Отчет о выполнении практического задания по предмету

«Прикладное программирование в задачах науки и техники»

на тему

«Разработка параллельной программы с использованием технологии OpenMP»

Выполнил:

Цаплин Никита Александрович

Аспирант первого года

ИПМ РАН

Москва, 2023

Оглавление

Постановка задачи

1) Требуется разработать две версии параллельного исполнения предложенной программы с использованием технологии OpenMP
2) Исследовать правильность реализованной параллельной версии программы при помощи средств анализа параллельных программ пакета InspXe
3) Провести сравнительный анализ реализованных версий программ с использованием различных компиляторов (gcc, icc, pgcc, clang) и различных уровней оптимизации (O0, 01, 02, 03)
4) Провести сравнение скорости работы предложенных версий программы на различных объемах данных с различным числом нитей, используемых при исполнении, с использованием различных компиляторов (gcc, icc, pgcc, clang)
5) Выбрать оптимальные параметры для исполнения программы и определить основные причины недостаточной масштабируемости программы

Использованные компиляторы

Компиляторы pgcc и clang не удалось суметь использовать для компиляции, поэтому анализ производился при сравнении двух компиляторов, gcc и icc.

1. gcc (GCC) 10.2.0 Copyright (C) 2020 Free Software Foundation, Inc.
2. icc (ICC) 17.0.0 20160721 Copyright (C) 1985-2016 Intel Corporation. All rights reserved.

Описание программы

Базовый алгоритм

В качестве базового алгоритма мне был предложен алгоритм итеративного процесса рассеяния тепла для трехмерной сетки, в котором во время расчета используются две копии состояния сетки — старой итерации и расчитываемой. Благодаря этому основной расчетный цикл хорошо параллелизуется, так как зависимости чтения-запись разведены на два разных этапа.

Он состоит из двух базовых функций: *init_array, kernel_heat_3d.* В функции *init_array* происходит инициализация массива, над которым будет производиться основная вычислительная работа.

Все вычисления производятся внутри *kernel_heat_3d.*

Программа также была с заголовочным файлом, в котором предопределены размеры датасетов.

Общие модификации алгоритма

Алгоритм почти не модифицировался. Исходные предопредленные датасеты были изменены, в частности потому что там менялась и размерность матрицы, и количество итераций — для более простого анализа графиков от размерности, было зафиксировано число итераций.

Во всех случаях параллелизация производилась только основной вычислительной функции, *kernel_heat_3d* как вносящей основной вклад во время работы программы.

Также, для замеров времени везде использовалась *omp_get_wtime().*

Версия программы с omp for

Была произведена параллелизация основных циклов с помощью прагмы:

#pragma omp for schedule(static)

Распределение было выбрано статичным, так как нагрузки на всех итерациях цикла по строкам матриц примерно равные.

Директива *collapse(N)* не использовалась, так как это приводило к крашам программ скомпилированных с icc с оптимизацией -O1 и ниже.

Инициализация параллельной секции *omp parallel* происходит на каждой новой итерации алгоритма, что не должно привести к большим накладным расходам.

Версия программы с omp task

Использовалась директива *omp taskloop,* по сути 1-в-1 эквивалентная *omp for*, с точки зрения расположения в коде и использования.

Генерация параллельной области производится один раз, на весь расчетный цикл, с помощью *omp parallel* + *omp single*, в этом одно из принциальных отличий между *for* и *task* версиями.

Экспериментальная оценка реализованных версий программы

Для автоматизации компиляции, запуска программ с разными параметрами, сбора статистики, построения графиков и прочих задач, были использованы скрипты на языке Python, основанные на скрипте, разработанном одногруппником Алексеем Поповым. Также были дописаны некоторые скрипты на Bash, для упрощения перекомпиляции icc версий. Всё это идет приложением к отчету.

Все полученные цифры – результат усреднения нескольких запусков. Конкретные значения также приложены к отчету отдельными файлами.

Проверка корректности написанной программы

Для проверки правильности реализованных версий программы была проведена их валидация при помощи компилятора *icc* и инспектора параллельных программ от компании *Intel – InspXe*.

Примеры использованных команд и ключей:

inspxe-cl -collect=ti3 и *inspxe-cl -collect=mi3*

В результате проверки были выявлены подозрительные места, которые были дополнительно проверены и обработаны.

