Spis treści

Rozdział 1

Wstęp

1.1 Wprowadzenie

Raytracing jest techniką służącą do generowania foto realistycznych obrazów scen 3D. Na przestrzeni lat technika ta ciągle się rozwijała. Doczekała się wielu modyfikacji, które usprawniają proces generowania realistycznej grafiki. Takimi technikami mogą być między innymi PathTracing, PhotonMapping, Radiostity i wiele innych. Z dnia na dzień wykorzystywanie raytracingu ciągle rośnie. W dzisiejszych czasach w grafice komputerowej oraz w kinematografii do uzyskania realistycznych efektów używana jest metoda śledzenia promieni. Dzięki takim zabiegom jesteśmy w stanie dosłownie zasymulować sceny oraz zjawiska, które nie muszą istnieć w rzeczywistym świecie. Czas generowania pojedynczej klatki/ujęcia takiej sceny niekiedy potrafi być liczony nawet w godzinach. Dlatego technika ta nie doczekała się jeszcze swojej wielkiej chwili w przemyśle rozrywkowym jakim są np. gry komputerowe oraz inne aplikacje generujące grafikę 3D w czasie rzeczywistym.

1.2 Dostępne technologie, pozwalające zrównoleglić obliczenia na kartach graficznych

Poniżej zaprezentowanych zostanie parę wybranych technologii wspomagających zrównoleglenie obliczeń. Niemniej jednak badania przeprowadzone i opisane w dalszej części pracy będą skupiały się na wykorzystaniu jednej z tych metod, a mianowicie technologii NVIDIA CUDA.

1.2.1 Open Computing Language (OpenCL)

Technologia tak zainicjowana została przez firmę Apple. Do inicjatywy i rozwijania tej technologii włączyły się w późniejszym czasie inne firmy takie jak: AMD, IBM, Intel, NVIDIA. W roku 2008 sformowana została grupa Khronos skupiająca powyższe firmy oraz wiele innych należących do branży IT. Grupa ta czuwa nad rozwojem technologii OpenCL. Technologia tak pozwala na pisanie kodu który jest przenośny między wieloma platformami: komputery, urządzenia przenośne, klastry obliczeniowe. OpenCL pozwala rozpraszać obliczenia na jednostki procesorowe CPU oraz na architektury graficzne GPU. Bardzo ważną zaletą OpenCL jest to, że pisanie z użyciem tej technologii nie jest zależne od sprzętu na jakim będzie ona uruchamiana. Model Platformy OpenCL znajduje się poniżej: (screen z platformy OpenCL)

1.2.2 ATI Stream Computing

Technologia ta została stworzona przez firmę AMD. Za pomocą tej platformy jesteśmy w stanie przeprowadzać złożone obliczenia na sprzęcie produkowanym przez AMD. W skład całego pakietu ATI Stream Computing wchodzi autorski język ATI Brook+ i kompilator tegoż języka. Dodatkowo ATI wspiera developerów własną biblioteka matematyczna (AMD Core Math Library) oraz narzędziami do profilowania wydajności kodu (Strem Kernel Analyzer). Model Platformy OpenCL znajduje się poniżej: (screen z platformy ATI Stream Computing)

1.2.3 NVIDIA CUDA

CUDA (Compute Unified Device Architecture) jest technologia opracowaną przez firmę NVIDIA. Swoje początki CUDA miała w 2007 roku i do dziś jest wiodącą technologią strumieniowego przetwarzania danych z wykorzystaniem układów graficznych GPU.

Rozdział 2

Cele pracy

2.1 Opracowanie techniki zrównoleglenia i przyspieszenia metody śledzenia promieni przy użyciu NVIDIA CUDA

Celem niniejszej pracy jest przeniesienie a zarazem zrównoleglenie algorytmu śledzenia promieni na procesory graficzne (GPU) firmy NVIDIA. Celem także jest przyśpieszenie obliczeń standardowego wstecznego Raytracingu w celu jak najszybszego generowania obrazów scen 3D.

