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 6409

Студент П.А. Маштаков

(*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 | float |
| N | 6300000 |
| Количество процессов/потоков | 3, 9, 12 |
| Q | 26 |

ВВЕДЕНИЕ

Использование параллельных вычислительных систем становится все более актуальным направлением в современном мире. Это обусловлено наличием задач, требующих больших вычислительных ресурсов, для которых существующей вычислительной техники недостаточно. Одними из наиболее популярных технологий параллельного программирования являются технологии MPI и OpenMP [1].

В ходе выполнения лабораторной работы было рассчитано время работы программ, реализующих поиск суммы элементов векторов и использующих технологии MPI и OpenMP.

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.

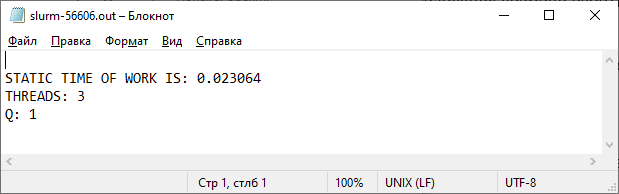


Рисунок 1 – Пример работы программы OpenMP на 3 процессах для static

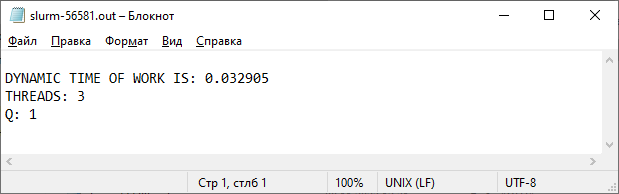


Рисунок 2 – Пример работы программы OpenMP на 3 процессах для dynamic

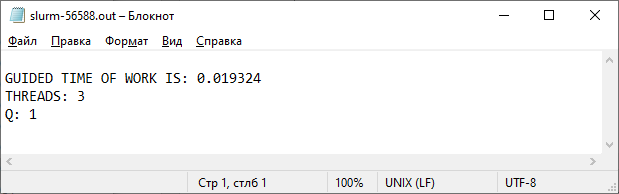


Рисунок 3 – Пример работы программы OpenMP на 3 процессах для guided

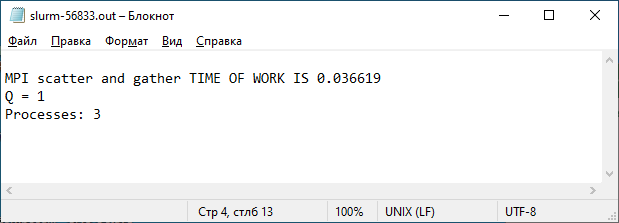


Рисунок 4 – Пример работы программы MPI на 3 процессах для кратных размерностей

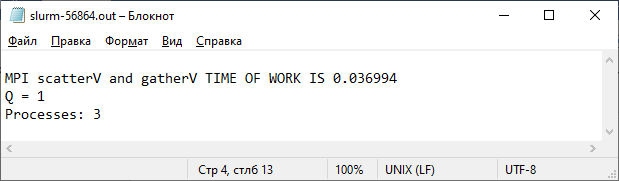


Рисунок 5 – Пример работы программы MPI на 3 процессах для некратных размерностей

Последовательная программа представляла собой поиск суммы элементов векторов. Время работы последовательной программы составило 0,0366 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 | guided | кратное | некратное |
| 3 | 0,0231 | 0,0329 | 0,0193 | 0,0363 | 0,0364 |
| 9 | 0,0097 | 0,0107 | 0,0066 | 0,0261 | 0,0260 |
| 12 | 0,0091 | 0,0087 | 0,0054 | 0,0234 | 0,0238 |

Таблица 4 – Ускорение параллельных программ без параметра Q

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Кол-во процессов, шт | Ускорение для OpenMP | | | Ускорение для MPI | |
| static | dynamic | guided | кратное | некратное |
| 3 | 1,5844 | 1,1125 | 1,8964 | 1,0082 | 1,0055 |
| 9 | 3,7734 | 3,4205 | 5,5454 | 1,4022 | 1,4076 |
| 12 | 4,0212 | 4,2069 | 6,7777 | 1,5641 | 1,5378 |