Сравнение различных компиляторов и уровней оптимизации

В качестве размера стороны матриц взято значение 120, второе по размеру значение среди исследованных параметров.

Результаты приложены на Графике 1 и Графике 2 соответственно.

Какие выводы можно сделать на основе полученных данных:

1. icc во всех случаях производит программы работающие быстрее, чем полученные через gcc;
2. В данной поставленной задаче, что task и for варианты почти что эквиваленты с точки зрения параллелизма. В случае gcc так и получается по замерам, однако

icc гораздо лучше обрабатывает for директивы, чем task. Возможно это связано с тем, что эта версия icc плохо поддерживает task модель, и это нивелирует оптимизированное использование «родного» железа.

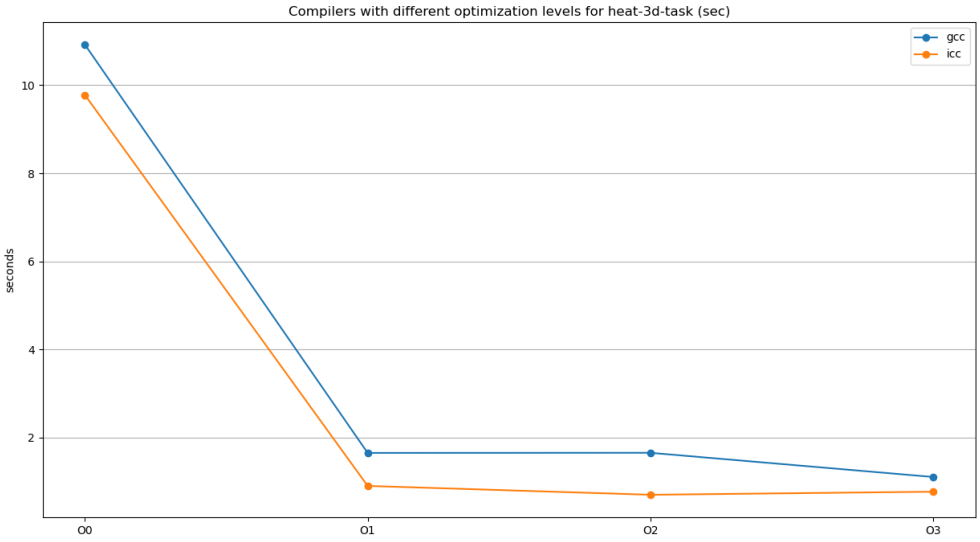


График 1: Сравнение уровней оптимизаций для heat-3d-task

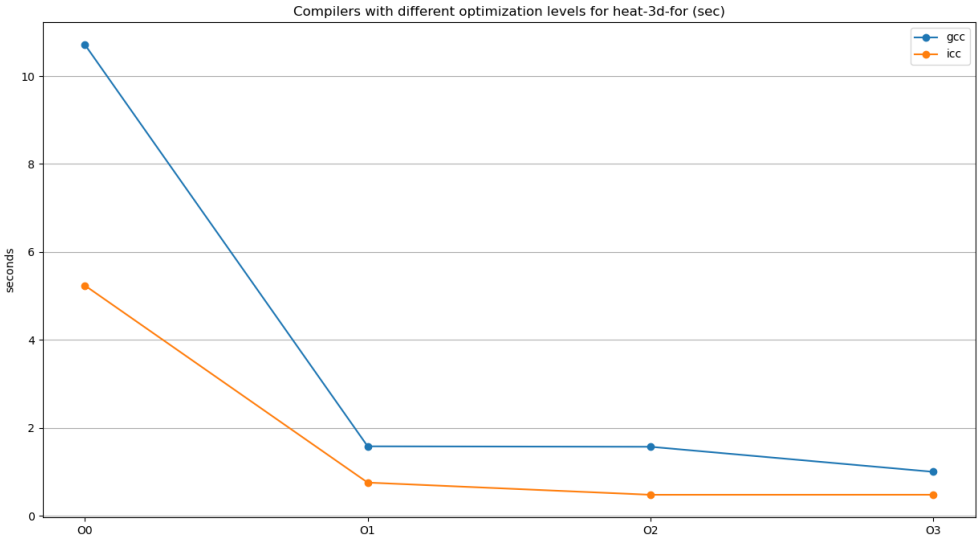


График 2: Сравнение уровней оптимизаций для heat-3d-for

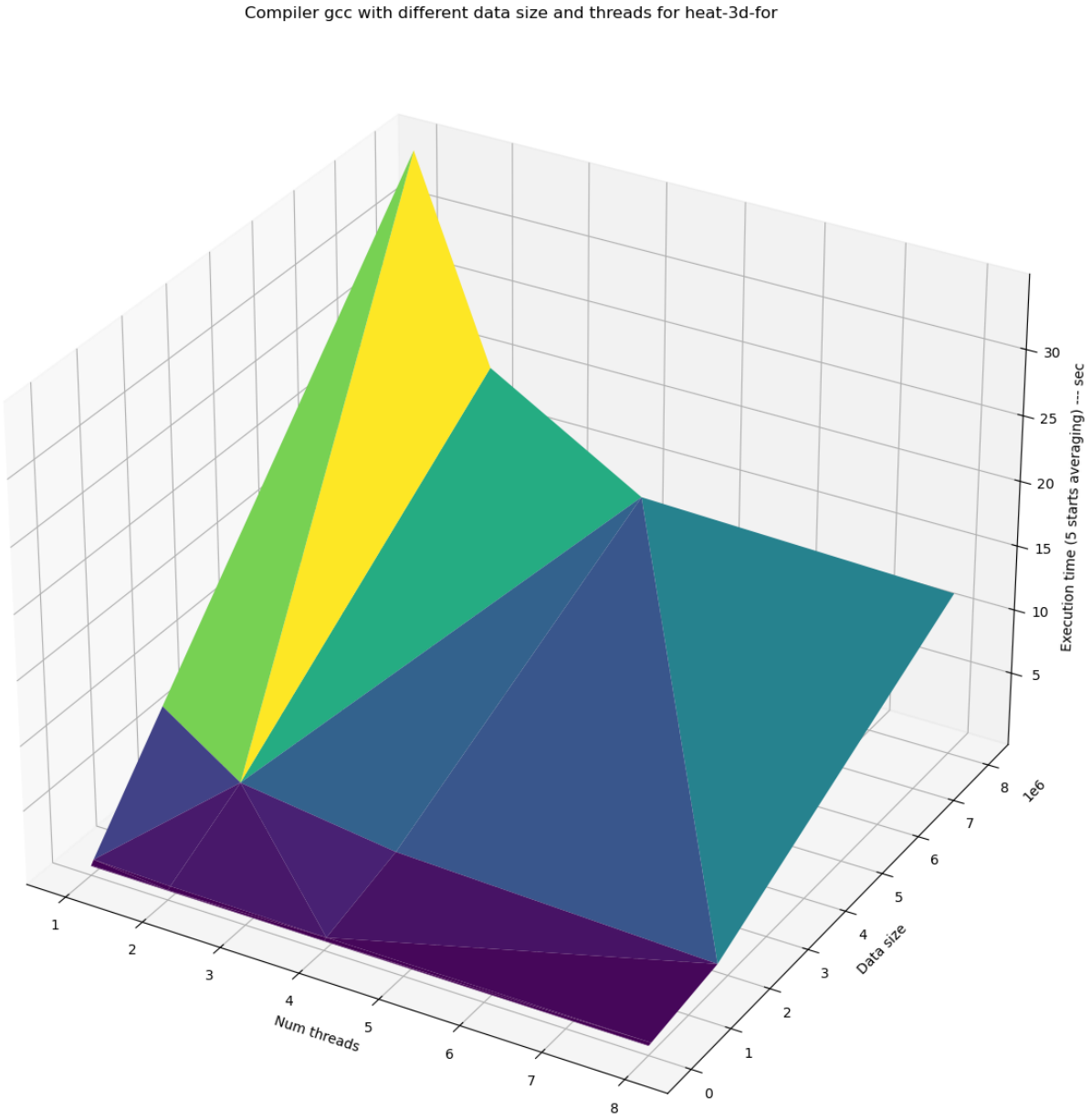
Сравнение различных компиляторов, размеров данных и количестве потоков