2.2 Projekt uniwersalnej aplikacji - benchmark

W ramach projektu napisany został uniwersalny system Raytracingu działający na wielo rdzeniowych procesorach komputerowych (CPU), a także na kartach graficznych (GPU) firmy NVIDIA które obsługują technologie NVIDIA CUDA. Aplikacja testowa jest benchmarkiem, który jest w stanie przetestować zadane sceny 3D na wielu różnych konfiguracjach sprzętowych. Aplikacja ma za zadanie po uruchomieniu na komputerze użytkownika, testować wszelkie sceny z odpowiedniego katalogu. Dodatkowo zbierać potrzebne informacje o sprzęcie użytkownika oraz czasy generowania obrazów z każdej ze scen. Po przeprowadzeniu wszelkich testów aplikacja jest w stanie wyslać na adres e-mail developera (w tym przypadku autora pracy) wszelkie zgromadzone dane.

Rozdział 3

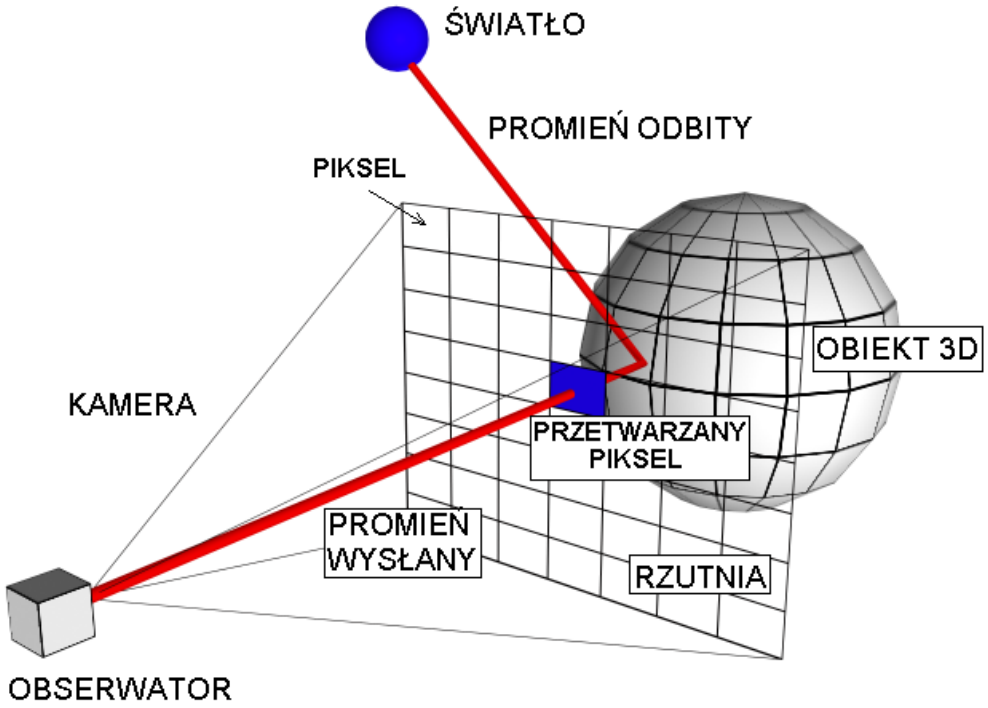
Wprowadzenie do Raytracingu

3.1 Wstępny opis

W rzeczywistym świecie promienie świetlne rozchodzą się od źródła światła do obiektów znajdujących się w świecie. Każde źródło światła wysyła nieskończoną liczbę swoich promieni świetlnych. Następnie te promienie odbijając się od obiektów i trafiają do oczu obserwatora powodując że widzi on określony kolor danego obiektu. Gdyby zaadaptować tą metodę do generowania realistycznej grafiki komputerowej, otrzymalibyśmy nieskończenie dokładny i realistyczny obraz. Z racji jednak na to, że sprzęt komputerowy ma ograniczone możliwości, a metoda ta jest bardzo nie efektywną metodą pod względem obliczeniowym. Najszerzej stosowaną metodą śledzenia promieni jest wsteczne śledzenie promieni (backward raytracing). W odróżnieniu od postępowego algorytmu śledzenia promieni (forward raytrcing), które opiera się na generowaniu jak największej liczby promieni dla każdego źródła światła. Algorytm wstecznego śledzenia promieni zakłada, że promienie śledzone są od obserwatora, poprzez scenę do obiektów z którymi kolidują. Poniżej przedstawiony jest poglądowy rysunek śledzenia pojedynczego promienia od obserwatora poprzez określony pixel na ekranie: 3.1

