GPU плагин для системы LuNA

Выполнил: Веретенников А.А.

Руководитель: Спирин В.А.

На данный момент параллельная реализация программ в системе LuNA выполняется только на процессоре и NPU.

Для определенных задач эффективнее использовать GPU.

!

Актуальность

Цель и задачи

Цель: реализовать плагин, обеспечивающий разработку параллельных программ на GPU.

Задачи

_1_ OpenCL	Написать параллельную программу для сложения векторов с использованием OpenCL
_2_ CPU + LuNA	Написать параллельную программу для сложения векторов (2 версии). 1 версия с использованием OpenMP, 2 — для LuNA
_3_ GPU + LuNA	Написать параллельную программу для сложения векторов (2 версии). 1 версия с использованием CUDA, 2 — для LuNA
_4_ Обертка для nvcc	Создать обертку, обеспечивающую совместимость интерфейса nvcc с системой сборки
_5_ Скрипт «Генератор»	Написать скрипт для генерации ядра CUDA и выделения и очистки памяти.
_6_ Плагин	Реализовать плагин, обеспечивающий разработку параллельных программ на GPU

_1_ OpenCL

OpenCL — это

Открытый стандарт, фреймворк язык программирования для написания кода, выполняемого параллельно на различных типах процессоров:
Графических (GPU)
Центральных (CPU)
А так же DSP и FPGA

Результат

Реализована программа для параллельного сложения векторов с помощью OpenCL

_2_ CPU + LuNA

Проблем: нет
Результат: реализована параллельная программа для сложения векторов на CPU.

- В первой версии программы использовался OpenMP
- Во второй — LuNA

_3_ GPU + LuNA

Проблемы:

1. Конфликты при сборке проекта с помощью LuNA при установленном компиляторе nvcc.
2. Пользователь LuNA должен знать синтаксис CUDA.

Решения:

1. Реализована обертка над компилятором nvcc (об этом далее).
2. Реализован плагин для поддержки GPU. (об этом далее)

Решение: реализована параллельная программа для сложения векторов на GPU.

Программы протестированы и работают корректно

_4_ Обертка для nvcc

Проблема: возникали ошибки при сборке проекта, связанные с тем, что **nvcc** не распознает определенные флаги для g++.

Решение: написать **обертку для nvcc**, которая будет пропускать проблемные флаги через **-Xcompiler**

Теперь сборка проходит успешно

example_1 — vim file.fa — 33×9

```
import vector_add(
    value a,
    value b,
    name c,
    int n
) : GPU
```

-- INSERT --

example_1 — vim file.cu — 37×8

```
void vector_add(const double *a,
                const double *b,
                double *c,
                int n) {
    c[i] = a[i] + b[i];
}
```

All

example_1 — vim generated.cu — 86×38

```
#include "ucenv.h"
#define THREADS_PER_BLOCK 256
__global__ void vectorAdd(const double *a, const double *b, double *c, int n) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;

    if (i < n) {
        c[i] = a[i] + b[i];
    }
}

export "C" void vector_add(const DF &a, const DF &b, DF &c, int n) {
    const double *h_a = a.getData<double>();
    const double *h_b = b.getData<double>();

    double *d_a, *d_b, *d_c;
    cudaMalloc(&d_a, n * sizeof(double));
    cudaMalloc(&d_b, n * sizeof(double));
    cudaMalloc(&d_c, n * sizeof(double));

    cudaMemcpy(d_a, h_a, n * sizeof(double), cudaMemcpyHostToDevice);
    cudaMemcpy(d_b, h_b, n * sizeof(double), cudaMemcpyHostToDevice);

    int blockPerGrid = (n + THREADS_PER_BLOCK - 1) / THREADS_PER_BLOCK;

    double *data = c.create<double>(n);
    vectorAdd<<<blockPerGrid, THREADS_PER_BLOCK>>>(d_a, d_b, d_c, n);

    cudaDeviceSynchronize();

    cudaMemcpy(data, d_c, n * sizeof(double), cudaMemcpyDeviceToHost);

    cudaFree(d_a);
    cudaFree(d_b);
    cudaFree(d_c);
}
```

_5_ Скрипт «Генератор»

_6_ Плагин

Проблема

При сборке проекта используется скрипт, через который запускается «Генератор», но генератор должен получать на вход не только .cu файл, но и .fa, доступа к которому через скрипт нет...

Результат

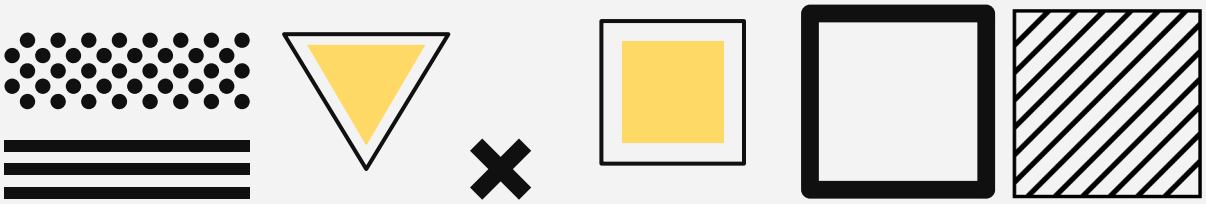
... Но есть ja — разбитый на токены файл .fa в формате .json
Переписать «Генератор» так, чтобы он на вход принимал .json.

Заключение

Реализован плагин для системы **LuNA**, обеспечивающий разработку параллельных программ для **GPU**.

Планы на будущее

Доработать плагин и исследовать наиболее эффективные сценарии перебрасывания данных между CPU и GPU



Спасибо за внимание!