МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное   
образовательное учреждение высшего образования  
«Самарский национальный исследовательский университет   
имени академика С.П. Королева»

(Самарский университет)

Институт информатики и кибернетики

Кафедра технической кибернетики

ЛАБОРАТОРНАЯ РАБОТА № 3

по курсу   
Параллельное программирование

Группа 6408

Студент Н.В. Носов

(*подпись*)

Преподаватель,

к.ф.-м.н. Е.С. Козлова

(*подпись*)

Самара 2023

ЗАДАНИЕ

**Произвести запуск программ** для поиска суммы векторов**, которые используют технологии MPI и OpenMP, на различном количестве процессоров (потоков).**

**В ходе анализа работы программы оценить время ее выполнения на различном количестве исполняющих нитей (процессов). Оценить влияние различных функций и директив(опций) на скорость работы приложений.**

**Для дальнейшего анализа технологий искусственно увеличить соотношение вычислений и операций по обеспечению параллелизма путем повторения функции сложения векторов Q раз. В ходе анализа работы программ также оценить время их выполнения на различном количестве исполняющих нитей (процессов). Оценить влияние различных функций и директив на скорость работы приложений.**

**Реализовать последовательный вариант программы. Оценить время ее выполнения для обычного и "усложненного" вариантов. Рассчитать ускорение параллельных программ относительно последовательного варианта.**

Таблица 1 – Исходные данные на ЛР № 3

|  |  |
| --- | --- |
| Тип | double |
| N | 3200000(+13) |
| Количество процессов/потоков | [4, 8, 16] |
| Q | 28 |

ВВЕДЕНИЕ

Текст.

­­­­

ЭКСПЕРИМЕНТАЛЬНЫЕ ИССЛЕДОВАНИЯ

2.1 Результаты работы программ без параметра Q

В ходе исследования времени работы программ здесь и далее проводилось усреднение не менее чем по 12 запускам.

На рисунках 1-5 представлены скрины запуска и работы программ без параметра Q.

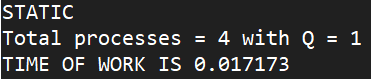


Рисунок 1 – Пример работы программы OpenMP на 4 процессах для static

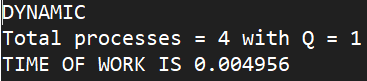


Рисунок 2 – Пример работы программы OpenMP на 4 процессах для dynamic

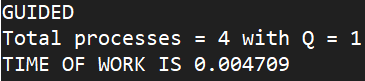


Рисунок 3 – Пример работы программы OpenMP на 4 процессах для guided

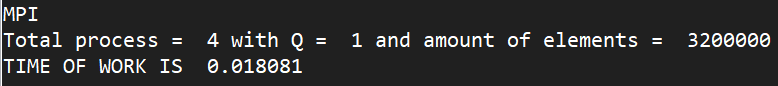


Рисунок 4 – Пример работы программы MPI на 4 процессах для кратных размерностей

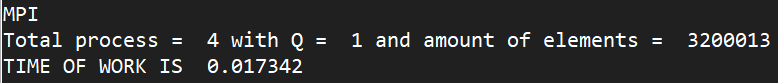


Рисунок 5 – Пример работы программы MPI на 4 процессах для некратных размерностей

Последовательная программа представляла собой программу вычисления суммы двух массивов одним процессом. Время работы последовательной программы составило 0.018333 с.

В таблицах 3-4 представлено время выполнения параллельных программ и их ускорение по сравнению с последовательным вариантом.

Таблица 3 – Время работы параллельных программ без параметра Q

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Кол-во процессов, шт | Время для OpenMP, с | | | Время для MPI, с | |
| static | dynamic | quided | кратное | некратное |
| 4 | 0.017173 | 0.004956 | 0.004709 | 0.018081 | 0.017342 |
| 8 | 0.015990 | 0.004260 | 0.004034 | 0.015487 | 0.014336 |
| 16 | 0.018359 | 0.006348 | 0.006808 | 0.014106 | 0.017783 |

Таблица 4 – Ускорение параллельных программ без параметра Q

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Кол-во процессов, шт | Ускорение для OpenMP | | | Ускорение для MPI | |
| static | dynamic | quided | кратное | некратное |
| 4 | 1.067548 | 3.699153 | 3.893183 | 1.013937 | 1.057145 |
| 8 | 1.146529 | 4.303521 | 4.544621 | 1.183767 | 1.278809 |
| 16 | 0.998584 | 2.887996 | 2.692861 | 1.299660 | 1.030928 |

На рисунке 6 приведен график зависимости времени работы программ от количества процессов. На рисунке 7 приведен график зависимости ускорения программ от количества процессов.