3.2 Rekursywna metoda śledzenia promieni

Przy omawianiu wstecznej metody śledzenia promieni warto wspomnieć o raytracingu rekursywnym. W zagadnieniu tym bada się rekurencyjnie promienie odbite zwierciadlane oraz załamane, które powstały z kolizji promieni pierwotnych z obiektami na scenie. Tak więc żywotność promienia pierwot-

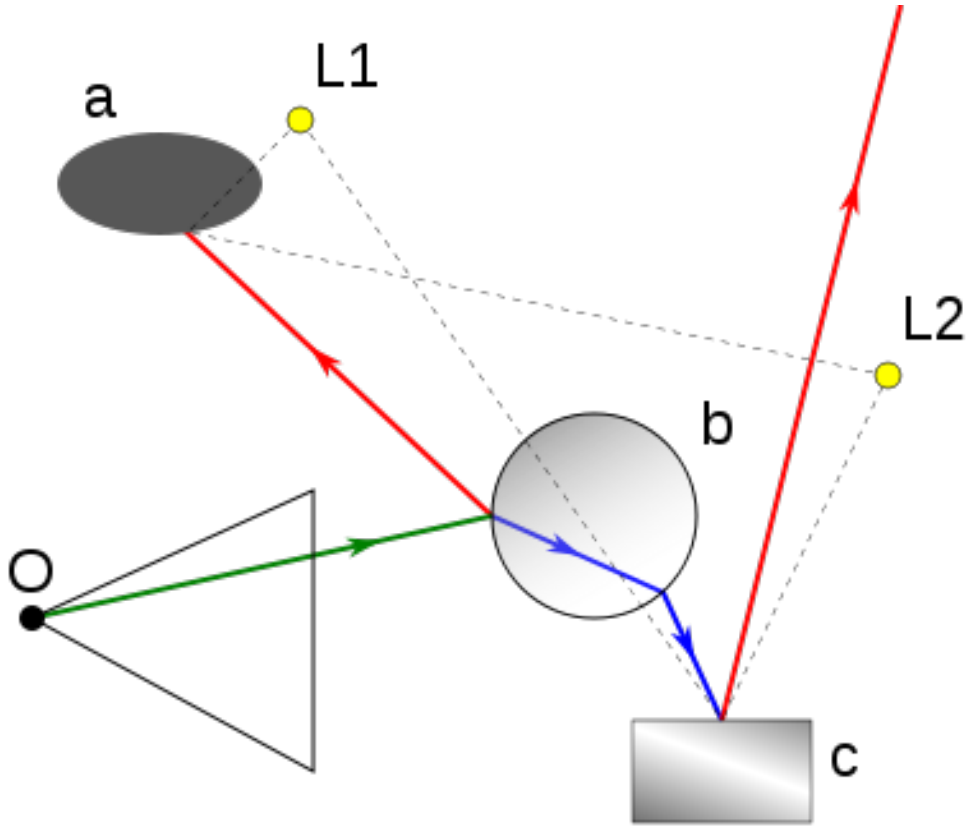


Rysunek 3.1: Sposób określania barwy piksela w raytracigu

nego wcale nie kończy się w momencie kolizji z obiektem sceny. To czy z danego promienia pierwotnego wygenerowane zostaną kolejne promienie w bardzo dużej mierze zależy od materiału jakim pokryty jest dany obiekt sceny. Z pomocą tego rekursywnej metody śledzenia promieni jesteśmy w stanie zasymulować obiekty lustrzane oraz obiekty półprzezroczyste. Rekurencja w tej metodzie trwa do osiągnięcia maksymalnego stopnia zagłębienia. Kolor wynikowy danego pojedynczego Pixela powstaje z sumy kolorów, obiektu w jaki trafił promień pierwotny oraz kolorów obiektów w jakie trafiły promienie pierwotne. Poniżej przedstawiony jest poglądowy schemat zasady działania rekursywnej metody śledzenia promieni: 3.2