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

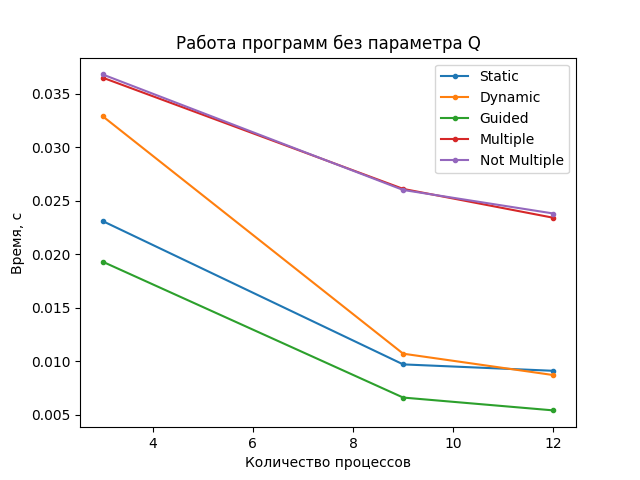


Рисунок 6 – Время работы программ без параметра Q

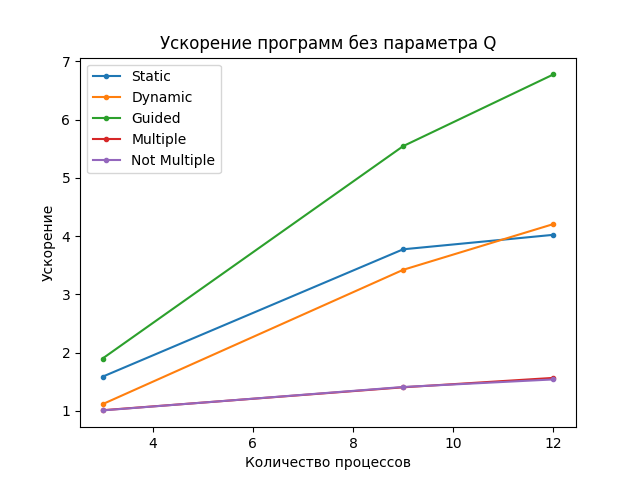


Рисунок 7 – Ускорение программ без параметра Q

2.3 Результаты работы программ с параметром Q

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

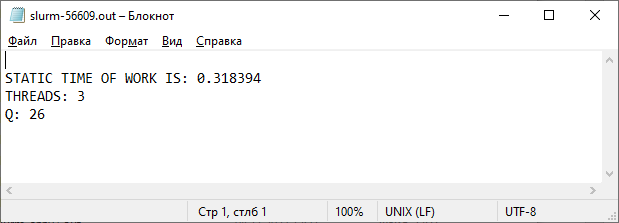


Рисунок 7 – Пример работы программы OpenMP на 3 процессах для static

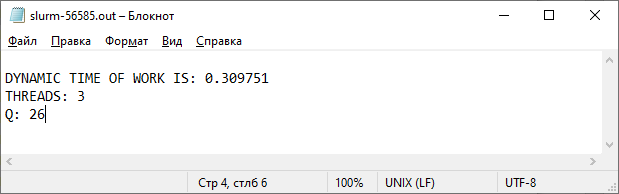


Рисунок 8 – Пример работы программы OpenMP на 3 процессах для dynamic

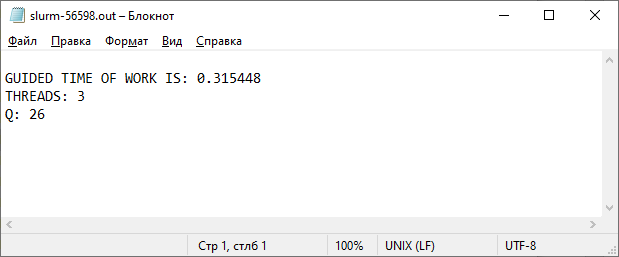


Рисунок 9 – Пример работы программы OpenMP на 3 процессах для guided

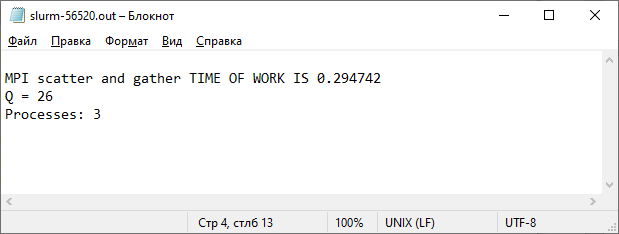


Рисунок 10 – Пример работы программы MPI на 3 процессах для кратных размерностей

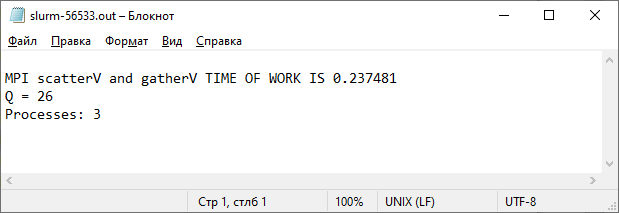


Рисунок 11 – Пример работы программы MPI на 3 процессах для некратных размерностей

Усложненная последовательная программа представляла собой поиск суммы элементов векторов с параметром пересчёта Q в цикле. Время работы последовательной программы составило 0,4908 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 | guided | кратное | некратное |
| 3 | 0,3184 | 0,3097 | 0,3154 | 0,2882 | 0,2828 |
| 9 | 0,1102 | 0,1102 | 0,0871 | 0,1070 | 0,1053 |
| 12 | 0,0825 | 0,0828 | 0,0788 | 0,0847 | 0,0842 |

Таблица 6 – Ускорение параллельных программ с параметром Q

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Кол-во процессов, шт | Ускорение для OpenMP | | | Ускорение для MPI | |
| static | dynamic | guided | кратное | некратное |
| 3 | 1,5414 | 1,5847 | 1,5561 | 1,7029 | 1,7355 |
| 9 | 4,4537 | 4,4537 | 5,6349 | 4,5869 | 4,6609 |
| 12 | 5,9490 | 5,9275 | 6,2284 | 5,7945 | 5,8289 |

