**Министерство науки и высшего образования Российской Федерации**

**федеральное государственное автономное образовательное учреждение высшего образования**

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»**

**(Университет ИТМО)**

**Факультет программной инженерии и компьютерной техники**

Отчет по лабораторной работе №3

по дисциплине «Параллельное программирование»

***Тема* «Распараллеливание циклов с помощью технологии OpenMP»**

Выполнили:

студенты гр. № P33212

Зиганшин Геннадий Муратович

Ян Цзяфэн

Проверил:

Балакшин Павел Валерьевич

Санкт-Петербург

2021

Содержание

[Краткая характеристика 3](#_Toc71668661)

[Описание задачи 3](#_Toc71668662)

[Коды программы 4](#_Toc71668663)

[Графики параллельного ускорения 4](#_Toc71668664)

[График загрузки процессора от времени 6](#_Toc71668665)

[График параллельного ускорения при N < N1 7](#_Toc71668666)

[Вывод 7](#_Toc71668667)

# Краткая характеристика

* Процессор: AMD Ryzen 5 3600 6-Core Processor
* Операционная система: 20.04.1-Ubuntu
* Компилятор GCC: gcc 9.3.0
* ОЗУ 32 гб

# Описание задачи

1. Добавить во все for-циклы в программе из ЛР №1 следующую директиву OpenMP: ”#pragma omp parallel for default(none) private(...) shared(...)”. Наличие всех перечисленных параметров в указанной директиве является обязательным.
2. Проверить все for-циклы на внутренние зависимости по данным между итерациями. Если зависимости обнаружились, использовать для защиты критических секций директиву ”#pragma omp critical” или ”#pragma omp atomic” (если операция атомарна), или параметр reduction (предпочтительнее) или вообще отказаться от распараллеливания цикла (свой выбор необходимо обосновать).
3. Убедиться, что получившаяся программа обладает свойством прямой совместимости с компиляторами, не поддерживающими OpenMP (для проверки этого можно скомпилировать программу без опции ”–fopenmp”, в результате не должно быть сообщений об ошибках, а программа должна корректно работать).
4. Провести эксперименты, замеряя параллельное ускорение. Привести сравнение графиков параллельного ускорения с ЛР №1 и ЛР №2.
5. Провести эксперименты, добавив параметр ”schedule” и варьируя в экспериментах тип расписания. Исследование нужно провести для всех возможных расписаний: static, dynamic, guided. Способ варьирования параметра chunk\_size выбрать самостоятельно (но должно быть не менее 5 точек варьирования). Привести сравнение параллельного ускорения при различных расписаниях с результатами п.4.
6. Выбрать из рассмотренных в п.4 и п.5 наилучший вариант при различных N. Сформулировать условия, при которых наилучшие результаты получились бы при использовании других типов расписания.
7. Найти вычислительную сложность алгоритма до и после распараллеливания, сравнить полученные результаты.
8. Необязательное задание №1 (для получения оценки «четыре» и «пять»). Для иллюстрации того, что программа действительно распараллелилась, привести график загрузки процессора (ядер) от времени при выполнении программы при N = N1 для лучшего варианта распараллеливания. Для получения графика можно как написать скрипт так и просто сделать скриншот диспетчера задач, указав на скриншоте моменты начала и окончания эксперимента (в отчёте нужно привести текст скрипта или название использованного диспетчера). Недостаточно привести однократное моментальное измерение загрузки утилитой htop, т.к. требуется привести график изменения загрузки за всё время выполнения программы.
9. Необязательное задание №2 (для получения оценки «пять»). Построить график параллельного ускорения для точек N < N1 и найти значения N, при которых накладные расходы на распараллеливание превышают выигрыш от распараллеливания (независимо для различных типов расписания).

# Коды программы

lab3.c, lab3\_schedule\_0.c, lab3\_schedule\_1.c, lab3\_schedule\_2.c, lab3\_schedule\_3.c, PPscript.txt в виде отдельных файлов.

# Графики параллельного ускорения

*Рис. 1 – График параллельного ускорения в лабе 3*

Влияние распараллеливания циклов на эффективность работы программы, заметно только при определенных N. Т.к. при малых N большие накладные расходы на обеспечение работы потоков, а при больших N основное время работы программы занимает сортировка, которая не параллелится.

*Рис. 2 – График параллельного ускорения в лабе 1*

*Рис. 3 – График параллельного ускорения в лабе 2*

# График загрузки процессора от времени

Для большей наглядности работы программы был выбран очень большой N. Благодаря этому на графиках четко видны этапы работы программы. Первый: Выделение памяти, когда максимально нагружено только одно ядро. Второй: вычисления, при которых нагрузка идет на все доступные ядра. Третий: сортировка, которая нагружает только одно ядро.



*Рис. 4 – График загрузки процессора от времени*

# График параллельного ускорения при N < N1

*Рис. 5 – График параллельного ускорения при N < N1*

Накладные расходы, на обеспечение работы потоков, при малых N достаточно велики, чтобы ухудшить время работы программы, при распараллеливании.

# Вывод

* Распараллеливаемая часть программы занимает малую долю от всей ее работы.
* Использование многопоточности имеет смысл, только при возможности распараллелить достаточно большую долю программы.