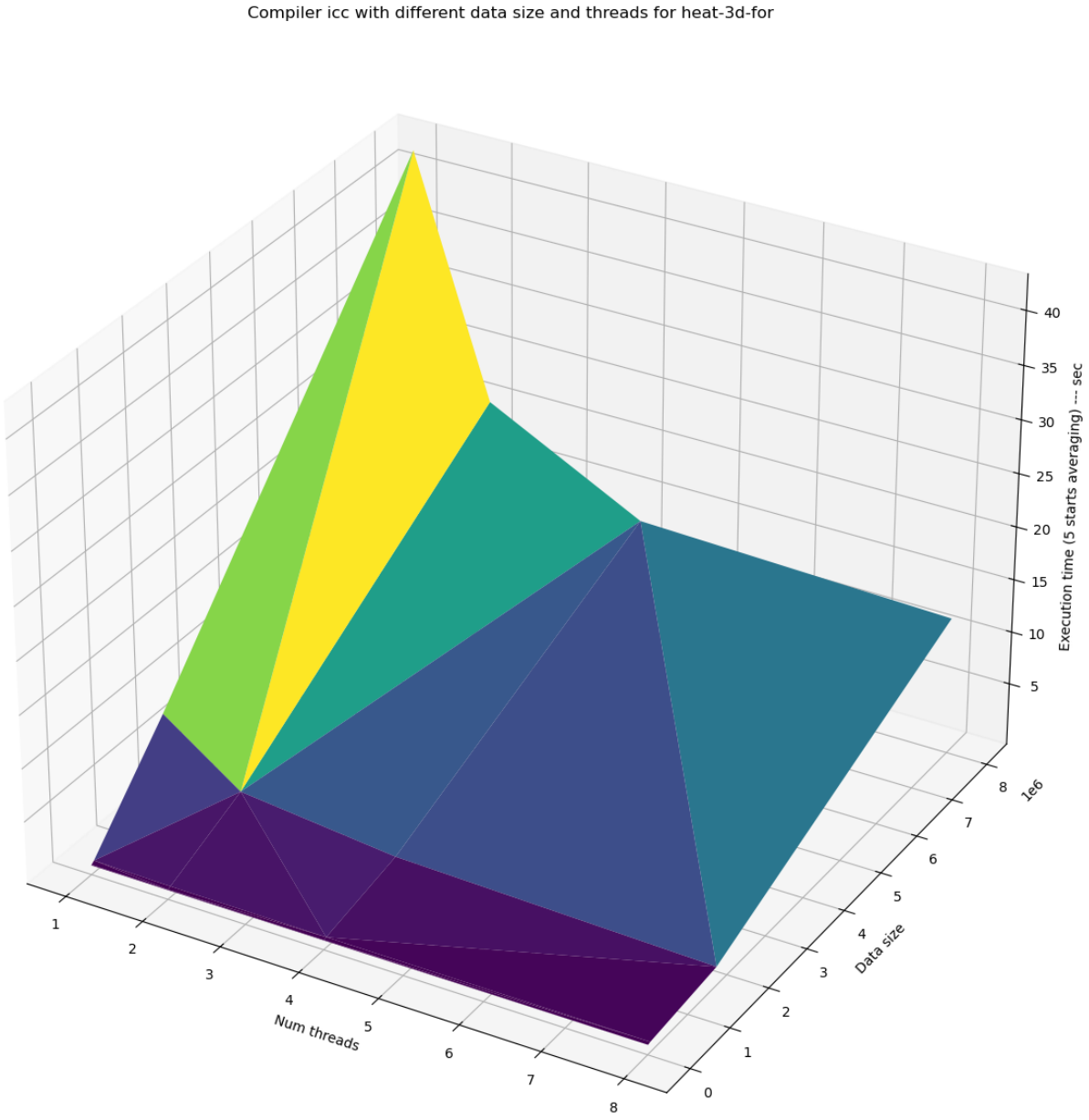
В качестве уровня оптимизации был выбран O3, как самый быстрый и при этом сохраняющий корректность работы программ.