Рисунок 6 – Время работы программ без параметра Q

Рисунок 7 – Ускорение программ без параметра Q

2.3 Результаты работы программ с параметром Q

На рисунках 7-11 представлены скрины запуска и работы программ с параметром Q.

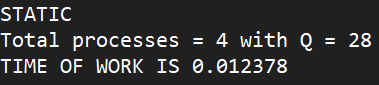


Рисунок 7 – Пример работы программы OpenMP на 4 процессах для static

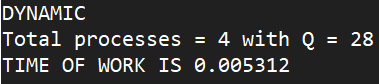


Рисунок 8 – Пример работы программы OpenMP на 4 процессах для dynamic

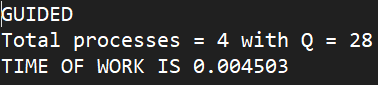


Рисунок 9 – Пример работы программы OpenMP на 4 процессах для guided

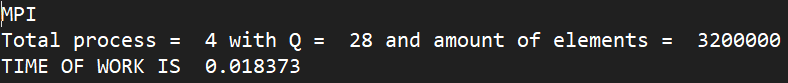


Рисунок 10 – Пример работы программы MPI на 4 процессах для кратных размерностей

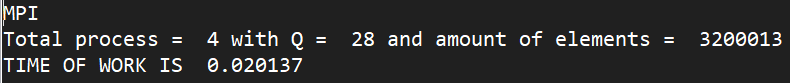


Рисунок 11 – Пример работы программы MPI на 4 процессах для некратных размерностей

Усложненная последовательная программа представляла собой программу вычисления суммы двух массивов одним процессом. Время работы последовательной программы составило 0.431667 с.

В таблицах 5-6 представлено время выполнения параллельных программ и их ускорение по сравнению с последовательным вариантом.

Таблица 5 – Время работы параллельных программ с параметром Q

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Кол-во процессов, шт | Время для OpenMP, с | | | Время для MPI, с | |
| static | dynamic | quided | кратное | некратное |
| 4 | 0.012378 | 0.005312 | 0.004503 | 0.018373 | 0.020137 |
| 8 | 0.010413 | 0.003707 | 0.004557 | 0.017438 | 0.016778 |
| 16 | 0.012885 | 0.005959 | 0.018150 | 0.014566 | 0.015447 |

Таблица 6 – Ускорение параллельных программ с параметром Q

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Кол-во процессов, шт | Ускорение для OpenMP | | | Ускорение для MPI | |
| static | dynamic | quided | кратное | некратное |
| 4 | 34.87373 | 81.26261 | 95.86209 | 23.49464 | 21.43651 |
| 8 | 41.45462 | 116.4465 | 94.72614 | 24.75439 | 25.72816 |
| 16 | 33.50151 | 72.4395 | 23.78331 | 29.63525 | 27.94504 |

На рисунке 12 приведен график зависимости времени работы программ от количества процессов. На рисунке 13 приведен график зависимости ускорения программ от количества процессов.

Рисунок 12 – Время работы программ с параметром Q

Рисунок 13 – Ускорение программ с параметром Q

**ВЫВОДЫ:**

Из полученных результатов видно, что:

1. Как влияют варианты распараллеливания для обеих технологий? Почему?
2. Какое максимальное ускорение/замедление получено для обеих технологий без параметра Q и с параметром Q? Почему?
3. Как влияет количество процессов для обеих технологий без параметра Q и с параметром Q? Кратно увеличивается/уменьшается время для обеих технологий без параметра Q и с параметром Q? Почему?
4. Как соотносится между собой время работы обеих технологий без параметра Q и с параметром Q? Почему? Какое влияние оказывает параметр Q?

ЗАКЛЮЧЕНИЕ

Цель лабораторной работы – написать параллельные программы суммы двух векторов с использованием технологий MPI и OpenMP и сравнить время выполнения с длительностью последовательной программы достигнута. Показано, что использование параллельных технологий для данного типа программ обосновано/не требуется, в виду того, что ?. Лучше всего себя показала технология ? с использованием ?. Это можно объяснить ?.

