Лабораторная работа №5

OpenMP

Цель работы: знакомство с основами многопоточного программирования.

Инструментарий и требования к работе: работа выполняется на C/C++ (C11 и новее / C++20). Стандарт OpenMP 2.0. В отчёте указать язык и компилятор, на котором вы работали у себя локально. Требования для всех работ: Правила оформления и написания работ.

Описание работы

Необходимо написать программу, решающую одну из задач, описанных в разделе Варианты. Чем сложнее вариант, тем больше баллов за работу вы можете получить.

Помимо написания программы, необходимо провести замеры времени работы на вашем компьютере и привести графики времени работы программы (некоторые графики из следующих подпунктов можно объединить в один):

1. при различных значениях числа потоков при одинаковом параметре schedule* (без chunk_size);
2. при одинаковом значении числа потоков при различных параметрах schedule* (с chunk_size);
3. с выключенным openmp и с включенным с 1 потоком.

*schedule(kind[, chunk_size]) - kind принимает значение static или dynamic, chunk_size – 1 и несколько (3+) других значений

Для минимизации влияния степени загруженности процессора другими процессами, время должно усредняться по нескольким запускам.

Время в программе нужно измерять при помощи `omp_get_wtime()` (раздел 3.3).

Про schedule также можно почитать в спецификации (разделы 2.4.1 и Appendix D).

Важно:

1. Будет оцениваться как правильность результата, так и скорость работы.
2. Отправляемый на проверку код должен иметь наиболее быстрые (с вашей точки зрения) параметры schedule, т. к. именно по ним будет оцениваться скорость работы.
3. Без кода теория не оценивается, поэтому пытаться посылать отчет в таком случае бессмысленно.
4. Использовать `reduction` нельзя, техники оптимизации были рассмотрены на паре.

Варианты

В каждом варианте результат работы программы выводится в выходной файл.

Количество используемых потоков и время работы программы должны выводиться в поток вывода (stdout) в формате C: `“Time (%i thread(s)): %g ms\n”.`

Если openmp не используется, то выводить 0 в качества числа потоков. Если указано использовать число потоков по умолчанию - выводить реально используемое число потоков.

Время работы - без учета времени на считывание данных и вывод результата.

Рекомендуется использовать функции `fscanf` и `fprintf`.

Корректность входных данных гарантируется.

Аргументы программе передаются через командную строку:

`lab5 <кол-во_потоков> <имя_входного_файла> <имя_выходного_файла>`

где:

1.1. **`lab5`** - имя исполняемого файла (то есть - argv[0]).

1.2. Число потоков может равняться -1 и больше. 0 соответствует значению числа потоков по умолчанию. -1 - запуск без openmp.

1.3. В выходной файл записывается результат расчётов в формате “`%g %g\n`”.

Файл, содержащий функцию `main` должен называться `easy.cpp` / `easy.c` (в случае выполнение easy варианта) или `normal.cpp` / `normal.c` (расширение используется для определения языка: C или C++). Если назвать файл с исходным кодом main по-другому, то закрытые автотесты не пройдут, т.к. будет непонятно, какой вариант вы выполнили. Если выполнен easy, а затем normal, то файл с решением easy варианта можно оставить в репо переименовав в easy.cc (в таком случае он будет игнорироваться тестирующей системой).

Easy

Расчет площади круга методом Монте-Карло.

Во входном файле записаны количество точек `N` (целое положительное), которые будут генерироваться, и радиус `r` (вещественное положительное).

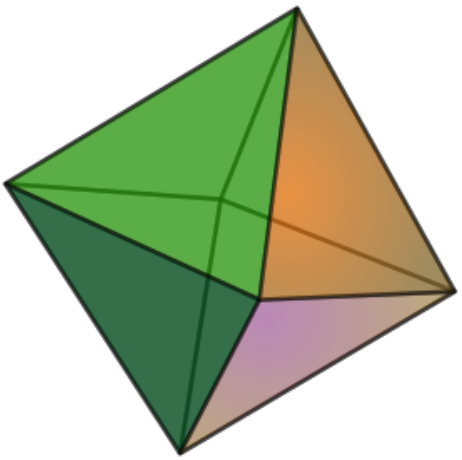
В выходном файле результаты расчётов: аналитический результат и значение, рассчитанное методом Монте-Карло.

Пример:

input	output
10 1	3.141593 2.8
100 1.0	3.141593 3.16
1000 1.	3.141593 3.148

Normal

Расчет объёма правильного октаэдра методом Монте-Карло.



Во входном файле записаны:

- в первой строке количество точек N (целое положительное), которые будут генерироваться;
- в остальных строках - координаты 3 вершин (вещественные значения) в формате `(X Y Z)`. Гарантируется, что указанные вершины однозначно задают октаэдр.

В выходном файле результаты расчётов: аналитический результат и значение, рассчитанное методом Монте-Карло.

