Вычисления на GPU
с использованием NVIDIA CUDA

Автор: студент 2-ПМИ МАГУ
Леутин Александр

Немного истории

- С самого появления GPU у разработчиков появилась идея перекладывать часть расчетов с CPU на GPU, но архитектура не позволяла это делать. Почти. Разработчикам удавалось выкрутиться и заставить GPU выполнять нужные инструкции, но зачастую это того не стоило.

Краткое введение в технологию

- Технология CUDA — это программно-аппаратная вычислительная архитектура Nvidia, основанная на расширении языка Си, которая даёт возможность организации доступа к набору инструкций графического ускорителя и управления его памятью при организации параллельных вычислений.

Основные возможности технологии

- Унифицированное программно-аппаратное решение для параллельных вычислений на видеочипах Nvidia;
- Стандартный язык программирования Си;
- Стандартные библиотеки численного анализа FFT (быстрое преобразование Фурье) и BLAS (линейная алгебра);
- Оптимизированный обмен данными между CPU и GPU;
- Взаимодействие с графическими API OpenGL и DirectX;
- Возможность разработки на низком уровне.

Основные сферы применения

- Симуляция поведения различных тел
- Обработка графики
- Расчет геометрии
- Вычисление различных хэшей
- Компьютерное зрение
- Искусственный интеллект

Техническая реализация

Для вызова функции на стороне GPU нужно:

1. Выделить память под аргументы
2. Скопировать данные с хоста в блок памяти GPU.
3. Произвести вызов функции (будет рассмотрено далее)
4. Освободить память после выполнения

Работа с памятью

Работа с памятью организована при помощи функций

- cudaMalloc – выделение блока памяти
- cudaMalloc3D – выделение блока под трехмерный массив (width*height*depth)
- cudaMalloc3DArray – аналогично cudaMalloc3D, но для массива таких объектов
- cudaMallocArray – выделение массива блоков
- cudaMallocPitch – выделение памяти под массивы*
- cudaFree – освободить блок памяти

Чуть подробнее про cudaMallocPitch

- Данная функция не только выделяет память под данные, но и гарантирует сохранение следующего соотношения

$$T * pElement = (T*)((char*)BaseAddress + Row * pitch) + Column;$$

- Что в итоге позволяет нам спокойно ориентироваться в памяти, зная базовый адрес, строку и значение pitch (которое получаем после cudaMallocPitch)

Исполнение инструкций

– Вызов функции, на стороне GPU идет немного необычно

$$someFunction <<< blocks, threadsPerBlocks >>> (args);$$

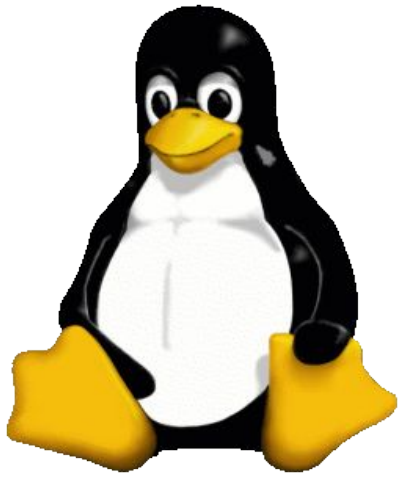
Так же функция должна иметь модификатор __global__

Плюсы технологии

- Быстрые вычисления
- Хорошая архитектура для многопоточности
- Удобный инструментарий и отсутствие лишних телодвижений при передаче инструкций (за исключением чутка странного вызова __global__ функций)
- Удобная работа с памятью
- Поддержка основных платформ

