МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №*2*
по курсу «Параллельные и распределенные вычисления»

Обработка изображений на GPU. Фильтры.

Выполнил: *Д.А. Коряков*
Группа: *М8О-211М-23*
Преподаватель: А.Ю. Морозов

Москва, 2024

Условие

Цель работы. Научиться использовать GPU для обработки изображений. Использование текстурной памяти и двухмерной сетки потоков.

Вариант 5. Выделение контуров. Метод Робертса.

Входные данные. На первой строке задается путь к исходному изображению, на второй, путь к конечному изображению. $w*h \leq 10^8$.

Формат изображений. Изображение является бинарным файлом, со следующей структурой:

width(w)	height(h)	r g b a	r g b a	r g b a	...	r g b a	r g b a
4 байта, int	4 байта, int	4 байта, значение пикселя [1,1]	4 байта, значение пикселя [2,1]	4 байта, значение пикселя [3,1]	...	4 байта, значение пикселя [w - 1, h]	4 байта значение пикселя [w,h]

В первых восьми байтах записывается размер изображения, далее построчно все значения пикселей, где

- r -- красная составляющая цвета пикселя
- g -- зеленая составляющая цвета пикселя
- b -- синяя составляющая цвета пикселя
- a -- значение альфа-канала пикселя

В данной лабораторной работе используются только цветовые составляющие изображения (r, g, b), альфа-канал не учитывается. Ограничение: $w < 2^{16}$ и $h < 2^{16}$. Для любых индексов i и j, координаты пикселя [ip, jp] будут определятся следующим образом:

ip := max(min(i, h), 1)

jp := max(min(j, w), 1)

Пример:

Входной файл	hex: in.data	hex: out.data
in.data out.data	03000000 03000000 01020300 04050600 07080900 09080700 06050400 03020100 00000000 14141400 00000000	03000000 03000000 04040400 03030300 07070700 0C0C0C00 12121200 03030300 1C1C1C00 1C1C1C00 00000000
in.data out.data	03000000 03000000 00000000 00000000 00000000 00000000 80808000 00000000 00000000 00000000 00000000	03000000 03000000 80808000 80808000 00000000 80808000 80808000 00000000 00000000 00000000 00000000

Программное и аппаратное обеспечение

IDE: CLion 2024.2.3 Build #CL-242.23726.125, built on October 27, 2024

gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0
Copyright (C) 2021 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2023 NVIDIA Corporation
Built on Wed_Nov_22_10:17:15_PST_2023
Cuda compilation tools, release 12.3, V12.3.107
Build cuda_12.3.r12.3/compiler.33567101_0

GPU:

Compute capability	: 8.6
Name	: NVIDIA RTX A6000
Total Global Memory	: 51032686592
Shared memory per block	: 49152
Max threads per block	: (1024, 1024, 64)
Max block	: (2147483647, 65535, 65535)
Total constant memory	: 65536
Multiprocessors count	: 84

CPU:

Architecture:	x86_64
CPU op-mode(s):	32-bit, 64-bit
Address sizes:	46 bits physical, 48 bits virtual
Byte Order:	Little Endian
CPU(s):	32
On-line CPU(s) list:	0-31
Vendor ID:	GenuineIntel
Model name:	Intel(R) Xeon(R) Silver 4215R CPU @ 3.20GHz
CPU family:	6
Model:	85
Thread(s) per core:	2
Core(s) per socket:	8
Socket(s):	2
Stepping:	7
CPU max MHz:	4000.0000
CPU min MHz:	1000.0000

BogoMIPS: 6400.00

Flags: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc art arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_cpl smx est tm2 ssse3 s dbg fma cx16 xtpr pdcm pcid dca sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch cpuid_fault epb cat_l3 cdp_l3 invpcid_single intel_ppin ssbd mba ibrs ibpb stibp ibrs_enhanced fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid cqm mpx rdt _a avx512f avx512dq rdseed adx smap clflushopt clwb intel_pt avx512cd avx512bw avx512vl xsaveopt xsavec xgetbv1 xsaves cqm_llc cqm_occup_llc cqm_mbm _total cqm_mbm_local dtherm ida arat pln pts pku ospke avx512_vnni md_clear flush_l1d arch_capabilities

Caches (sum of all):

L1d: 512 KiB (16 instances)

L1i: 512 KiB (16 instances)

L2: 16 MiB (16 instances)

L3: 22 MiB (2 instances)

NUMA:

NUMA node(s): 2

NUMA node0 CPU(s): 0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30

NUMA node1 CPU(s): 1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31

Vulnerabilities:

Gather data sampling: Mitigation; Microcode

Itlb multihit: KVM: Mitigation: VMX unsupported

L1tf: Not affected

Mds: Not affected

Meltdown: Not affected

Mmio stale data: Mitigation; Clear CPU buffers; SMT vulnerable

Retbleed: Mitigation; Enhanced IBRS

Spec rstack overflow: Not affected

Spec store bypass: Mitigation; Speculative Store Bypass disabled via prctl and seccomp

Spectre v1: Mitigation; usercopy/swapgs barriers and __user pointer sanitization

Spectre v2: Mitigation; Enhanced IBRS, IBPB conditional, RSB filling, PBRSB-eIBRS SW sequence

Srbds: Not affected

Tsx async abort: Mitigation; TSX disabled

RAM:

	total	used	free	shared	buff/cache	available
Mem:	62Gi	9.6Gi	12Gi	10Mi	40Gi	52Gi
Swap:	8.0Gi	4.0Mi	8.0Gi			

Storage:

Filesystem	Size	Used	Avail	Use%	Mounted on
tmpfs	6.3G	3.5M	6.3G	1%	/run
/dev/mapper/ubuntu--vg-ubuntu--lv	1.9T	1.6T	182G	90%	/
tmpfs	32G	416K	32G	1%	/dev/shm
tmpfs	5.0M	4.0K	5.0M	1%	/run/lock
/dev/nvme0n1p2	2.0G	340M	1.5G	19%	/boot
/dev/nvme0n1p1	1.1G	6.1M	1.1G	1%	/boot/efi
tmpfs	6.3G	120K	6.3G	1%	/run/user/1000
tmpfs	6.3G	68K	6.3G	1%	/run/user/1001

OS:

Linux sedanllm 5.15.0-97-generic #107-Ubuntu SMP Wed Feb 7 13:26:48 UTC 2024 x86_64 x86_64 x86_64 GNU/Linux

Метод решения

Описание алгоритма

Программа выполняет выделение границ изображения с использованием оператора Робертса на GPU:

1. **Чтение данных**: Размеры изображения и пиксели (`uchar4`) считываются из входного файла.
2. **Инициализация текстурного объекта**: Данные изображения загружаются в `cudaArray`, создается текстурный объект для быстрого доступа к пикселям.
3. **Вычисление градиента на GPU**:
    - Ядро CUDA обрабатывает каждый пиксель, вычисляет градиенты (по Робертсу), преобразует результат в градации серого и записывает в выходной массив.
4. **Вывод данных**: Обработанные пиксели копируются с GPU и записываются в файл.

Архитектура

- **Текстурный объект**: Используется для оптимального доступа к данным изображения.
- **Параллелизм**: Размеры блока (`32x32`) и сетки (`16x16`) обеспечивают распределение задач между нитями GPU.
- **Управление ошибками**: Макрос `CSC` проверяет корректность вызовов CUDA.

Описание программы

Программа разделена на один файл с основным кодом, содержащим всю логику загрузки данных, обработки на GPU и записи результата.

Основные типы данных

1. **uchar4**:
    - Тип данных для хранения пикселей изображения. Каждый пиксель состоит из четырех каналов (R, G, B, A), где x, y, z — цветовые компоненты, а w — альфа-канал.
2. **cudaTextureObject_t**:
    - Текстурный объект для эффективного доступа к данным изображения на GPU. Используется для ускорения операций чтения.
3. **cudaArray**:
    - Двумерный массив для хранения данных изображения в памяти устройства, совместимый с текстурными объектами.

Основные функции

1. **kernel (ядро CUDA)**:
    - Описание: Вычисляет градиенты изображения с использованием оператора Робертса.
    - Входные параметры:
        - tex: текстурный объект, содержащий исходные пиксели.
        - out: указатель на массив выходных пикселей.
        - w, h: ширина и высота изображения.
    - Логика:
        - Обрабатывает каждый пиксель, считывая данные через tex2D<uchar4>.
        - Преобразует цвет в яркость по формуле: $Y=0.299 \cdot R+0.587 \cdot G+0.114 \cdot BY = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot BY=0.299 \cdot R+0.587 \cdot G+0.114 \cdot B$.
        - Вычисляет градиенты по X и Y, их величину и нормализует результат в диапазоне [0, 255].
        - Записывает результат в массив out.
2. **Функции CPU**:

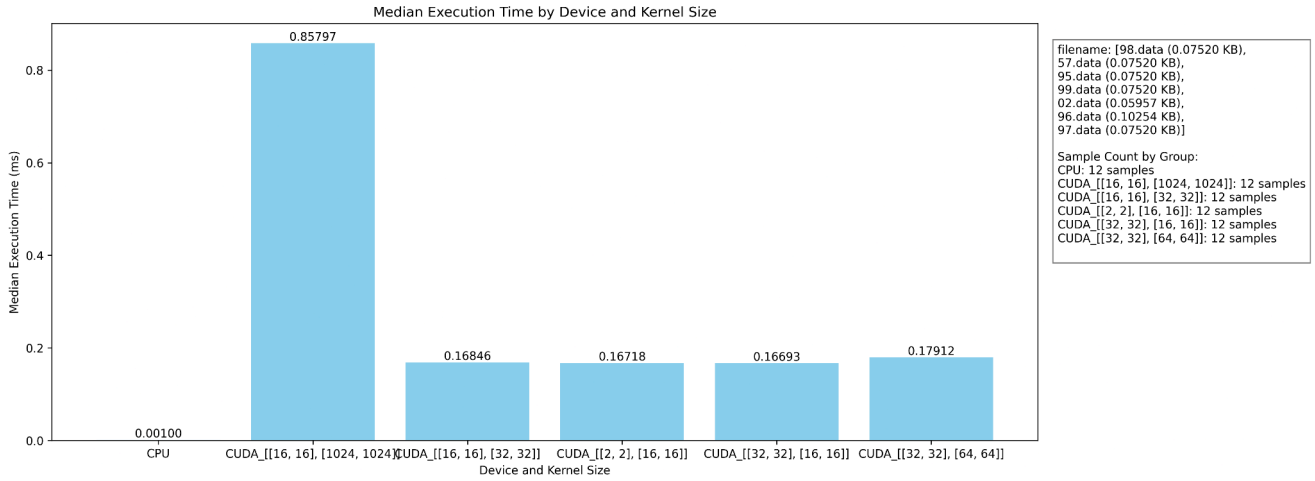
- **Чтение входных данных**:
  - Считывает ширину, высоту и массив пикселей из бинарного файла.
- **Инициализация текстуры**:
  - Создает cudaArray и текстурный объект для обработки на GPU.
- **Запуск ядра**:
  - Настраивает параметры блока (32x32) и сетки (16x16), запускает ядро для обработки изображения.
- **Вывод данных**:
  - Копирует результаты с устройства на хост и записывает их в файл.

Реализованные ядра

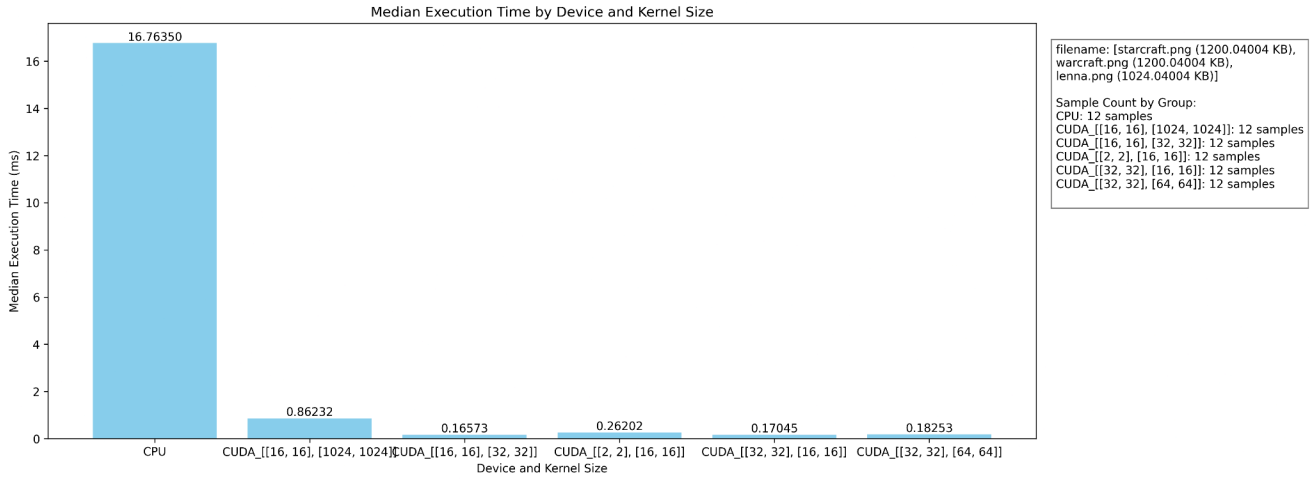
- **kernel**:
  - Выполняет преобразование изображения и вычисление градиентов:
    - Обеспечивает параллельную обработку каждого пикселя.
    - Использует текстурный объект для оптимизации доступа к данным.
    - Рассчитывает результат, который преобразуется в черно-белое изображение с выделением границ.

