МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

имени М.В. Ломоносова

ФАКУЛЬТЕТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И КИБЕРНЕТИКИ

ПРОГРАММА «РАЗРАБОТЧИК ПРОФЕССИОНАЛЬНО-ОРИЕНТИРОВАННЫХ КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ»

ПРОЕКТНОЕ ЗАДАНИЕ ПО КУРСУ «ПАРАЛЛЕЛЬНОЕ ПРОГРАММИРОВАНИЕ»:

ВЫЧИСЛЕНИЕ СУММЫ РЯДА

работу выполнил:

МОСКОВСКИЙ ВЛАДИСЛАВ ВАСИЛЬЕВИЧ

МОСКВА 2021

Постановка задачи

1. Требуется написать последовательную версию программы вычисления суммы ряда на языке Си.

Сумма ряда: $$\sum_{n=1}^{10} \frac{n^3+2}{n^5+\sin 2^n}$$

2. Произвести анализ времени её выполнения.
3. Выполнить расчёты с различными значениями N.
4. Полученные результаты занести в таблицу
5. Написать параллельную программу с использованием базовых функций MPI. Результаты замеров времени работы занести в таблицу.

Последовательная версия программы

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>


static inline double f(unsigned long long int index) {

	return pow(index, 3) / (pow(index, 5) + sin(pow(2, index)));
}

double sum(unsigned long long int N) {
	double SUM = 0;
#pragma loop count min(16)
	for (unsigned long long int n = 1; n <= N; ++n) {
		SUM += f(n);
	}
	return SUM;
}

int main(int argc, char* argv[]) {
	unsigned long long int N = argc > 1 ? atol(argv[1]) : 1000;

	printf("sum %lld = ", N);
	clock_t time = clock();
	double SUM;
	SUM = sum(N);
	printf("%lf\n", SUM);
	time = clock() - time;
	printf("evaluation time: %lf\n", (double)time / CLOCKS_PER_SEC);
	return 0;
}
```

Параллельная реализация

```
// Параллельная версия программы

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <mpi.h>
#include <iostream>

static inline double f(unsigned long long int index) {

    return pow(index,3)/(pow(index,5)+sin(pow(2,index)));
}

// функция расчета частичных сумм

long double sum(unsigned long long int start, unsigned long long int end) {
    double s = 0;
    for (unsigned long long int i = start; i < end; ++i) {
        s += f(i);
    }
    return s;
}

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

    setlocale(LC_ALL, "Russian");


    unsigned long long int N; // размерность ряда
    long double s; // частичная сумма ряда
    int comm_size, comm_rank; // количество процессоров и номер процессора
    double global_start_time, global_end_time, global_work_time; // время работы




    N = argc > 1 ? atoll(argv[1]) : 1000; // если параметр N нельзя получить из
коммандной строки, используем значение по умолчанию

    MPI_Init(&argc, &argv); // распараллеливание начинается отсюда


    MPI_Comm_rank(MPI_COMM_WORLD, &comm_rank); // каждый процесс узнает о количестве
процессов всего
    MPI_Comm_size(MPI_COMM_WORLD, &comm_size); // каждый процесс узнаёт о собственном
номере
    std::cout << "Общее число процессов: " << comm_size << std::endl;
    std::cout << "Номер процесса : " << comm_rank << std::endl;
    if (comm_rank == 0) global_start_time = MPI_Wtime(); // главный процесс запоминает
время начала расчёта

    // разбиваем на подзадачи

    unsigned long long int partials = N / comm_size;
    unsigned long long int m = N % comm_size;
    unsigned long long int start = comm_rank * partials + 1;
    unsigned long long int end = (comm_rank + 1) * partials + 1;
    if (m > 0) {
        if (comm_rank < m) {
            start += comm_rank;
            end += comm_rank + 1;
        }
        else {
```