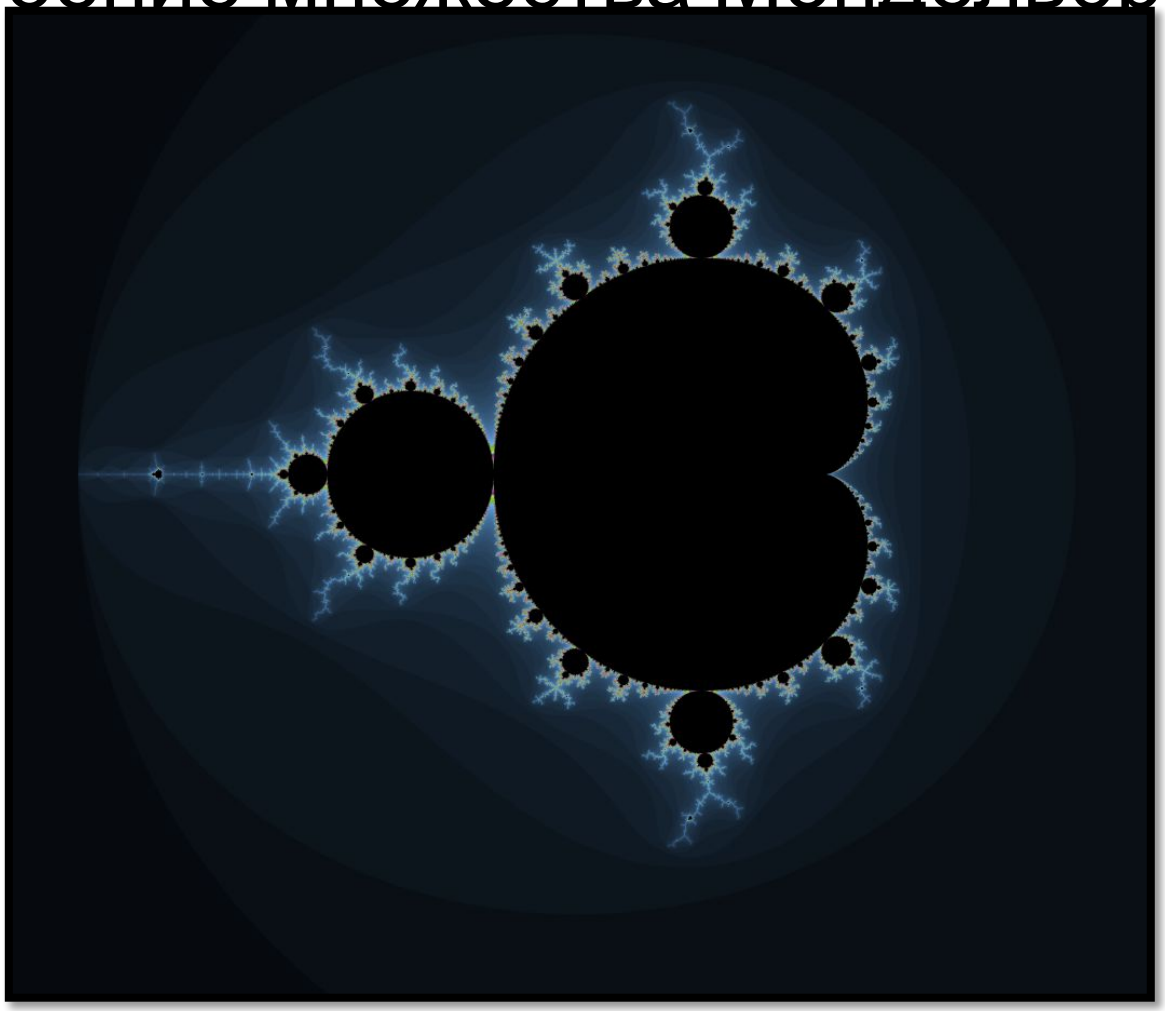
Минусы технологии

- Передача данных от CPU к GPU достаточно дорогая операция. Иногда это заставляет передавать лишние данные.
- Проприетарная архитектура CUDA.
- Устаревшая поддержка Visual Studio 2012, но кому это нужно, когда есть он