Результаты

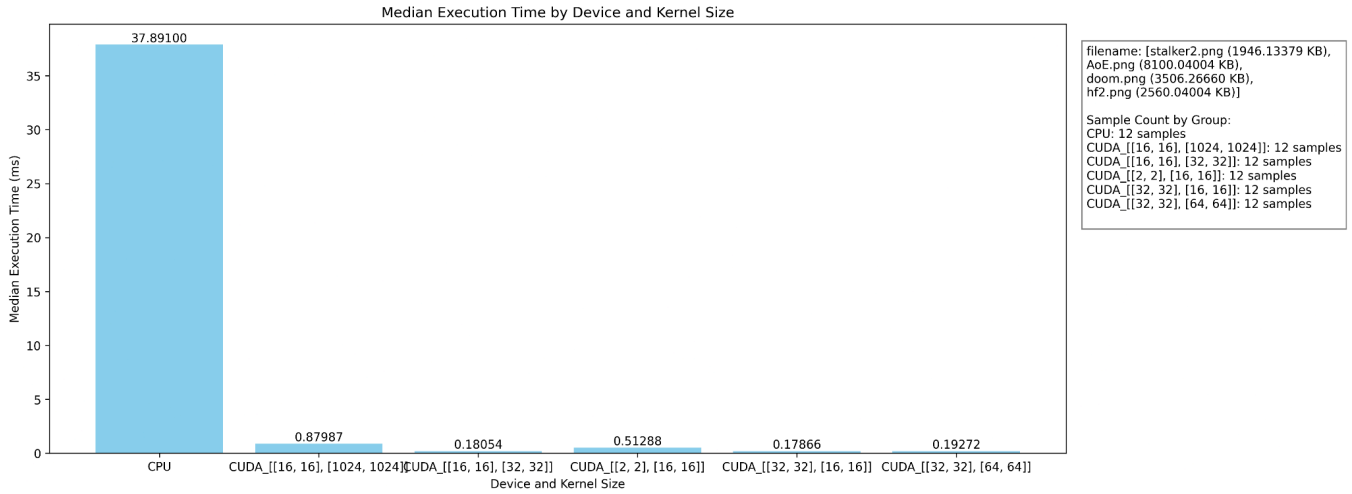
Мелкие изображения:



Средние изображения:



Крупные изображения:



Для проведения экспериментов, были написаны python скрипты, в дальнейшем они могут быть использованы для последующих ЛР. https://github.com/KoryakovDmitry/cuda-mpi-openmp

Основные выводы

Как мы можем заметить с мелкими входными данными CPU справляется быстрее, однако при обработке более крупных файлов, CUDA значительнее выигрывает.

Скриншот работы

На входе:

На выходе:

На входе:

На выходе:

Выводы

Область применения алгоритма

Алгоритм выделения границ с использованием оператора Робертса применяется в задачах компьютерного зрения, таких как обнаружение границ объектов, предварительная обработка изображений, анализ медицинских снимков и навигация роботов.

Типовые задачи

- Сегментация и анализ изображений.
- Обработка изображений для распознавания объектов.
- Обнаружение контуров для анализа формы.
- Выделение объектов на видео в реальном времени.

Сложность программирования

Реализация алгоритма на GPU требует навыков оптимизации под параллельные вычисления и тщательного подбора параметров CUDA.

Проблемы при решении задачи

- Подбор оптимальной конфигурации ядра для разных объемов данных.
- Накладные расходы на передачу данных при малых файлах.
- Сложности с отладкой кода CUDA.

Сравнение и объяснение результатов

- CPU vs CUDA: На малых данных CPU быстрее, но CUDA значительно выигрывает на больших объемах данных.
- Влияние конфигурации ядра: Для малых и средних данных оптимальны меньшие сетки, для больших — крупные блоки, такие как [32, 32], [16, 16].
- Эффективность CUDA: CUDA показала себя эффективной для обработки крупных изображений и подходит для задач реального времени в компьютерном зрении.