```
            start += m;
            end += m;
        }
    }

    double start_time, end_time; // локальные времена для времени расчёта частичных сумм
    start_time = MPI_Wtime(); // запоминаем время начала расчёта
    s = sum(start, end); // расчёт частичной суммы
    end_time = MPI_Wtime(); // запоминаем время конца расчёта
    double work_time = end_time - start_time; // получаем время конца расчёта
    MPI_Barrier(MPI_COMM_WORLD); // ждём все процессы

    if (comm_rank != 0) { // если процесс не главный
        MPI_Send(&s, 1, MPI_DOUBLE, 0, comm_size * comm_rank, MPI_COMM_WORLD); //
посылаем главному процессу результат
    }

    double global_sum = 0;
    if (comm_rank == 0) { // если процесс главный
        double partial_sum;
        global_sum = s; // глобальная сумма пока это частичная сумма, посчитанная главным
процессом
        MPI_Status status;
        for (int i = 1; i < comm_size; ++i) { // принимаем все частичные суммы от других
процессов
            MPI_Recv(&partial_sum, 1, MPI_DOUBLE, i, comm_size * i, MPI_COMM_WORLD,
&status);
            global_sum += partial_sum; // и доавляем к глобальной сумме
        }

        std::cout << "Сумма ряда " << global_sum << std::endl;

        global_end_time = MPI_Wtime(); // глобальное время конца расчёта
        global_work_time = global_end_time - global_start_time;
    }

    fprintf(stderr, "proc %d of %d (%lf sec):\tsum from %lld to %lld:\t%1.8lf\n",
comm_rank, comm_size, work_time, start, end - 1, s); // вывод своего результата каждым
процессом