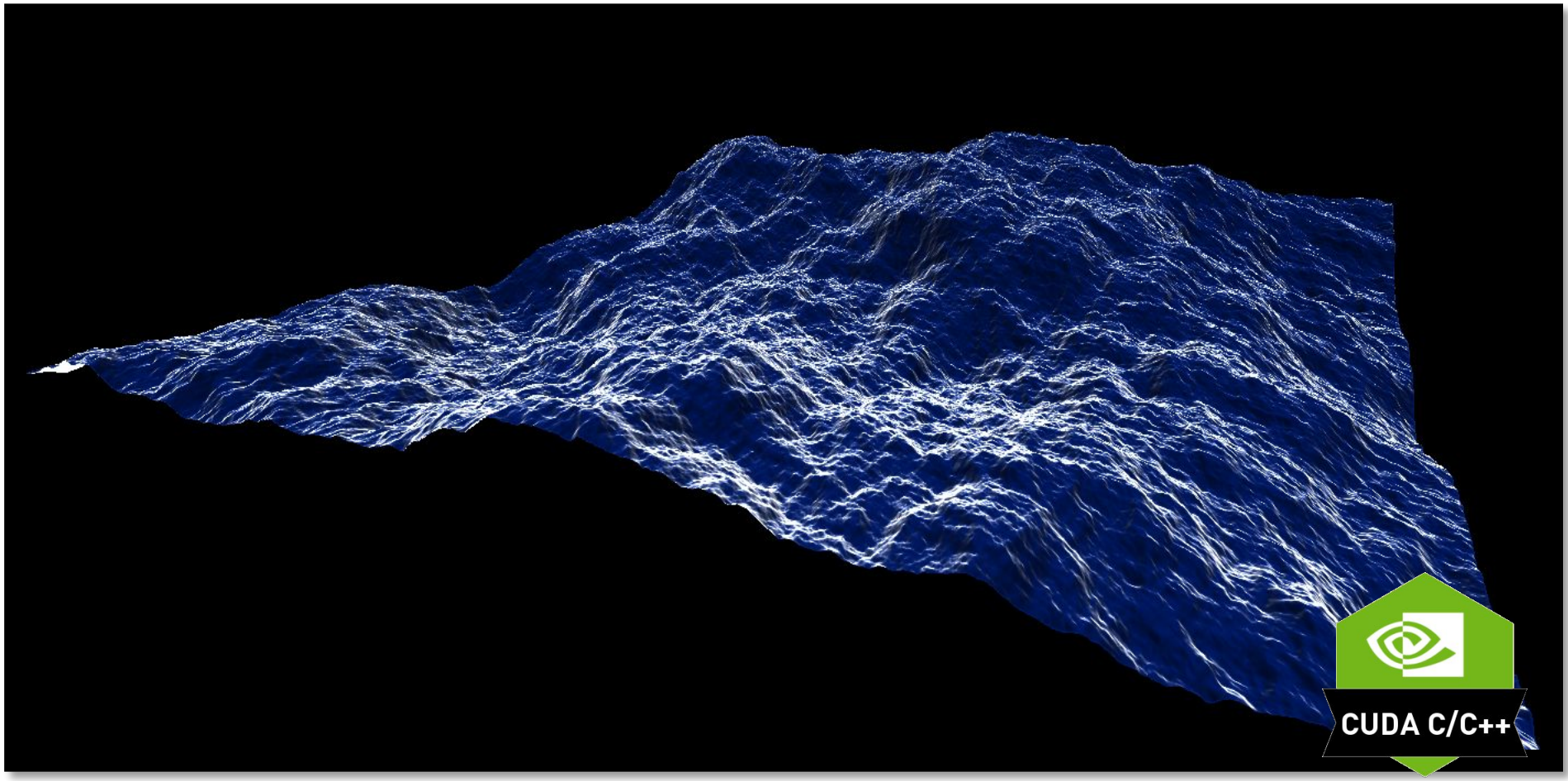


Примеры

- Построение множества Мондельброта



Симуляция поверхности воды



Спасибо за внимание