МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**

ПАРАЛЛЕЛЬНОЕ И РАСПРЕДЕЛЕННОЕ ПРОГРАММИРОВАНИЕ. WORK17

ОТЧЕТ О ПРАКТИКЕ

студента 3 курса 311 группы
направления 02.03.02 — Фундаментальная информатика и информационные технологии
факультета КНиИТ
Вильцева Данила Денисовича

Проверил
Старший преподаватель _____ М. С. Портенко

Саратов 2023

СОДЕРЖАНИЕ

1 Work 17

1.1 Условие задачи

Аналогично работе с OMP выполните следующее задание через MPI

Реализуйте параллельную версию бит-реверсирования. Оцените вклад в ускорение, который внесет такая реализация.

1.2 Решение. Параллельная версия

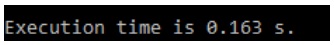
```
#include <iomanip>
#include <iostream>
#include <cmath>
#include <complex>
#include <time.h>
#include <mpi.h>
#include <algorithm>
using namespace std;
#define PI 3.14159265358979323846

int NProc, ProcId;

void PrintSignal(complex<double>* signal, int size) {
        cout << "Result signal" << endl;
        for (int i = 0; i < size; i++)
                cout << signal[i] << endl;
}

void DummyDataInitialization(complex<double>* mas, int size) {
        for (int i = 0; i < size; i++)
                mas[i] = 0;
        mas[size - size / 4] = 1;
}

void ProcessInitialization(complex<double>*& inputSignal, complex<double>*& outputSignal,
↪  int size) {
        inputSignal = new complex<double>[size];
        outputSignal = new complex<double>[size];
        DummyDataInitialization(inputSignal, size);
}

void ProcessTermination(complex<double>*& inputSignal, complex<double>*& outputSignal) {
        delete[] inputSignal;
        inputSignal = NULL;
        delete[] outputSignal;
        outputSignal = NULL;
}
```

```
void SerialBitReversing(complex<double>* inputSignal, complex<double>* outputSignal, int
↪  size) {
        int bitsCount = 0;
        for (int tmp_size = size; tmp_size > 1; tmp_size /= 2, bitsCount++);
        for (int ind = 0; ind < size; ind++) {
                int mask = 1 << (bitsCount - 1);
                int revInd = 0;
                for (int i = 0; i < bitsCount; i++) {
                        bool val = ind & mask;
                        revInd |= val << i;
                        mask = mask >> 1;
                }
                outputSignal[revInd] = inputSignal[ind];
        }
}

void ParallelBitReversing(complex<double>* inputSignal, complex<double>* outputSignal, int
↪  size) {
        int bitsCount = 0;
        for (int tmp_size = size; tmp_size > 1; tmp_size /= 2, bitsCount++);
        int DecBitsCount = bitsCount - 1;
        for (int ind = 0; ind < size; ind++) {
                int st = 1 << DecBitsCount;
                int revInd = 0;
                for (int i = ProcId; i < bitsCount; i += NProc) {
                        int mask = st >> i;
                        //revInd |= bool(ind & mask) << i;
                        bool val = ind & mask;
                        revInd |= val << i;
                }
                int revIndNew;
                MPI_Allreduce(&revInd, &revIndNew, 1, MPI_INT, MPI_BOR, MPI_COMM_WORLD);
                outputSignal[revIndNew] = inputSignal[ind];
        }
}

__inline void Butterfly(complex<double>* signal, complex<double> u, int offset, int
↪  butterflySize) {
        complex<double> tem = signal[offset + butterflySize] * u;
        signal[offset + butterflySize] = signal[offset] - tem;
        signal[offset] += tem;
}

void FFTCalculation(complex<double>* signal, int size) {
        int m = 0;
        for (int tmp_size = size; tmp_size > 1; tmp_size /= 2, m++);
        for (int p = 1; p <= m; p++) {
                int butterflyOffset = 1 << p;
```

```
                int butterflySize = butterflyOffset >> 1;
                double coeff = PI / butterflySize;
                for (int i = 0; i < size / butterflyOffset; i++)
                        for (int j = 0; j < butterflySize; j++)
                                Butterfly(signal, complex<double>(cos(-j * coeff),
                                        sin(-j * coeff)), j + i * butterflyOffset,
↪  butterflySize);
        }
}

void SerialBitReversingFFT(complex<double>* inputSignal, complex<double>* outputSignal, int
↪  size) {
        SerialBitReversing(inputSignal, outputSignal, size);
        FFTCalculation(outputSignal, size);
}

void ParallelBitReversingFFT(complex<double>* inputSignal, complex<double>* outputSignal,
↪  int size) {
        ParallelBitReversing(inputSignal, outputSignal, size);
        FFTCalculation(outputSignal, size);
}

void TestResult(complex<double>* inputSignal, complex<double>* outputSignal, int size) {
        complex<double>* SerialBitReversingSignal;
        double Accuracy = 1.e-6;
        bool equal = true;
        int i;
        SerialBitReversingSignal = new complex<double>[size];
        SerialBitReversingFFT(inputSignal, SerialBitReversingSignal, size);
        for (i = 0; i < size; i++) {
                if (abs(outputSignal[i] - SerialBitReversingSignal[i]) >= Accuracy)
                        equal = false;
        }
        if (!equal) printf("The results of serial and parallel algorithms are NOT
↪  identical.\n");
        else printf("The results of serial and parallel algorithms are identical.\n");
        delete[] SerialBitReversingSignal;
}

int main() {
        MPI_Init(NULL, NULL);
        MPI_Comm_size(MPI_COMM_WORLD, &NProc);
        MPI_Comm_rank(MPI_COMM_WORLD, &ProcId);
        complex<double>* inputSignal = NULL;
        complex<double>* outputSignal = NULL;
        int size = 524288;
        const int repeatCount = 10;
        double startTime, finishTime;
```

```
	double duration;
	double minDuration = DBL_MAX;
	ProcessInitialization(inputSignal, outputSignal, size);
	for (int i = 0; i < repeatCount; i++) {
		if (ProcId == 0) {
			startTime = clock();
		}
		ParallelBitReversingFFT(inputSignal, outputSignal, size);
		if (ProcId == 0) {
			finishTime = clock();
			duration = (finishTime - startTime) / CLOCKS_PER_SEC;
			if (duration < minDuration)
				minDuration = duration;
		}
	}
	if (ProcId == 0) {
		cout << setprecision(6);
		cout << "Execution time is " << minDuration << " s." << endl;
		TestResult(inputSignal, outputSignal, size);
		//PrintSignal(outputSignal, size);
	}
	ProcessTermination(inputSignal, outputSignal);
	MPI_Finalize();
	return 0;
}
```

2 Результат работы



Мы получаем при значении входного сигнала 524288 время работы 0.163, что означает ускорение в 1.135 раза

Распараллеливание процедуры ParellelFFTCalculation оказывается эффективнее.

3 Характеристики компьютера

Характеристики устройства

Имя устройства	DESKTOP-MSS8D39
Процессор	Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz 3.20 GHz
Оперативная память	8,00 ГБ
Код устройства	E3BB953D-13B0-42A7-944B-1ED9FD0E C328
Код продукта	00330-80000-00000-AA153
Тип системы	64-разрядная операционная система, процессор x64