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

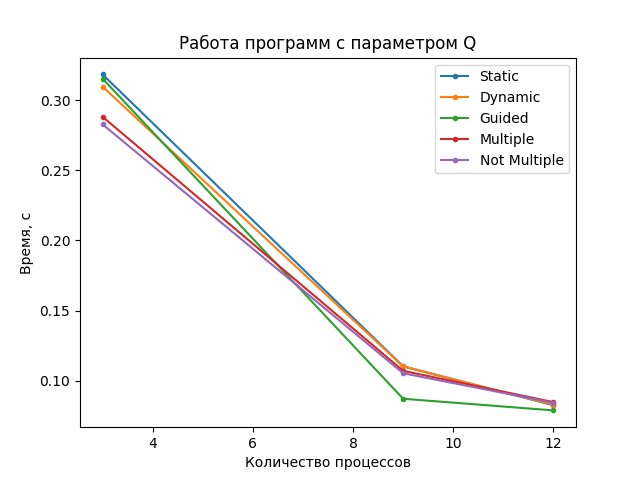


Рисунок 12 – Время работы программ с параметром Q

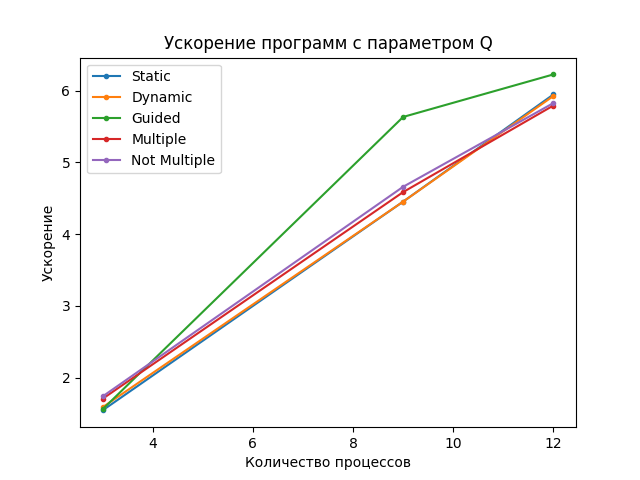


Рисунок 13 – Ускорение программ с параметром Q

**ВЫВОДЫ:**

Из полученных результатов видно, что:

1. Все варианты распараллеливания показывают ускорение, при этом с увеличением количества процессов увеличивается ускорение программ, что говорит об обоснованности их применения для данного типа задач.
2. Для технологии OpenMP без параметра и с параметром Q максимальное ускорение было получено вариантом guided – 6,7777 и 6,2284 соответственно при 12 процессах. Это можно объяснить тем, что для данного типа задачи динамическое распределение итераций с последующим уменьшением размера порции оказалось наиболее эффективным. Для технологии MPI без параметра Q наибольшее ускорение показал кратный вариант – 1,5641 при 12 процессах, с параметром Q наибольшее ускорение показал некратный вариант – 5,8289 при 12 процессах. В целом, показатели ускорения между кратным и некратным вариантами MPI несущественно отличались друг от друга, поэтому однозначно определить лидера среди двух вариантов нельзя. Это можно объяснить тем, что количество итераций было практически одинаковым, а некратный вариант является векторной формой кратного. Стоит также отметить, что в программах с параметром Q, оба варианта MPI показали лучшие результаты, чем в программе без параметра. Это объясняется тем, что при использовании параметра Q, время выполнения операций коммуникации составляет более меньшую часть от общего времени работы программы.
3. Благодаря распараллеливанию, с увеличением количества процессов уменьшается время выполнения программ для обеих технологий с параметром Q и без параметра.
4. Использование параметра Q приводит к увеличению количества итераций, соответственно время работы обеих технологий увеличивается.

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, в виду того, что все варианты рассматриваемых технологий показали ускорение. Лучше всего себя показала технология OpenMP с использованием guided. Это можно объяснить постепенным уменьшением порции задач, выделяемой для каждого потока.

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 была реализация программ MPI. Интерес вызвала функция MPI\_ScatterV, позволяющая посылать каждому процессу различное количество элементов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Рыжков, Е. А. Параллельные заметки №1 – технология OpenMP / Е. А. Рыжков // Хабр : [сайт]. – Тула, 2010. – URL: https://habr.com/ru/companies/intel/articles/82486/ (дата обращения: 02.10.2023).
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 "mpi.h"

#include <stdlib.h>

#include "stdio.h"

/\*

//Multiple

#define Q 1

//#define Q 26

#define SIZE 6300000

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

{

int ProcRank, ProcNum;

float\* a = (float\*)malloc(SIZE \* sizeof(float));

float\* b = (float\*)malloc(SIZE \* sizeof(float));

float\* c = (float\*)malloc(SIZE \* sizeof(float));

double st\_time, end\_time;

MPI\_Init(&argc, &argv);

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

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

int i = 0;

if (ProcRank == 0) {

for (i = 0; i < SIZE; ++i) {

a[i] = 1.0;

b[i] = 2.0;

}

}

int count = SIZE / ProcNum;

float\* a\_ = (float\*)malloc(count \* sizeof(float));

float\* b\_ = (float\*)malloc(count \* sizeof(float));

float\* c\_ = (float\*)malloc(count \* sizeof(float));

double time = 0;

st\_time = MPI\_Wtime();