3.3 Przedstawienie algorytmu śledzenia promieni

Śledzenie promieni przez scenę rozpoczyna się od obserwatora określanego często jako kamery występującej na scenie. Przez każdy pixel ekranu śledzone są promienie które poruszają się po scenie. Gdy któryś ze śledzonych promieni napotka obiekt i zacznie z nim kolidować.



Rysunek 3.2: Zasada działania rekursywnego algorytmu ray tracingu

3.4 Przykład szósty

Oto przykładowy wydruk:

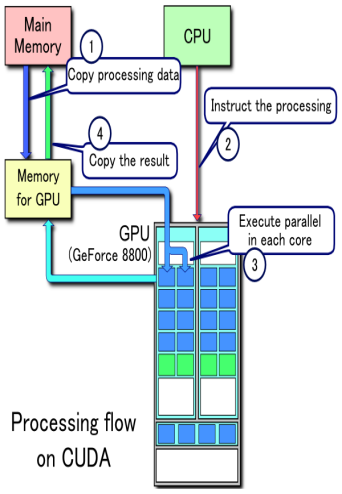
```
1 /* ta funkcja oblicza a+b */
2 int sum(int a, int b)
3 {
4   int suma=0;
6   suma=a+b;
8   return suma;
9 }
```

Rozdział 4

NVIDIA CUDA jako znakomita platforma do zrównoleglenia obliczeń

4.1 Wstępny opis

CUDA(Compute Unified Device Architecture) jest dość nową technologią wprowadzoną na rynek przez firmę NVIDIA. Technologia ta swój początek miała w 2007 roku. Od samego początku stała się ona wiodącą technologią przetwarzania strumieniowego z wykorzystaniem GPU. CUDA jako, że jest technologią stworzoną przez firmę NVIDIA, wspierana jest przez układy graficzne właśnie tej firmy. Wsparcie dla tej technologii rozpoczęło się od układów graficznych serii GeForce 8, Quadro oraz Tesla. Seria układów graficzny Quadro oraz Tesla są wyspecjalizowanymi układami obliczeniowymi do zastosowań naukowych. Natomiast serie GeForce można spotkać na co dzień w komputerach stacjonarnych oraz laptopach. Z pomocą technologii CUDA jesteśmy wstanie uzyskać wielokrotne przyśpieszenie w obliczeniach w stosunku do obliczeń na zwykłym procesorze CPU. Poniżej przedstawiony został przykładowy schemat przepływu obliczeń w CUDA: 4.1



Rysunek 4.1: Przykład przepływu przetwarzania w technologii CUDA.

4.2 Wspierane karty oraz zdolność obliczeniowa

We wstępnym opisie powiedziane było, że technologia CUDA zapoczątkowana była w układach graficznych serii GeForce, Tesla oraz Quadro. Poniżej przedstawiona została tabela ukazująca oficjalne wsparcie określonej wersji CUDA w poszczególnych układach graficznych: 4.1