В ходе выполнения лабораторной работы я изучил(а) основы MPI и OpenMP, приобрел(а) навыки по написанию параллельных программ с использованием вышеперечисленных технологий. Наиболее сложной частью выполнения лабораторной работы было ?. Интерес вызвали ?.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Ваши источник из Введения.
2. Козлова, Е.С. Лабораторные работы по курсу «Параллельное программирование»: Методические указания [Текст] / Сост. Е.С. Козлова, А.С. Широканев − Самара, 2019. – 61 с.
3. оеводин, В. В. Параллельные вычисления [Текст] / В. В. Воеводин, Вл. В. Воеводин. — СПб.: БХВ-Петербург, 2002. — 608 с.
4. Богачёв К.Ю. Основы параллельного программирования: учебное пособие, 2-е изд. [Текст] / К. Ю. Богачёв ‑ М. : БИНОМ. Лаборатория знаний, 2013. ‑ 344 с.
5. Гергель, В. П. Теория и практика параллельных вычислений, 2-е изд. [Текст] / B. II. Гергель. — М.: Интуит. 2016. - 500 с.
6. Боресков А.В. Параллельные вычисления на GPU. Архитектура и программная модель CUDА Учеб. пособие [Текст] / А.В. Боресков ‑ М.: Издательство Московского университета, 2012. - 336 с.
7. Библиографическое описание документа. Общие требования и правила составления [Электронный ресурс] / сост.: В.С. Крылова, С.М. Григорьевская, Е.Ю. Кичигина // Официальный интернет-сайт научной библиотеки Томского государственного университета. – Электрон. дан. – Томск, [2010]. – <http://www.lib.tsu.ru/win/produkzija/metodichka/metodich.html> (дата обращения: 10.09.2019).

ПРИЛОЖЕНИЕ А

Код программы с технологией MPI

#include "stdlib.h"

#include "stdio.h"

#include "mpi.h"

#define NMAX 3200000

#define ITERATIONS 12

#define Q 28

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

{

double \*a = (double \*)malloc(sizeof(double) \* NMAX);

double \*b = (double \*)malloc(sizeof(double) \* NMAX);

double \*c = (double \*)malloc(sizeof(double) \* NMAX);

int ProcRank, ProcNum, i;

MPI\_Status Status;

double st\_time, end\_time;

MPI\_Init(&argc, &argv);

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

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

int count = NMAX / ProcNum;

int remainder = NMAX % ProcNum;

if (ProcRank == 0)

{

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

{

a[i] = rand();

b[i] = rand();

}

}

double \*a\_loc = (double \*)malloc(sizeof(double) \* (count + (ProcRank < remainder ? 1 : 0)));

double \*b\_loc = (double \*)malloc(sizeof(double) \* (count + (ProcRank < remainder ? 1 : 0)));

double \*c\_loc = (double \*)malloc(sizeof(double) \* (count + (ProcRank < remainder ? 1 : 0)));

st\_time = MPI\_Wtime();

int t;

for (t = 0; t < ITERATIONS; t++)

{

int \*sendcounts = (int \*)malloc(sizeof(int) \* ProcNum);

int \*displs = (int \*)malloc(sizeof(int) \* ProcNum);

if (ProcRank == 0)

{

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

{

sendcounts[i] = count + 1;

}

for (i = remainder; i < ProcNum; i++)

{

sendcounts[i] = count;

}

displs[0] = 0;

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

{

displs[i] = displs[i - 1] + sendcounts[i - 1];

}

}

MPI\_Scatterv(a, sendcounts, displs, MPI\_INT, a\_loc, count + (ProcRank < remainder ? 1 : 0), MPI\_INT, 0, MPI\_COMM\_WORLD);

MPI\_Scatterv(b, sendcounts, displs, MPI\_INT, b\_loc, count + (ProcRank < remainder ? 1 : 0), MPI\_INT, 0, MPI\_COMM\_WORLD);

for (i = 0; i < count + (ProcRank < remainder ? 1 : 0); i++)

{

c\_loc[i] = 0;

int q;

for (q = 0; q < Q; q++)

{

c\_loc[i] += a\_loc[i] + b\_loc[i];

}

}

MPI\_Barrier(MPI\_COMM\_WORLD);

MPI\_Gatherv(c\_loc, count + (ProcRank < remainder ? 1 : 0), MPI\_INT, c, sendcounts, displs, MPI\_INT, 0, MPI\_COMM\_WORLD);

free(sendcounts);

free(displs);

}

end\_time = MPI\_Wtime();

if (ProcRank == 0)

{

printf("MPI");

printf("\nTotal process = % d with Q = % d and amount of elements = % d", ProcNum, Q, NMAX);

printf("\nTIME OF WORK IS % f", (end\_time - st\_time) / ITERATIONS);

}

free(a);

free(b);

free(c);

free(a\_loc);

free(b\_loc);

free(c\_loc);

MPI\_Finalize();

return 0;

}

ПРИЛОЖЕНИЕ Б

Код программы с технологией OpenMP для static

#include <omp.h>

#include <math.h>

#include <stdlib.h>

#include <stdio.h>

#define CHUNK 10

#define NMAX 3200000

#define Q 28

#define NUM\_THREADS 4

#define ITERATION 12

int main()

{

omp\_set\_num\_threads(NUM\_THREADS);

double \*a = (double \*)malloc(sizeof(double) \* NMAX);

double \*b = (double \*)malloc(sizeof(double) \* NMAX);

double \*c = (double \*)malloc(sizeof(double) \* NMAX);

double s = 0.0;

int m;

for (m = 0; m < NMAX; m++)

