

Цель работы: знакомство со стандартом распараллеливания команд OpenMP.

Инструментарий и требования к работе: C++ с библиотекой <omp.h>

Теоретическая часть

Описание OpenMP

OpenMP позволяет добавлять в программы параллельность, не задумываясь о самостоятельном создании, синхронизации, удалении потоков, а просто указывая необходимые директивы компилятора. Компилятор интерпретирует эти директивы и вставляет в соответствующие места программы библиотечные вызовы для распараллеливания участков кода.

Параллельная область объявляется с помощью директивы `omp parallel` с набором необязательных параметров, рассмотренных ниже.

После запуска программы создается единственный процесс с единственным master-потоком. Встретив параллельную область, master порождает несколько slave-потоков. Можно регулировать общее количество потоков с помощью функции `omp_set_num_threads` или переменной окружения `OMP_NUM_THREADS`. Число потоков по умолчанию равно количеству вычислительных ядер. После выполнения параллельного участка кода все потоки, кроме основного, завершаются.

У каждого потока есть собственная память для локальных переменных и доступ к общей памяти процесса.

Для определения, какие переменные считать глобальными, а какие локальными к директиве `parallel` добавляются параметры `private()` и `shared()` со списками имён соответствующих переменных. Есть так же параметр `default`, с помощью которого можно либо считать все переменные глобальными (значение `shared`), либо обязать прописывать `private` и `shared` вручную (значение `none`).

Если потоки обращаются к глобальной переменной, может возникнуть race condition (состояние гонки): если несколько потоков одновременно хотят изменить переменную, неизвестно в каком порядке они это сделают, и какой будет результат; если один поток хочет читать переменную, а другой изменять, неизвестно, старое или новое значение прочитает первый.

Для предотвращения этого существует директива `critical`, в которую обрамляется блок кода, взаимодействующий с глобальной переменной. Этот блок будет выполняться в каждый момент времени только одним потоком, а перед входом и после выхода из него синхронизировать со всеми потоками информацию об изменении переменной. Это очень сильно замедляет программу, поэтому в некоторых случаях (инкремент, декремент и $x = x \text{ op } y$, где op одна из этих операций: `+`, `*`, `-`, `/`, `&`, `^`, `|`, `<<`, `>>`) есть аналогичная оптимизированная директива `atomic`.

В определенных случаях типа суммирования, умножения, поиска максимума в цикле можно добавить параметр `reduction(операция : переменная)` в директиву `parallel`. Тогда у каждого потока создается локальная копия глобальной переменной, а в конце параллельного участка ко всем этим копиям применится соответствующая операция, и результат запишется в глобальную переменную.

Для распараллеливания циклов есть директива `omp for`, которая часто совмещается с `omp parallel` (получается `omp parallel for`). Она распределяет итерации цикла между потоками в соответствии с параметром `schedule`.

- `schedule(static)` — итерации цикла будут поровну (приблизительно) поделены между потоками
- `schedule(static, k)` — блочно-циклическое распределение итераций. Каждый поток получает k итераций в начале цикла, затем (если остались итерации) процедура распределения продолжается. Планирование выполняется один раз, таким образом, каждый поток сразу узнает, какие итерации будет выполнять
- `schedule(dynamic)`, `schedule(dynamic, k)` — динамическое планирование. По умолчанию параметр опции равен 1. Каждый поток

получает k итераций, выполняет их и запрашивает новую порцию среди еще нераспределенных. В отличие от статического планирования, выполняется многократно (во время выполнения программы). Конкретное распределение итераций между потоками зависит от темпов работы потоков и трудоемкости итераций

- `schedule(guided)`, `schedule(guided, k)` — разновидность динамического планирования с изменяемым при каждом последующем распределении числе итераций. Распределение начинается с некоторого начального размера, зависящего от реализации библиотеки до значения, задаваемого в опции (по умолчанию 1). Размер выделяемой порции зависит от количества еще нераспределенных итераций

Описание алгоритма согласно варианту

Вариант: 9

Условие:

Автоматическая коррекция яркости изображения в цветовом пространстве YCbCr.601.

