Programowanie równoległe i rozproszone
Sprawozdanie końcowe z projektu

Maja Nagarnowicz 304472
Agnieszka Stefankowska 304498

15 czerwca 2023

1 Wstęp

Celem tego dokumentu jest przedstawienie wyników realizacji projektu w ramach przedmiotu Programowanie równoległe i rozproszone. Projekt ten polega na implementacji gry "Game of Life", napisanej w języku C++, z użyciem technologii CUDA do zrównoleglenia obliczeń i OpenGL do wizualizacji wyników. W ramach tego dokumentu omówimy uruchomienie i wywołanie programu, opis realizacji problemu oraz porównanie czasu pomiędzy programem sekwencyjnym a równoległym.

2 Opis uruchomienia i wywołania programu

Poniżej znajduje się instrukcja uruchomienia i wywołania naszego programu:

1. Otwórz terminal i przejdź do katalogu głównego projektu. Ten katalog powinien zawierać plik CMakeLists.txt.

```
cd sciezka/twojego/projektu
```

2. Utwórz katalog 'build', jeśli jeszcze nie istnieje, i przejdź do niego.

```
mkdir build
cd build
```

3. Uruchom cmake w celu generowania plików makefile.

```
cmake ..
```

4. Po utworzeniu plików Makefile, możemy zbudować nasz projekt za pomocą polecenia make.

```
make
```

5. Gdy nasz program jest skompilowany, możemy go uruchomić z różnymi flagami. W zależności od wybranej flagi, uruchamianie programu wygląda następująco:

   (a) Uruchomienie na CPU:

   ```
   ./main -c
   ```

   (b) Uruchomienie na GPU (z użyciem CUDA):

   ```
   ./main -g
   ```

   (c) Pomiary czasu dla CPU:

   ```
   ./main -t
   ```

   (d) Pomiary czasu dla GPU (z użyciem CUDA):

   ```
   ./main -T
   ```

3 Opis problemu

Gra 'Game of Life', stworzona przez brytyjskiego matematyka Johna Conwaya, jest automatem komórkowym, gdzie plansza składa się z siatki komórek, które mogą być 'żywe' lub 'martwe'. Zasady gry są proste, ale prowadzą do zaskakująco skomplikowanych rezultatów. W kolejnych turach, stan komórki zależy od liczby żywych sąsiadów. Jeśli komórka jest żywa i ma dwóch lub trzech żywych sąsiadów, to przeżywa. Natomiast jeśli komórka jest martwa i ma dokładnie trzech żywych sąsiadów, to ożywa. Wszystkie inne żywe komórki umierają w następnej turze. Dodatkowo, nasz projekt wprowadza element koloru komórek, gdzie kolor potomstwa jest średnią z kolorów jej sąsiadów.

4 Dokładny opis realizacji problemu

Zdecydowałyśmy się na użycie technologii CUDA i OpenGL w języku C++. CUDA pozwolił nam na wykorzystanie mocy obliczeniowej GPU do przyspieszenia obliczeń, a OpenGL na efektywną wizualizację wyników.

4.1 Przedstawienie siatki jako tekstury

W naszym projekcie plansza "Game of Life" została przedstawiona jako tekstura wizualizowana na ekranie. Tekstura ta jest definiowana jako wektor 2D złożony z parametrów typu float. Jedna komórka reprezentowana jest jako czwórka liczb typu float (R,G,B,A). Dla każdej iteracji gry, aktualizujemy stan komórek i ich kolory, a następnie aktualizujemy teksturę planszy na podstawie nowego stanu. Tekstura jest renderowana przy użyciu standardu OpenGL.

4.2 CUDA

W projekcie wykorzystujemy paradygmat obliczeń równoległych z użyciem CUDA, w którym zastosowano bloki wątków o rozmiarze 16x16.

Każdy blok składa się z 256 wątków (16x16), które są wykonywane równolegle, a każdy wątek odpowiada za przetwarzanie pojedynczej komórki na planszy gry. Taka organizacja pozwala na efektywne mapowanie problemu na architekturę GPU, gdzie problem przestrzenny (2D w naszym przypadku) jest obsługiwany przez bloki i wątki również zorganizowane w strukturę dwuwymiarową.

Liczba bloków, a co za tym idzie, całkowita liczba wątków, jest dynamicznie dostosowywana do rozmiaru planszy gry, co pozwala na efektywne pokrycie całej przestrzeni pikseli. Takie podejście sprawia, że architektura naszego rozwiązania jest skalowalna i potrafi dostosować się do różnych rozmiarów problemu.

5 Uzyskane przyspieszenie

Porównanie czasu jako średnia ze stu iteracji dla obliczania 2 073 600 komórek:

CPU: average calculation time over	95 ms
CPU: average rendering	97 ms
GPU: average calculation time	0.6 ms
GPU: average rendering time *	27.3 ms
GPU: average rendering time	1.8 ms

*Sytuacja w której GPU wykonuje obliczenia, a następnie przekazuje wyniki do CPU, które jest odpowiedzialne za dalsze instrukcje renderowania.
Do obliczenia przyśpieszenia użyłyśmy wzoru:

$$S = \frac{\text{czas wykonania zadania sekwencyjnie}}{\text{czas wykonania zadania równolegle}}$$

Używając powyższego wzoru, obliczyłyśmy przyśpieszenie jako S_1=158.3 dla obliczania stanu komórek oraz S_2=53.9 dla renderowania obrazu.