{

a[m] = rand();

b[m] = rand();

}

double st\_time, end\_time;

s = 0.0;

int j;

int q;

for (j = 0; j < ITERATION; j++)

{

int i;

st\_time = omp\_get\_wtime();

#pragma omp parallel for schedule(static, CHUNK) private(i)

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

{

for (q = 0; q < Q; q++)

c[i] = a[i] + b[i];

}

end\_time = omp\_get\_wtime();

end\_time = end\_time - st\_time;

s += end\_time;

}

printf("STATIC\nTotal processes = %d with Q = %d", NUM\_THREADS, Q);

printf("\nTIME OF WORK IS %f ", s / ITERATION);

free(c);

free(a);

free(b);

return 0;

}

ПРИЛОЖЕНИЕ В

Код программы с технологией OpenMP для dynamic

#include <math.h>

#include <stdlib.h>

#include <omp.h>

#include <stdio.h>

#define CHUNK 100

#define NMAX 3200000

#define Q 28

#define NUM\_THREADS 4

#define ITERATION 12

int main()

{

omp\_set\_num\_threads(NUM\_THREADS);

double \*a = (double \*)malloc(sizeof(double) \* NMAX);

double \*b = (double \*)malloc(sizeof(double) \* NMAX);

double \*c = (double \*)malloc(sizeof(double) \* NMAX);

double s = 0.0;

int m;

for (m = 0; m < NMAX; m++)

{

a[m] = rand();

b[m] = rand();

}

double st\_time, end\_time;

s = 0.0;

int j;

int q;

for (j = 0; j < ITERATION; j++)

{

int i;

st\_time = omp\_get\_wtime();

#pragma omp parallel for schedule(dynamic, CHUNK) private(i)

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

{

for (q = 0; q < Q; q++)

c[i] = a[i] + b[i];

}

end\_time = omp\_get\_wtime();

end\_time = end\_time - st\_time;

s += end\_time;

}

printf("DYNAMIC\nTotal processes = %d with Q = %d", NUM\_THREADS, Q);

printf("\nTIME OF WORK IS %f ", s / ITERATION);

free(c);

free(a);

free(b);

return 0;

}

ПРИЛОЖЕНИЕ Г

Код программы с технологией OpenMP для guided

#include <math.h>

#include <stdlib.h>

#include <omp.h>

#include <stdio.h>

#define CHUNK 10

#define NMAX 3200000

#define Q 28

#define NUM\_THREADS 4

#define ITERATION 12

int main()

{

omp\_set\_num\_threads(NUM\_THREADS);

double \*a = (double \*)malloc(sizeof(double) \* NMAX);

double \*b = (double \*)malloc(sizeof(double) \* NMAX);

double \*c = (double \*)malloc(sizeof(double) \* NMAX);

double s = 0.0;

int m;

for (m = 0; m < NMAX; m++)

{

a[m] = rand();

b[m] = rand();

}

double st\_time, end\_time;

s = 0.0;

int j;

int q;

for (j = 0; j < ITERATION; j++)

{

int i;

st\_time = omp\_get\_wtime();

#pragma omp parallel for schedule(guided, CHUNK) private(i)

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

{

for (q = 0; q < Q; q++)

c[i] = a[i] + b[i];

}

end\_time = omp\_get\_wtime();

end\_time = end\_time - st\_time;

s += end\_time;

}

printf("GUIDED\nTotal processes = %d with Q = %d", NUM\_THREADS, Q);

printf("\nTIME OF WORK IS %f ", s / ITERATION);

free(c);

free(a);

free(b);

return 0;

}

ПРИЛОЖЕНИЕ Д

Код последовательной программы

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define N\_SIZE 3200000

#define ITERATION 12

#define Q 28

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

int i, j, count;

double sum, total\_time = 0;

double\* a = new double[N\_SIZE];

double\* b = new double[N\_SIZE];

double\* c = new double[N\_SIZE];

for (i = 0; i < N\_SIZE; ++i) {

a[i] = 1;

b[i] = 1;

}

clock\_t start\_time, end\_time;

// Последовательное суммирование

for (count = 0; count < ITERATION; count++) {

start\_time = clock();

for (i = 0; i < N\_SIZE; i++) {

for (j = 0; j < Q; ++j) {

c[i] = a[i] + b[i];

}

}

end\_time = clock();

end\_time = end\_time - start\_time;

total\_time += end\_time;

}

sum = sum / (ITERATION \* Q);

printf("\nConsistent Q: %d", Q);

printf("\nTotal Sum = %10.2f", sum);

printf("\nTIME OF WORK IS %f ", total\_time / CLOCKS\_PER\_SEC / ITERATION);

return 0;

}