Zdolność obliczeniowa (wersja)	GPUs	Cards
1.0	G80	GeForce 8800GTX/Ultra/GTS, Tesla C/D/S870, FX4/5600, 360M
1.1	G86, G84, G98, G96, G96b, G94, G94b, G92, G92b	GeForce 8400GS/GT, 8600GT/GTS, 8800GT, 9600GT/GSO, 9800GT/GTX/GX2, GTS 250, GT 120/30, FX 4/570, 3/580, 17/18/3700, 4700x2, 1xxM, 32/370M, 3/5/770M, 16/17/27/28/36/37/3800M, NVS420/50
1.2	GT218, GT216, GT215	GeForce 210, GT 220/40, FX380 LP, 1800M, 370/380M, NVS 2/3100M
1.3	GT200, GT200b	GTX 260/75/80/85, 295, Tesla C/M1060, S1070, CX, FX 3/4/5800
2.0	GF100, GF110	GTX 465, 470/80, Tesla C2050/70, S/M2050/70, Quadro 600,4/5/6000, Plex7000, 500M, GTX570, GTX580
2.1	GF108, GF106, GF104	GT 420/30/40, GTS 450, GTX 460

Tabela 4.1: Zestawienie kart graficznych oficjalnie wspierających technologię CUDA.

Kolejną ważną rzeczą wyróżniająca karty graficzne jest ich zdolność obliczeniowa (ang. compute capability). Identyfikuje ona możliwości obliczeniowe danej karty graficznej w odniesieniu do technologii NVIDIA CUDA. Poniżej przedstawiona została tabela ukazująca możliwości kart graficznych w zależności od profilu CUDA: 4.2

Zdolność obliczeniowa	1.0	1.1	1.2	1.3
Funkcje atomowe w pamięci globalnej	-	✓	✓	✓
Funkcje atomowe w pamięci współdzielonej	-	-	✓	✓
Ilość rejestrów na multiprocesor	8192	8192	16384	16384
Maksymalna liczba warpów na multiprocesor	24	24	32	32
Maksymalna liczba aktywnych wątków na multiprocesor	768	768	1024	1024
Podwójna precyzja	-	-	-	✓

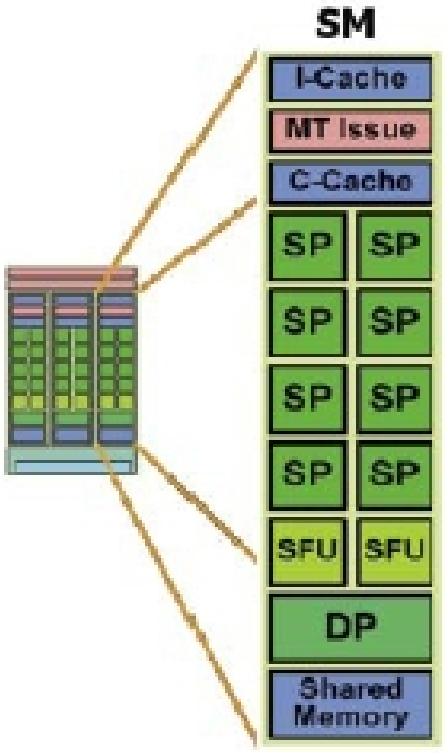
Tabela 4.2: Porównanie zdolności obliczeniowych kart graficznych wspierających NVIDIA CUDA.

4.3 Architektura

Karty graficzne GPU znacznie różnią się wydajnością od zwykłych procesorów CPU. Różnica w wydajności wynika głównie z faktu, iż procesory graficzne specjalizują się w równoległych, wysoce intensywnych obliczeniach. Karty graficzne składają się z większej liczby tranzystorów które są odpowiedzialne za obliczenia na danych. Nie posiadają natomiast takiej kontroli przepływu instrukcji oraz jednostek odpowiedzialnych za buforowanie danych jak procesory komputerowe CPU. Układy graficzne wspierające technologię CUDA zbudowane z multiprocesorów strumieniowych (ang. stream multiprocessor). Różne modele kart graficznych firmy NVIDIA posiadają różną liczne multiprocesorów, co przekłada się także na wydajność i zdolnosć obliczeniową danej architektury. Na rysunku 4.2 Przedstawiona jest przykładowa budowa takiego właśnie multiprocesora.

Każdy z multiprocesorów składa się z: (napisac z kad to wziete!!!!!)