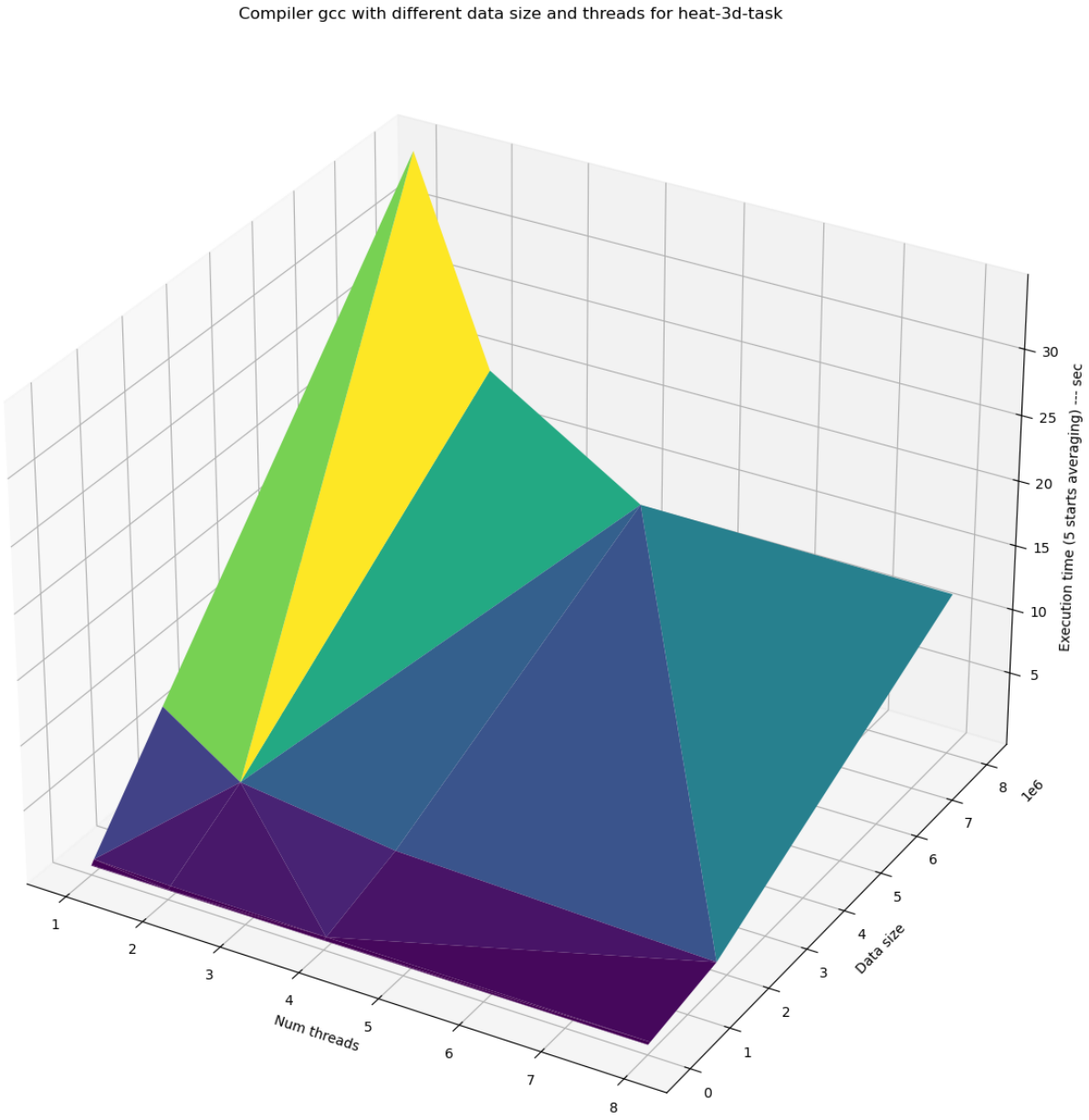
Результаты в виде 3D графиков можно увидеть далее.