Значение пикселей изображения находится в диапазоне [0; 255]. Изображение может иметь плохую контрастность: используется не весь диапазон значений, а только его часть. Например, если самые тёмные места изображения имеют значение 20.

Задание состоит в том, чтобы изменить значения пикселей таким образом, чтобы получить максимальную контрастность (диапазон значений [0; 255]) и при этом не изменить цветность (оттенок). Этого можно достигнуть регулировкой контрастности в канале яркости Y цветового пространства YCbCr (601 в РС диапазоне: [0; 255]).

Важно: согласно стандарту PNM изображения хранятся в цветовом пространстве RGB.

`<параметры_алгоритма> = <имя_входного_файла>`

Входной файл содержит данные в формате PPM (P6).

В качестве выходного файла будет новое изображение в формате PPM (P6). Имя выходного файла для данного варианта гарантировано будет указано в аргументах.

Алгоритм:

Для каждого пикселя перевести RGB в YCbCr по формулам из спецификации, определить диапазон Y, для каждого пикселя пересчитать Y в диапазоне 0-255, затем перевести YCbCr в RGB по формулам из спецификации.

Практическая часть

Описание работы кода

1. Берем количество тредов из параметров запуска или дефолтное из `omp_get_max_threads`.
2. Читаем длину и ширину, представленные посимвольно в ASCII в ppm-файле.
3. Сохраняем информацию о каждом пикселе, значения r, g, b читаются уже побайтово. Это ввод, который невозможно распараллелить. Пиксели хранятся в одномерном массиве длины `width*height`, потому что итерирование по такому массиву параллелится лучше, чем по двумерному с помощью вложенных циклов.
4. Распараллеленно проходим по всем пикселям и переводим в YCbCr по спецификации jreg, потому что она соответствует BT.601, но дает значения в диапазоне 0-255. При этом ищется минимальный и максимальный Y; чтобы не возникал race condition, используется параметр `reduction`.
5. Распараллеленно проходим по всем пикселям и переводим в RGB по спецификации jreg. Для этого используется Y, пересчитанный по

формуле для перевода диапазона Y_{min} - Y_{max} в диапазон 0-255: $(Y - Y_{min}) * 255 / (Y_{max} - Y_{min})$.

6. Записываем в выходной файл заголовок ppm как строку и значения r, g, b каждого пикселя как байты. Это вывод, который невозможно распараллелить.

Графики

На графиках ниже бралось среднее за 10 запусков время работы в миллисекундах для картинки разрешением 1920x1080.