- I-Cache - bufor instrukcji;
- MT Issue - jednostka która rozdziela zadania dla SP i SFU
- C-Cache -bufor stałych (ang. constant memory) o wielkości 8KB, który przyspiesza odczyt z obszaru pamięci stałej
- 8 x SP - 8 jednostek obliczeniowych tzw stream processors, które wykonują większość obliczeń pojedynczej precyzji (każdy zawiera własne 32-bitowe rejestry)



Rysunek 4.2: Przykładowy schemat multiprocesora strumieniowego.

- 2 x SFU - jednostki specjalne (ang. special function units). Zadaniem ich jest obliczanie funkcji przestępnych, np. trygonometrycznych, wykładniczych i logarytmicznych, czy interpolacja parametrów.
- DP -procesor, który wykonuje obliczenia podwójnej precyzji
- SM - pamięć współdzielona (ang. shared memory) o wielkości 16KB.

4.4 Rodzaje pamięci w architekturze CUDA

- Pamięć globalna (ang. global memory) - Ta pamięć jest dostępna dla wszystkich wątków. Nie jest pamięcią buforowaną. Dostęp do niej trwa od około 400 do 600 cykli. Pamięć ta służy przede wszystkim do zapisuj wyników działań programu obliczeniowego.
- Pamięć lokalna (ang. local memory) - Ma taki sam czas dostępu jak pamięć globalna (400-600 cykli). Nie jest także pamięcią buforowaną.

Jest ona zdefiniowana dla danego wątku. Każdy wątek CUDA posiada własną pamięć lokalną. Zajmuje się ona przechowywaniem bardzo dużych struktur danych. Pamięć ta jest najczęściej używana gdy obliczenia danego wątku nie mogą być w całości wykonane na dostępnych rejestrach procesora graficznego.

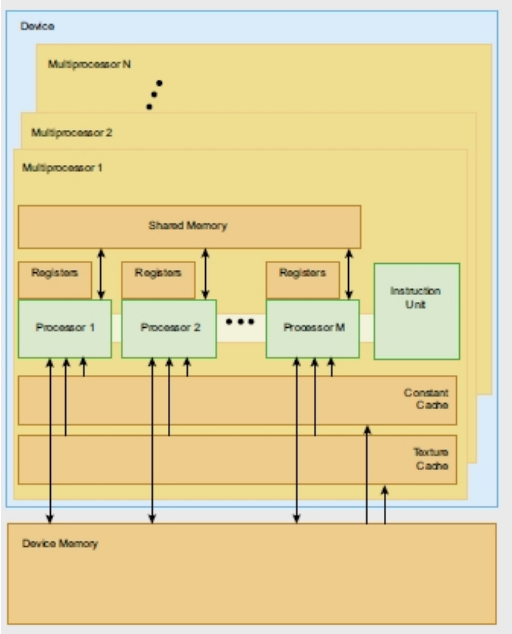
- Pamięć współdzielona (ang. shared memory) - Jest to bardzo szybki rodzaj pamięci, dorównujący szybkości rejestrów procesora graficznego. Przy pomocy tej pamięci, wątki przydzielone do jednego bloku są wstanie się ze sobą komunikować. Należy jednak obchodzić się ostrożnie z tym rodzajem pamięci, gdyż mogą powstać Momoty w których wątki w jednym bloku będą chciały jednocześnie zapisywać i odczytywać z tej pamięci. Występowanie takich konfliktów w odczycie i zapisie powoduje duże opóźnienia.

- Pamięć stała (ang. const memory) - Ta pamięć w odróżnieniu do powyższych rodzajów pamięci, jest buforowaną pamięcią tylko do odczytu. Gdy potrzebne dane znajdują się aktualnie w buforze dostęp do nich jest bardzo szybki. Czas dostępu rośnie gdy danych nie ma w buforze i muszą być doczytane z pamięci karty.

- Pamięć Tekstur (ang. texture memory) - Jest pamięcią podobną do pamięci stałej gdyż udostępnia tylko odczyt danych. Jest także pamięcią buforowaną. W pamięci tej bufor danych został zoptymalizowany pod kątek odczytu danych z bliskich sobie adresów. Najkorzystniejszą sytuacją jest gdy wątki dla danego warpa (grupa 32 wątków zarządzanych przez pojedynczy multiprocesor) odczytują adresy, które znajdują się blisko siebie. CUDA w swojej implementacji udostępnia możliwość posługiwania się teksturami 1D,2D,3D.