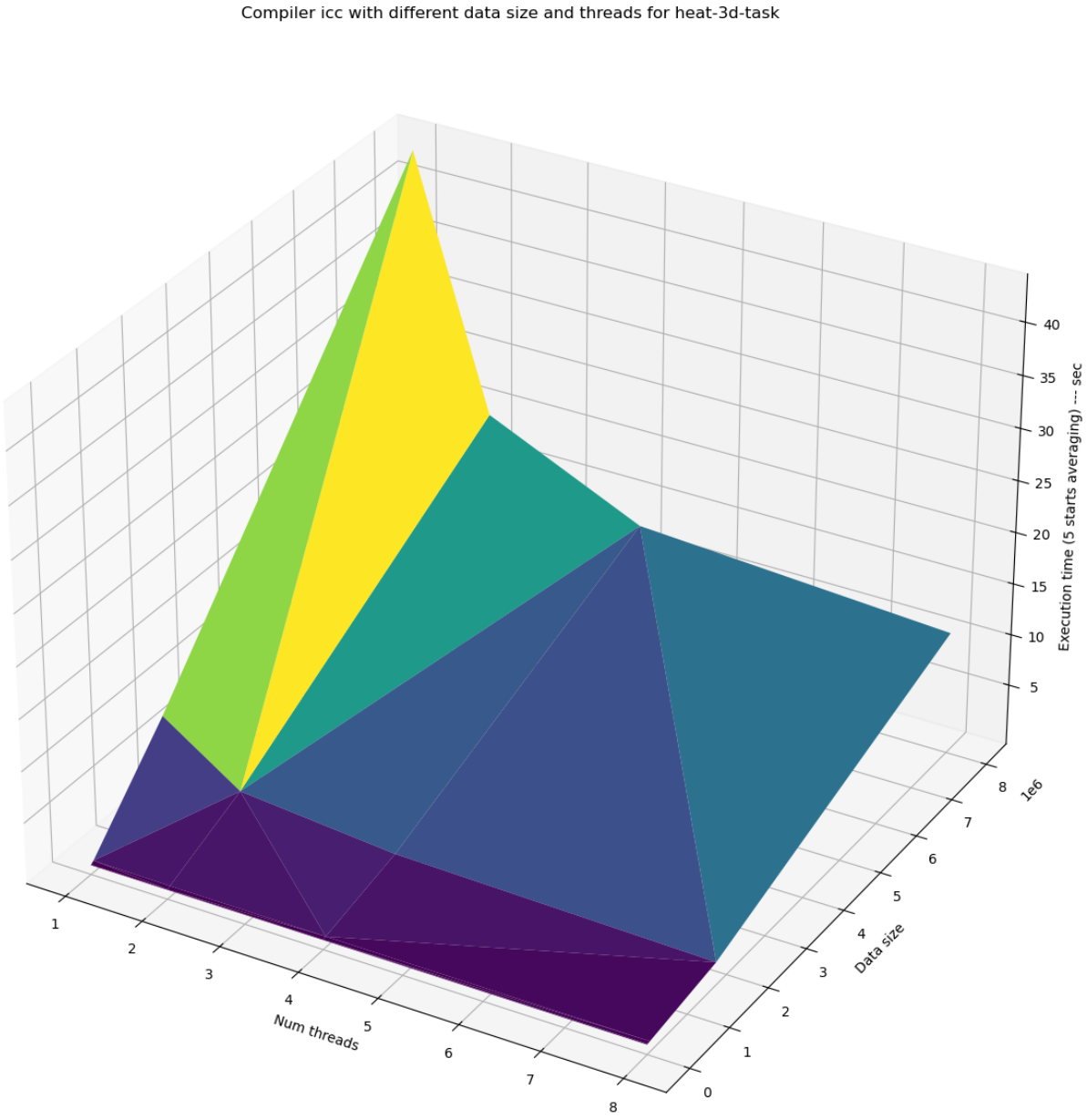
Выводы, которые можно сделать на основе полученных данных:

1. Для данной задачи task и for вариант почти что эквиваленты, с точки зрения производительности.
2. *icc* и *gcc* показывают примерно похожую производительность при большом числе потоков, однако *icc* работает медленней при меньшем числе потоков;
3. Программа в текущем исполнении (и task, и for) масштабируется при малых значениях размера матрицы почти что линейно с числом нитей, что замечательный результат;
4. Однако на самом большом датасете, при N=200, масштабируется не слишком хорошо — при росте числа потоков больше чем 4, время практически не уменьшается. В целом, это возможно следует из способа распараллелирования — распределяется только один, внешний цикл. При его значении в случае самого большего датасета, распределяется слишком маленькое число итераций со слишком большим числом вычислений. Решением было бы переписать тройной цикл с прямым обходом по измерениям, на более сложный обход индексов, или использовать компиляторы с поддержкой OpenMP директивы *collapse(N)* — в этот раз она не использовалась, так как плохо поддерживается компилятором *icc*.


Compiler gcc with different data size and threads for heat-3d-for
Num threads
Data size
1e6
Execution time (5 starts averaging) --- sec


Compiler icc with different data size and threads for heat-3d-for
Num threads
Data size
1e6
Execution time (5 starts averaging) --- sec


Compiler gcc with different data size and threads for heat-3d-task
Num threads
Data size
1e6
Execution time (5 starts averaging) --- sec


Compiler icc with different data size and threads for heat-3d-task
Num threads
Data size
1e6
Execution time (5 starts averaging) --- sec