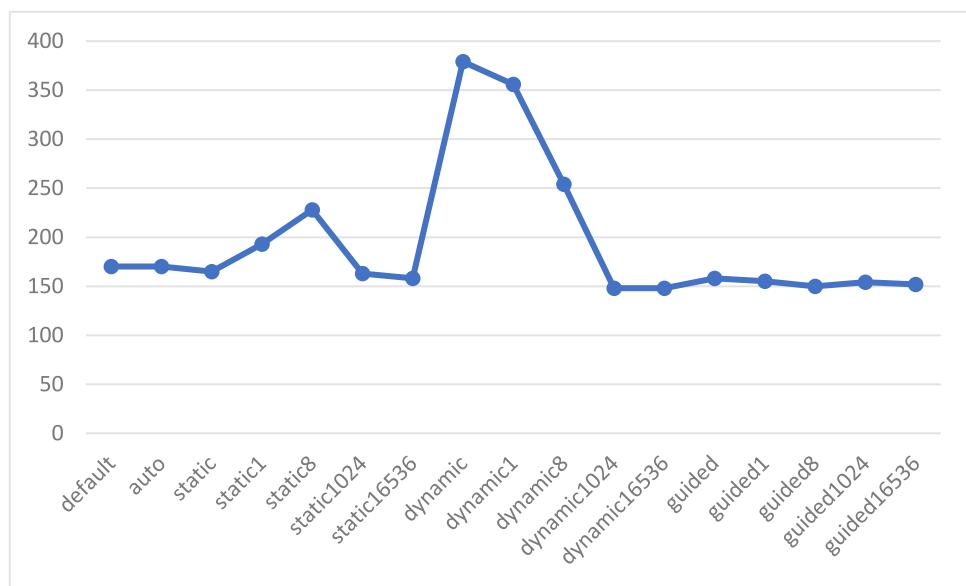


Рисунок №1 – Время работы при 6 потоках и различных значениях schedule

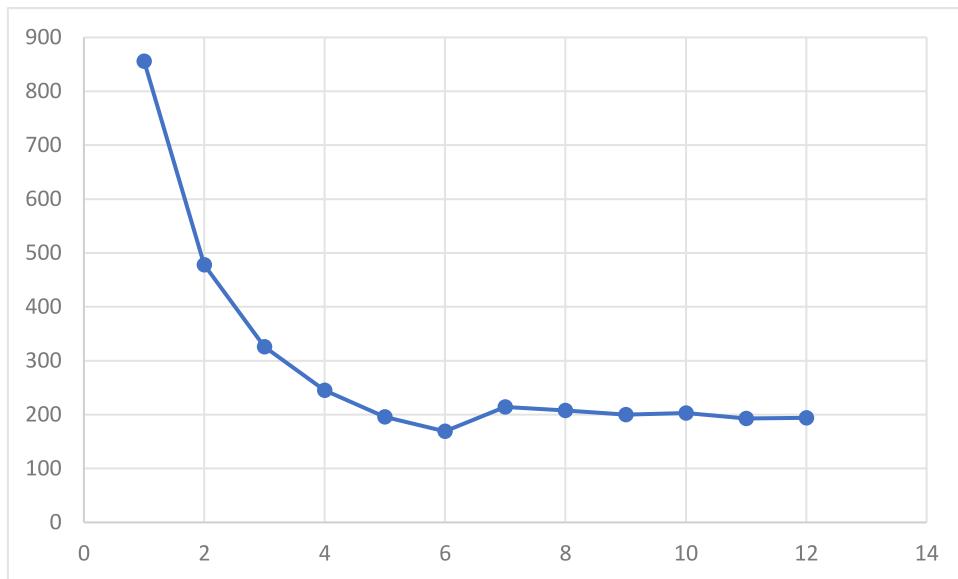


Рисунок №2 – Время работы при schedule(static) и различном количестве ПОТОКОВ

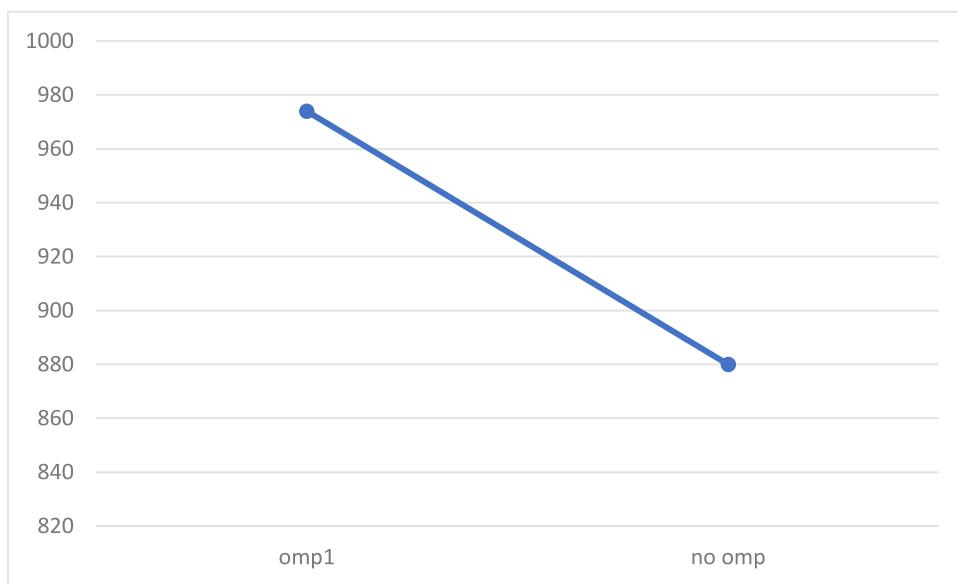


Рисунок №3 – Время работы с отключенным openmp и с одним потоком

Пример работы



Рисунок №4 – Исходная картинка с пониженной контрастностью



Рисунок №5 – Результат работы программы