- Rejestry - Jest to najszybszy rodzaj pamięci. Dostęp do niego nie powoduje żadnych dodatkowych opóźnień, chyba że próbujemy odczytać z rejestru do którego dopiero co zostało coś zapisane. Każdy multiprocesor w urządzeniu CUDA posiada 8192 lub 16384 rejestrów 32-bitowych. Zależy to od wersji(zdolności obliczeniowej) danego urządzenia. W celu uniknięcia powyższych konfliktów ilość wątków na pojedynczy multiprocesor ustawia się jako wielokrotność liczby 64. NAPISAC Z KAD TO WIEM!!!!!!!!!!!!1

Na obrazku 4.2 poniżej przedstawiony został poglądowy schemat pamięci w architekturze CUDA.



Rysunek 4.3: Schemat pamięci.

4.5 Przykładowy kod pod architekturę CUDA

```
1  #include <stdlib.h>
2  #include <cuda_runtime.h>
3  #include <cutil.h>
4  // definicja funkcji obliczeniowej (wykorzystanie słowa kluczowego __global__)
5  __global__ void testKernel(float *data)
6  {
7  int id = blockIdx.x * blockDim.x + threadIdx.x; // oblicz indeks w tablicy
8  data[id] = sqrt(data[id]); // oblicz funkcję sqrt() dla danego elementu tablicy
9  }
10 int main()
11 {
12 cudaSetDevice(0); // inicjalizacja urządzenia CUDA
13 float *hData = (float*)malloc(sizeof(float) * ARRAY_SIZE);
14 float *dData;
15 cudaMalloc((void**)&dData, sizeof(float) * ARRAY_SIZE); // alokacja pamięci CUD
16 InitializeData(hData); // inicjalizacja tablicy hData
17 // skopiowanie tablicy hData do pamięci CUDA reprezentowanej przez dData
18 cudaMemcpy(dData, hData, sizeof(float) * ARRAY_SIZE, cudaMemcpyHostToDevice)
19 dim3 dimBlock(BLOCK_SIZE, 1); // ustalenie wielkości bloku 1D
20 dim3 dimGrid(ARRAY_SIZE / dimBlock.x, 1); // ustalenie wielkości kraty 1D
21 testKernel<<<dimGrid, dimBlock>>>(dData); // wywołanie funkcji obliczeniowej
22 // skopiowanie wynikowej zawartości tablicy dData do hData
```

```
23 cudaMemcpy(dData, hData, sizeof(float) * ARRAY_SIZE, cudaMemcpyDeviceToHost);
24 return 0;
25 }
```

Bibliografia

[1] J. Bloch. Effective Java. Addison–Wesley, Stoughton, USA, second edition, 2008. http://www.thinkingblackberry.com/.

[2] E. Ciurana. Developing with Google App Engine. Apress, Berkeley, USA, 2008.

[3] Community. BlackBerry Developer Zone. http://na.blackberry.com/eng/developers/.

[4] S. Hartwig and M. Buchmann. Empty Seats Traveling. http://research.nokia.com/files/NRC-TR-2007-003.pdf.

[5] C. King. Advanced BlackBerry Development. Apress, Berkeley, USA, 2009.

[6] A. Rizk. Thinking BlackBerry. http://www.thinkingblackberry.com/.

[7] D. Sanderson. Programming Google App Engine. O'Reilly Media, Sebastopol, USA, 2009.

[8] Charles Severance. Using Google App Engine. O'Reilly Media, Sebastopol, USA, 2009.

[9] Wikipedia. Stosunek liczby samochodów do zaludnienia. http://pl.wikipedia.org/wiki/Stosunek˙liczby˙samochod%C3%B3w˙do˙zaludnienia.

Listings

Spis rysunków

Spis tabel