Теория параллелизма

Отчет

Уравнение теплопроводности (Разностная схема)

Выполнила Рубан Василина Александровна, 22930

31.05.2024

Цель работы:

- Ознакомиться с OpenACC.

- Реализовать решение уравнения теплопроводности на C++ с использованием разностной схемы (пятиточечный шаблон).

- Изучить методы линейной интерполяции для задания граничных условий.

- Оптимизировать программу для работы на CPU и GPU.

- Сравнить производительность программы на CPU и GPU.

- Провести профилирование программы с использованием Nsight Systems.

Используемый компилятор: pgc++.

Используемый профилировщик: "Nsight Systems".

Как производились замеры времени работы: для измерения времени выполнения программы была использована библиотека std::chrono.

Выполнение на CPU

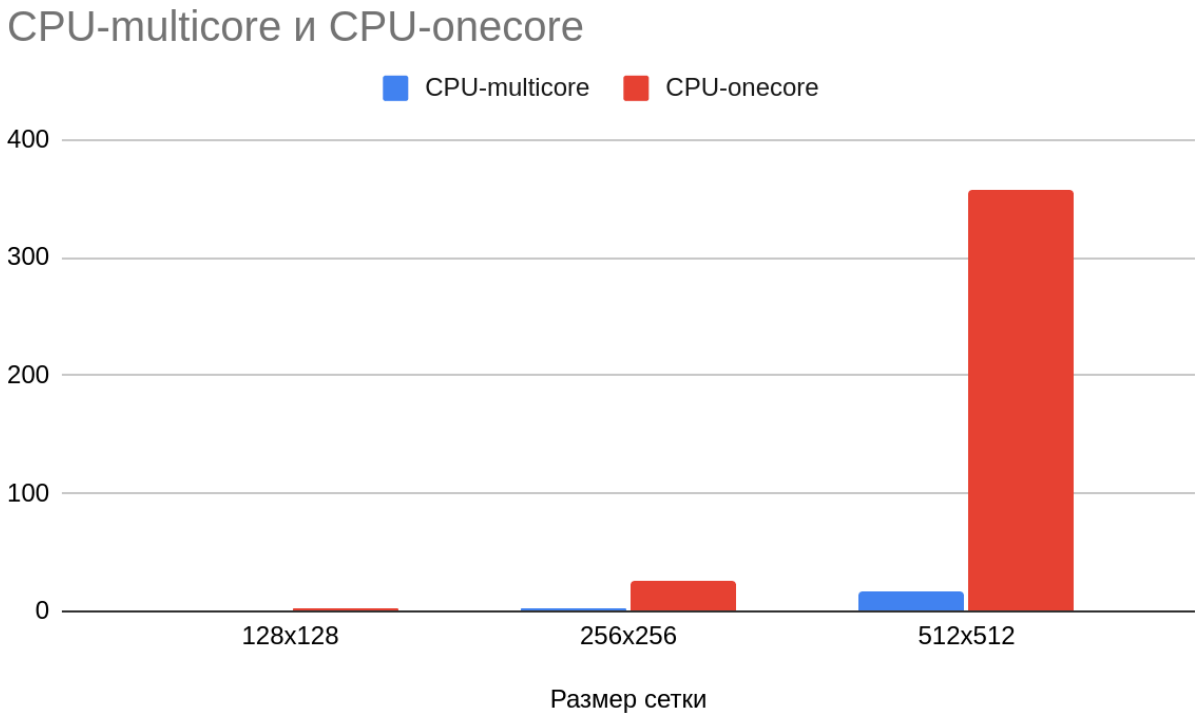
CPU-onecore

Размер сетки	Время выполнения, в секундах	Точность	Количество итераций
128x128	1.812	1e-6	36700
256x256	26.105	1e-6	125000
512x512	356.595	1e-6	409700

CPU-multicore

Размер сетки	Время выполнения, в секундах	Точность	Количество итераций
128x128	0.331	1e-6	18700
256x256	2.209	1e-6	52700
512x512	17.773	1e-6	121700

Диаграмма сравнения время работы CPU-one и CPU-multi



Выполнение на GPU

Размер сетки	Время выполнения, в секундах	Точность	Количество итераций
128x128	0.384	1e-6	36700
256x256	1.400	1e-6	125000
512x512	5.367	1e-6	409800
1024x1024	42.845	1e-6	1274200

Оптимизации:

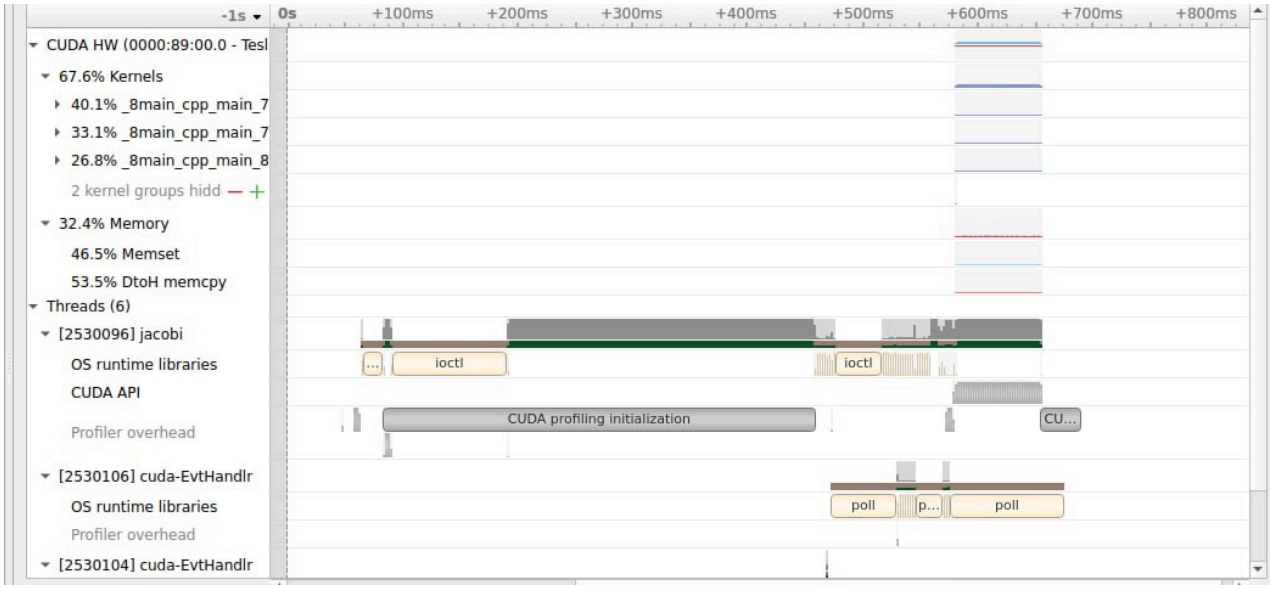
Оптимизация цикла с использованием директивы collapse(2):

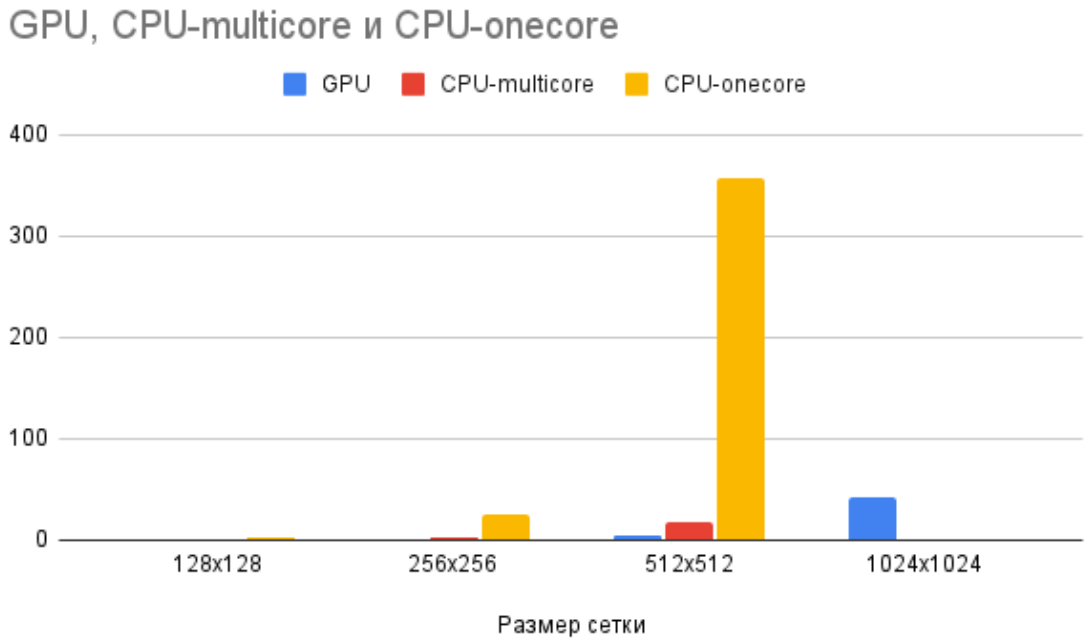
Использование директивы collapse(2) в OpenACC позволяет обрабатывать два вложенных цикла как один, что увеличивает количество доступных параллельных итераций и улучшает балансировку нагрузки на устройствах с большим количеством потоков.

Оптимизация вычисления ошибки с помощью reduction(max:err):

Использование директивы reduction(max:err) для вычисления максимальной ошибки позволяет эффективно параллелизовать операцию редукции, что ускоряет вычисление максимального значения ошибки.

Инициализация массивов с использованием memset.


-1s
0s
+100ms
+200ms
+300ms
+400ms
+500ms
+600ms
+700ms
+800ms
CUDA HW (0000:89:00.0 - Tesl
67.6% Kernels
40.1% _8main_cpp_main_7
33.1% _8main_cpp_main_7
26.8% _8main_cpp_main_8
2 kernel groups hidd
32.4% Memory
46.5% Memset
53.5% DtoH memcpy
Threads (6)
[2530096] jacobi
OS runtime libraries
CUDA API
Profiler overhead
[2530106] cuda-EvtHandlr
OS runtime libraries
Profiler overhead
[2530104] cuda-EvtHandlr
...
ioctl
ioctl
CUDA profiling initialization
CU...
poll
p...
poll



Вывод:

Наибольшее время, как и ожидалось, занимает выполнение на CPU-onecore, а наименьшее - на GPU.