MPI\_Scatter(a, count, MPI\_FLOAT, a\_, count, MPI\_FLOAT, 0, MPI\_COMM\_WORLD);

MPI\_Scatter(b, count, MPI\_FLOAT, b\_, count, MPI\_FLOAT, 0, MPI\_COMM\_WORLD);

int j = 0;

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

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

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

}

}

MPI\_Gather(c\_, count, MPI\_FLOAT, c, count, MPI\_FLOAT, 0, MPI\_COMM\_WORLD);

MPI\_Barrier(MPI\_COMM\_WORLD);

end\_time = MPI\_Wtime();

time = end\_time - st\_time;

if (ProcRank == 0)

{

printf("\nMPI scatter and gather TIME OF WORK IS %f ", time);

printf("\nQ = %d", Q);

printf("\nProcesses: %d", ProcNum);

}

MPI\_Finalize();

free(a);

free(b);

free(c);

free(a\_);

free(b\_);

free(c\_);

return 0;

}\*/

//No multiple

#define Q 1

#define SIZE 6300017

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

{

int ProcRank, ProcNum;

float\* a = (float\*)malloc(SIZE \* sizeof(float));

float\* b = (float\*)malloc(SIZE \* sizeof(float));

float\* c = (float\*)malloc(SIZE \* sizeof(float));

double st\_time, end\_time;

MPI\_Init(&argc, &argv);

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

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

int i = 0;

if (ProcRank == 0) {

for (i = 0; i < SIZE; ++i) {

a[i] = 1.0;

b[i] = 2.0;

}

}

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

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

int equalPortionSize = (SIZE) / ProcNum;

if (SIZE % ProcNum != 0) {

equalPortionSize += 1;

}

for (i = 0; i < ProcNum - 1; ++i) {

sendcounts[i] = equalPortionSize;

}

int lastPortionSize = SIZE - equalPortionSize \* (ProcNum - 1);

int portionSize;

if (ProcRank == ProcNum - 1) {

portionSize = lastPortionSize;

}

else {

portionSize = equalPortionSize;

}

sendcounts[ProcNum - 1] = lastPortionSize;

displs[0] = 0;

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

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

}

float\* a\_ = (float\*)malloc(portionSize \* sizeof(float));

float\* b\_ = (float\*)malloc(portionSize \* sizeof(float));

float\* c\_ = (float\*)malloc(portionSize \* sizeof(float));

double time = 0;

st\_time = MPI\_Wtime();

MPI\_Scatterv(a, sendcounts, displs, MPI\_FLOAT, a\_, sendcounts[ProcRank], MPI\_FLOAT, 0, MPI\_COMM\_WORLD);

MPI\_Scatterv(b, sendcounts, displs, MPI\_FLOAT, b\_, sendcounts[ProcRank], MPI\_FLOAT, 0, MPI\_COMM\_WORLD);

int j = 0;

for (i = 0; i < portionSize; ++i) {

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

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

}

}

MPI\_Gatherv(c\_, sendcounts[ProcRank], MPI\_FLOAT, c, sendcounts, displs, MPI\_FLOAT, 0, MPI\_COMM\_WORLD);

MPI\_Barrier(MPI\_COMM\_WORLD);

end\_time = MPI\_Wtime();

time = end\_time - st\_time;

if (ProcRank == 0)

{

printf("\nMPI scatterV and gatherV TIME OF WORK IS %f ", time);

printf("\nQ = %d", Q);

printf("\nProcesses: %d", ProcNum);

}

MPI\_Finalize();

free(a);

free(b);

free(c);

free(a\_);

free(b\_);

free(c\_);

free(sendcounts);

free(displs);

return 0;

}

ПРИЛОЖЕНИЕ Б

Код программы с технологией OpenMP

#include <omp.h>

#include "stdio.h"

