000000 |
| 1 | 1.115 | 1.008 | 1.006 | 0.999 | 1.024 |
| 2 | 0.07 | 0.522 | 1.533 | 1.932 | 2.085 |
| 3 | 0.061 | 0.505 | 1.995 | 2.833 | 3.163 |
| 4 | 0.036 | 0.317 | 1.814 | 3.549 | 4.196 |
| 5 | 0.038 | 0.321 | 2.041 | 4.348 | 5.055 |
| 6 | 0.036 | 0.311 | 2.105 | 5.048 | 5.968 |
| 7 | 0.037 | 0.323 | 2.248 | 5.296 | 7.622 |
| 8 | 0.035 | 0.31 | 2.503 | 5.805 | 7.958 |

График зависимости времени работы от числа потоков при гранулярности 1000000.



Как видно из таблиц, при низких значениях гранулярности наблюдается снижение производительности. Это можно объяснить тем, что затраты на распределение задачи между потоками, рассылка данных по сети превосходят получаемую пользу.

Исходя из этого можно сказать, что затраты на коммуникацию между ЭВМ в кластере, оказывают значительное влияние на производительность при малых объемах вычислений.

Также можно сказать о том, что время работы программы при распараллеливании зависит от скорости передачи данных между машинами в кластере.

**Исходный код**

#include <iostream>

#include <cmath>

#include <mpi.h>

#include <time.h>

using namespace std;

float f(float x) {

return cos(x);

}

double integral(int n, double a, double b)

{

int i;

double result, h;

result = 0;

h = (b - a) / n;

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

{

result += f(a + h \* (i + 0.5));

}

result \*= h;

return result;

}

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

double a = 0, b = 1;

double res = 0;

int process\_id;

int ierr;

int process\_num;

MPI\_Status status;

int master = 0;

MPI\_Init(&argc, &argv);

ierr = MPI\_Comm\_rank(MPI\_COMM\_WORLD, &process\_id); //

ierr = MPI\_Comm\_size(MPI\_COMM\_WORLD, &process\_num);

double proc\_arg[3];

double delta = fabs(a - b) / (process\_num);

for(int n = 100; n <= 1000000; n \*= 10){

clock\_t start = clock();

integral(n, a, b);

clock\_t end = clock();

float duringBez = ((double)(end - start) / CLOCKS\_PER\_SEC);

double proc\_n = (double)n / (process\_num);

int tag = 1;

float minTime = 1<<20;

for(int j = 0; j < 100; j++){

start = clock();

if (process\_id == master) {

for (int process = 1; process < process\_num; process++)

{

proc\_arg[0] = a + delta \* (process);

proc\_arg[1] = proc\_arg[0] + delta;

proc\_arg[2] = proc\_n;

ierr = MPI\_Send(proc\_arg, 3, MPI\_DOUBLE, process, tag, MPI\_COMM\_WORLD);

}

}

else {

ierr = MPI\_Recv(proc\_arg, 3, MPI\_DOUBLE, master, tag, MPI\_COMM\_WORLD, &status);

//

}

ierr = MPI\_Barrier(MPI\_COMM\_WORLD);

if (process\_id != master) {

double res\_l = integral(proc\_arg[2], proc\_arg[0], proc\_arg[1]);

int target = master;

tag = 2;

ierr = MPI\_Send(&res\_l, 1, MPI\_DOUBLE, target, tag, MPI\_COMM\_WORLD);

}

else {

res = integral(proc\_n, a, a + delta); // master process

for (int i = 0; i < process\_num - 1; i++) {

double res\_l = 0;

tag = 2;

ierr = MPI\_Recv(&res\_l, 1, MPI\_DOUBLE, MPI\_ANY\_SOURCE, tag, MPI\_COMM\_WORLD, &status);

res += res\_l;

}

end = clock();

float during = ((double)(end - start) / CLOCKS\_PER\_SEC);

if(during < minTime)

minTime = during;

}

}

if (process\_id == master)

cout << "res " << res << "; time " << minTime \* 1e6<< " speedUp: " << duringBez / minTime << endl;

}

ierr = MPI\_Finalize();

return 0;

}