Пример:

input	output
1000000000 (-2.37648 -1.28132 1.64637) (2.94148 1.12998 0.266185) (-0.818226 2.66201 1.49807)	36 35.9992

Модификация ППА

В работе необходимо реализовать функция многопоточного расчёта площади/объёма в 3 вариантах, существенно отличающихся используемым генератором псевдослучайных чисел.

Аргументы программе теперь передаются в след. формате:

`lab5 <кол-во_потоков> <имя_входного_файла> <имя_выходного_файла> <генератор>`

где **`генератор`** – порядковый номер реализации. Возможные значения: 0, 1, 2. Этот аргумент также может быть не указан, тогда запускается самый оптимальный (с вашей точки зрения).

Добавить график зависимости времени работы реализаций при оптимальной конфигурации schedule (определённой ранее) от числа потоков. Все замеры (по трём генераторам) должны быть размещены на одном графике. По графику необходимо сделать выводы.

Содержание отчета

1. Минититульник (таблица с ФИО, группой и названием работы из шаблона).
2. Ссылка на репозиторий.
3. Инструментарий (язык и версия компилятора/интерпретатора).
4. *Результат работы написанной программы* (то, что выводится в стандартный поток вывода и выходной на тесте с наилучшим распределением потоков) с указанием *процессора*, на котором производилось тестирование.
5. Описание конструкций OpenMP для распараллеливания команд. Какие конструкции использовались и как они работают.

6. Описание работы написанного кода. Как реализовано распараллеливание кода при помощи ранее описанных конструкций. При описании работы написанного кода может быть полезно приложить небольшие рисунки для иллюстрации пояснений и ссылки на соответствующие документы (а не просто фразу “взял из спецификации”).
7. Тестирование. Раздел должен содержать графики, которые подробно описаны в Описание работы. По каждому графику необходимо сделать выводы.
8. Ссылки на используемые источники.

Порядок сдачи работы

1. Выполнить работу.
2. Оформить миниотчёт в формате pdf.
3. Загрузить файл отчета и файлы с исходным кодом (расширения *.c/*.h/*.cpp/*.hpp) в выданный вам репозиторий в корень.
4. Запустить автотесты (на GH проверяется совпадение с ожидаемым значением с относительной погрешностью 1e-2). Подробнее: Автотесты на GitHub - comp-arch-course (gitbook.io) ВАЖНО: если не проходит автотест на GH, то запуска на закрытом тесте не будет.
5. Отправить на проверку работу (в открытом PR отмечаем Assignee Викторию и ставим label submit) и ждём ответа.

В репозиторий необходимо загружать файл с отчётом в формате pdf, названным в формате **“Фамилия_Имя_Группа_НомерРаботы.pdf”**. Номер группы – только 2 последние цифры группы, номер работы – порядковый номер лабораторной работы: 1, 2 и т.д.

Требования к работе программы

2. Корректно выделяется и освобождается память, закрываются файлы, есть обработка ошибок: не удалось открыть файл, формат файла не поддерживается.
3. Если программе передано значение, которое не поддерживается – следует сообщить об ошибке.
4. В программе можно вызывать только стандартные библиотеки (например, <bits/stdc++.h> таковой не является и ее использование влечет за собой потерю баллов). То есть сторонние библиотеки использовать нельзя (библиотека <omp.h> и модули для подключения openmp конечно разрешены).
5. Если программа использует библиотеки, которые явно не указаны в файле с исходным кодом (например, <algorithm>), то за это также будут снижаться баллы.

Полезное

Build (сборка решения на GH под C++) : `clang++ -std=c++20 -D _CRT_SECURE_NO_WARNINGS -D _USE_MATH_DEFINES -fopenmp -O2 <все *c/*.cpp файлы во всех подпапках репозитория> -o <execute file>`

C/C++

Для тех, кто не знает C/C++, предлагается ознакомится с содержимым файла ".github/workflows/example.cpp", в котором приведён код небольшой программы. В нём показано, как читать из файла, создавать массивы данных, проходиться по нему и что-то записывать в выходной файл с освобождением памяти программы.

Синтаксис языка и полезные функции можно посмотреть на следующем сайте: cppreference.com

Компиляция

Для включения OpenMP нужно указать ключ компиляции.