#include <stdlib.h>

#define THREADS 3

#define Q 1

#define CHUNK 100

#define SIZE 6300000

/\*

float\* omp\_guided(float\* vec1, float\* vec2) {

float\* sum = (float\*)malloc(SIZE \* sizeof(float));

int i, j, k;

for (k = 0; k < 12; ++k) {

#pragma omp parallel for shared(vec1, vec2, sum) private(i, j) schedule(guided, CHUNK)

for (i = 0; i < SIZE; ++i) {

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

sum[i] = vec1[i] + vec2[i];

}

sum[i] /= Q;

}

}

return sum;

}

float\* omp\_dynamic(float\* vec1, float\* vec2) {

float\* sum = (float\*)malloc(SIZE \* sizeof(float));

int i, j, k;

for (k = 0; k < 12; ++k) {

#pragma omp parallel for shared(vec1, vec2, sum) private(i, j) schedule(dynamic, CHUNK)

for (i = 0; i < SIZE; ++i) {

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

sum[i] = vec1[i] + vec2[i];

}

sum[i] /= Q;

}

}

return sum;

}

\*/

float\* omp\_static(float\* vec1, float\* vec2) {

float\* sum = (float\*)malloc(SIZE \* sizeof(float));

int i, j, k;

for (k = 0; k < 12; ++k) {

#pragma omp parallel for shared(vec1, vec2, sum) private(i, j) schedule(static, CHUNK)

for (i = 0; i < SIZE; ++i) {

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

sum[i] = vec1[i] + vec2[i];

}

sum[i] /= Q;

}

}

return sum;

}

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

omp\_set\_num\_threads(THREADS);

double st\_time, total\_time;

float\* a = (float\*)malloc(SIZE \* sizeof(float));

float\* b = (float\*)malloc(SIZE \* sizeof(float));

float\* sum\_vec;

int i;

for (i = 0; i < SIZE; ++i) {

a[i] = 1.0;

b[i] = 2.0;

}

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

sum\_vec = omp\_static(a, b);

//sum\_vec = omp\_guided(a, b);

//sum\_vec = omp\_dynamic(a, b);

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

printf("\nSTATIC TIME OF WORK IS: %f ", total\_time / 12);

printf("\nTHREADS: %d", THREADS);

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

free(sum\_vec);

free(a);

free(b);

return 0;

}

ПРИЛОЖЕНИЕ В

Код последовательной программы

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define SIZE 6300000

#define Q 1

/\*#define Q 26\*/

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

int count;

double total\_time = 0, start\_time, end\_time;

float\* a = (float\*)malloc(SIZE \* sizeof(float));

float\* b = (float\*)malloc(SIZE \* sizeof(float));

float\* c = (float\*)malloc(SIZE \* sizeof(float));

int i;

for (i = 0; i < SIZE; i++) {

a[i] = 1.0;

b[i] = 2.0;

}

for (count = 0; count < 12; count++) {

start\_time = clock();

int j = 0;

for (i = 0; i < SIZE; ++i) {

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

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

}

}

end\_time = clock();

total\_time += end\_time - start\_time;

}

printf("\nSEQ TIME OF WORK IS %f ", (total\_time / 12) / CLOCKS\_PER\_SEC);

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

free(a);

free(b);

free(c);

return 0;

}

ПРИЛОЖЕНИЕ Г

Код скрипта для запуска программ с MPI

#!/bin/bash

#SBATCH --job-name=Mashtakoff

#SBATCH --time=00:03:00

#SBATCH --nodes=1 --ntasks-per-node=3

#SBATCH --mem=1gb

export I\_MPI\_LIBRARY=/usr/lib64/slurm/mpi\_pmi2.so

srun --mpi=pmi2 ./v1

ПРИЛОЖЕНИЕ Д

Код скрипта для запуска программ с OpenMP

#!/bin/bash

#SBATCH --job-name=Mashtakoff

#SBATCH --time=00:03:00

#SBATCH --nodes=1 --ntasks-per-node=1

#SBATCH --mem=1gb

./g13