**д**

**Отчет о выполнении практического задания по предмету**

**«Прикладное программирование в задачах науки и техники»**

**на тему**

**«Разработка параллельной программы с использованием технологии OpenACC»**

Выполнил:

Попов Алексей Павлович

Аспирант первого года

ИПМ РАН

Москва, 2023

**Оглавление**

[Постановка задачи 3](#_Toc152286938)

[Описание программы 4](#_Toc152286939)

[Базовый алгоритм 4](#_Toc152286940)

[Модифицированный алгоритм 4](#_Toc152286941)

[Экспериментальная оценка реализованных версий программы 4](#_Toc152286942)

[Сравнение различных размеров данных и типов ускорителей 4](#_Toc152286943)

[Приложение 6](#_Toc152286944)

# **Постановка задачи**

1. Требуется разработать версию параллельного исполнения предложенной программы с использованием технологии *OpenACC*
2. Провести сравнение скорости работы предложенных версий программы на различных объемах данных с различными типами ускорителя, используемых при исполнении
3. Выбрать оптимальные параметры для исполнения программы и определить основные причины недостаточной масштабируемости программы

# **Описание программы**

## **Базовый алгоритм**

В качестве базового алгоритма мне был предложен алгоритм с регулярной зависимостью по данным, на вход которому подавалась трехмерный тензор, над которым выполнялся итеративный процесс (со всесторонней зависимостью по данным).

Он состоит из трех базовых функций: *init, relax, verify.* В функции *init* происходит инициализация массива, над которым будет производиться основная вычислительная работа, в функции *relax* происходит выполнение основной вычислительной работы, при этом функция запускается в цикле до сходимости задачи. Функция *verify* производит вычисление контрольной суммы по массиву данных, после завершения процесса итерации.

В качестве подготовки к реализации параллельной версии программы был также изменен порядок обхода массивов для обеспечения наиболее эффективной работы.

Код базового алгоритма можно посмотреть в Приложении 1.

## **Модифицированный алгоритм**

В процессе анализа программного кода базового алгоритма было замечено, что в основной вычислительной функции *relax* (она занимает наибольшую по времени часть исполнения алгоритма) есть регулярная зависимость по данным. Для того чтобы сделать работу программы более подходящей для параллелизации была проведена работа по переходу на циклический обход диагональных гиперплоскостей куба. Таким образом удалось избавиться от зависимостей внутри гиперплоскостей, что позволяет параллельно выполнять работу внутри каждой гиперплоскости.

При реализации параллельной версии функции *relax* был использован базовый подход, в котором выбранные гиперплоскости оставались неизменно лежать в памяти, что негативно влияет на скорость работы такой программы GPU.

Код модифицированного алгоритма можно посмотреть в Приложении 2.

# **Экспериментальная оценка реализованных версий программы**

## **Сравнение различных размеров данных и типов ускорителей**

В качестве базового уровня оптимизации был выбран уровень O3. Далее были проведены эксперименты с подсчетом на разном размере входных данных и с использованием разных видов ускорителей, дополнительно исследовалась версия программы без распараллеливания. Для этого был реализован скрипт на языке *Python* (Приложение 3), который позволил автоматизировать все запуски программы на суперкомпьютере *MIC.* Важно отметить, что во всех экспериментах используется усредненное время исполнения программы за 5 запусков. Результаты запусков программы можно посмотреть на Графике 1. График построен при помощи языка Python и библиотеки Matplotlib, ноутбук с кодом для построения графиков можно найти в Приложении 4.

A graph of a graph

Description automatically generated with medium confidence A screenshot of a computer

Description automatically generated

График 1. Сравнение скорости работы программы (*var7\_2*)  с использованием различных ускорителей, на разных размерах входных данных.

По результатам сравнения скорости работы программы можно сделать следующие выводы: ускорение при использовании GPU не очень получается, так как данные неэффективно расположены в памяти. При этом использование ускорителя позволяет ускорить работу алгоритма, по сравнению с базовой последовательной версией, даже использование GPU с таким распределением по памяти. Важно также отметить, что при небольших размерах входных данных использование GPU позволяет обработать данные быстрее, чем обработка на CPU, но с увеличением размера входных данных GPU начинает проигрывать. Для данной задачи для GPU наилучшим образом подходит размер данных до 66, дальше он начинает драматические проигрывать CPU.

# **Приложение**

Приложение 1

Файл ***var7.c*** с кодом программы приложен вместе с отчетом.

Приложение 2

Файл ***var7\_2.c*** с кодом программы приложен вместе с отчетом.

Приложение 3

Файл ***start\_diff\_opt\_data.py*** с кодом скриптов запуска программы на разных объемах данных, а также на разных видах ускорителей кластере *MIC* приложен вместе с отчетом.

Приложение 4

Файл ***padder.ipynb*** с кодом для отрисовки графиков, по файлам сгенерированным скриптами из Приложения 3, приложен вместе с отчетом.

Приложение 5

Файл ***diff\_opt\_data.json*** с результатами замеров полученный при помощи скрипта из Приложения 3.