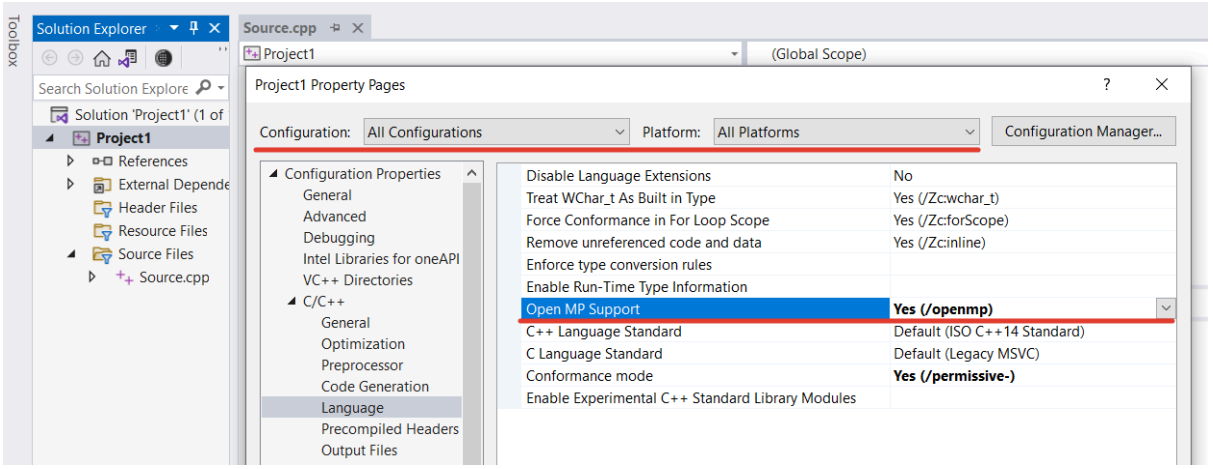
Компиляция через командную строку

Компилятор	Ключ компиляции
msvc (компилятор от Microsoft Visual Studio)	/openmp
gcc и clang	-fopenmp

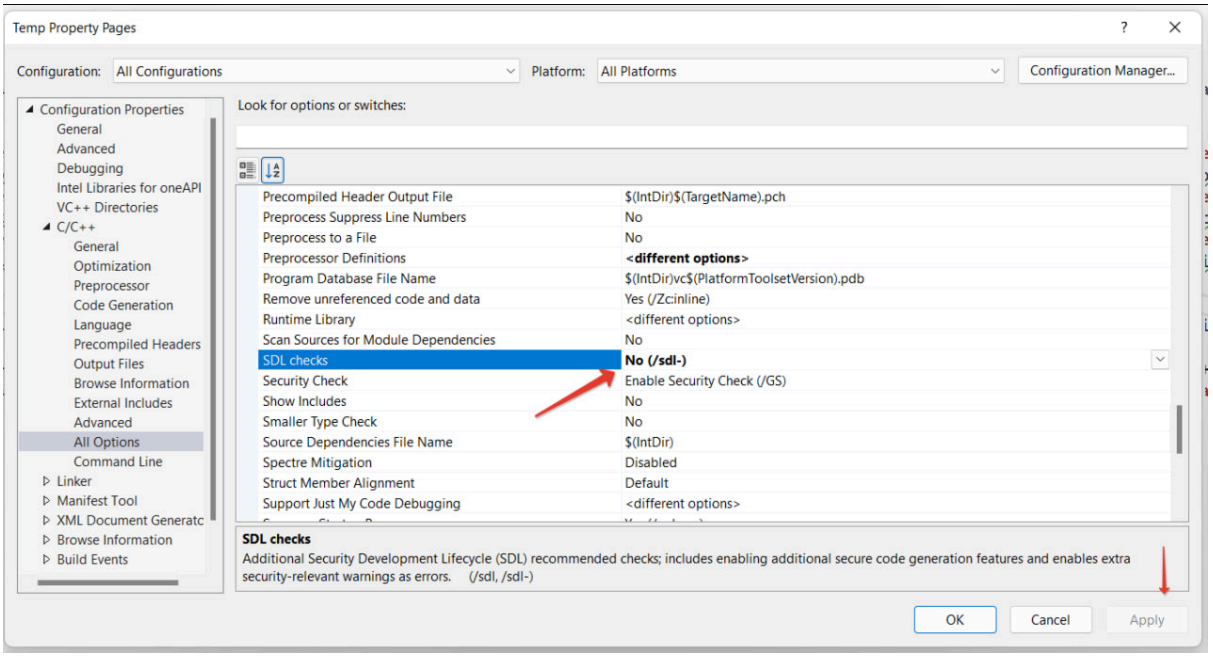
Примечание: clang, который установлен на MacOS по умолчанию, может не содержать OpenMP. Нужно либо установить полный clang, либо поставить gcc. Подробнее можно посмотреть здесь: https://www.programmersought.com/article/93289356924/

Visual Studio

Свойства проекта (ПКМ по проекту в обозревателе проектов) - C/C++ - Language - OpenMP support - Yes.



Дополнительно стоит отключить SDL check, чтобы “немодные” с точки зрения MSVC функции по типу fopen можно было использовать.



CMakeLists

```
OPTION (USE_OpenMP "Use OpenMP" ON)
IF(USE_OpenMP)
    FIND_PACKAGE(OpenMP)
    IF(OPENMP_FOUND)
        SET(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} ${OpenMP_C_FLAGS}")
        SET(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} ${OpenMP_CXX_FLAGS}")
    ENDIF()
ENDIF()
```

Для владельцев Mac на ARM64 также может помочь `set(CMAKE_OSX_ARCHITECTURES x86_64)`