    MPI_Barrier(MPI_COMM_WORLD); // ждём все процессы
    if (comm_rank == 0) {
        fprintf(stderr, "global work time: %lf s; global sum of %lld elements = %lf\n",
global_work_time, N, global_sum);
    }
    MPI_Finalize(); // конец параллелизации
    return 0; // конец программы
}
```

Сравнение времени выполнения

Последовательная программа

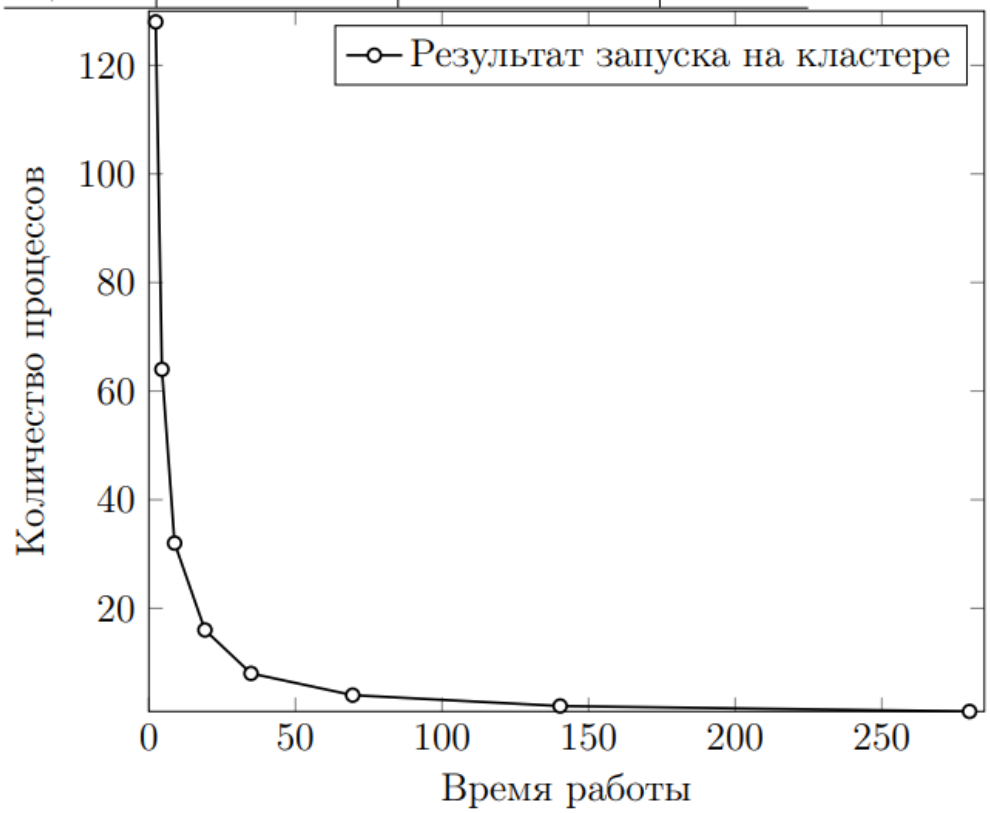
Компилятор	Ключи	N	Время работы, сек	Результат
gcc	нет	1400000000	143.140000	0.056538
gcc	нет	2400000000	245.260000	0.056538
gcc	нет	3400000000	347.570000	0.056538
gcc	-O0	1400000000	143.110000	0.056538
gcc	-O0	2400000000	245.270000	0.056538
gcc	-O0	3400000000	347.650000	0.056538
gcc	-O1	1400000000	126.310000	0.056538
gcc	-O1	2400000000	216.570000	0.056538
gcc	-O1	3400000000	306.820000	0.056538
gcc	-O2	1400000000	126.040000	0.056538
gcc	-O2	2400000000	216.200000	0.056538
gcc	-O2	3400000000	306.180000	0.056538
gcc	-O3	1400000000	126.220000	0.056538
gcc	-O3	2400000000	217.200000	0.056538
gcc	-O3	3400000000	308.320000	0.056538
icc	нет	1400000000	35.290000	0.056538
icc	нет	2400000000	60.450000	0.056538
icc	нет	3400000000	85.650000	0.056538
icc	-O0	1400000000	39.920000	0.056538
icc	-O0	2400000000	68.270000	0.056538
icc	-O0	3400000000	96.500000	0.056538
icc	-O1	1400000000	31.820000	0.056538
icc	-O1	2400000000	54.500000	0.056538
icc	-O1	3400000000	77.230000	0.056538
icc	-O2	1400000000	35.290000	0.056538
icc	-O2	2400000000	60.800000	0.056538
icc	-O2	3400000000	85.690000	0.056538
icc	-O3	1400000000	35.440000	0.056538
icc	-O3	2400000000	60.490000	0.056538
icc	-O3	3400000000	85.650000	0.056538
icc	-fast	1400000000	34.710000	icc
-fast	2400000000	59.490000	0.056538	
icc	-fast	3400000000	84.300000	0.056538

Тут мы видим, что даже в самом быстром случае и с меньшим количеством элементов программа, скомпилированная компилятором GCC всё равно медленнее самой медленной программы с большим количеством элементов, скомпилированной компилятором ICC. Также видно, что GCC по умолчанию не оптимизирует программу, а ICC как минимум выставляет флаг -O2. В моём случае самой быстрой оказалась программа, собранная ICC с флагом -O1, выдающим минимальный размер исполняемого файла. При этом использование ключа -fast не дало существенного ускорения.

Параллельная программа MPI

Запуск программы осуществлялся на AWS ParallelCluster, потому что я не смог распараллелить программу на локальной машине.

Элементы	Узлы	Процессы	Время работы, сек	Результат
$3,4 \cdot 10^9$	1	1	279.954479	0.056538
$3,4 \cdot 10^9$	1	2	140.301850	0.056538
$3,4 \cdot 10^9$	1	4	69.530332	0.056538
$3,4 \cdot 10^9$	1	8	34.855666	0.056538
$3,4 \cdot 10^9$	2	16	19.137289	0.056538
$3,4 \cdot 10^9$	4	32	8.778247	0.056538
$3,4 \cdot 10^9$	8	64	4.460338	0.056538
$3,4 \cdot 10^9$	16	128	2.347080	0.056538



Из данной лабораторной работы я сделал вывод, что компиляция программы под конкретное железо, на котором она будет работать, играет существенную роль в скорости её выполнения. В данном случае, код, сгенерированный компилятором производителя моего процессора(intel) выполнялся на порядок быстрее кода, сгенерированного GCC для всех процессоров. Хотя дальнейшие оптимизации «под конкретную систему» уже не даёт существенного прироста скорости выполнения. Также, увидел, что параллелизация выполнения независимого кода позволяет существенно ускорить выполнение программы и, при этом, более-менее равномерно загрузить все системные вычислительные ресурсы, большая часть которых бы простаивала в случае выполнения последовательной программы. Ускорение = 119