UMCS

UNIWERSYTET MARII CURIE-SKŁODOWSKIEJ
W LUBLINIE
Wydział Matematyki, Fizyki i Informatyki

Kierunek: **informatyka**

Emil Szerafin
nr albumu: 307490

Wykorzystanie GPU do akceleracji generowania i przetwarzania dźwięku

Use of GPU in acceleration of
sound generation and processing

Praca licencjacka
napisana w Katedrze Oprogramowania Systemów Informatycznych
pod kierunkiem **dr hab. Przemysława Stpiczyńskiego**

Lublin 2022

Spis treści

Wstęp

Przetwarzanie sygnałów dźwiękowych jest dużym obszarem informatyki. Wraz z jego rozwojem, powstawaniem nowych algorytmów, wzrostem standardów dotyczących jakości dźwięku, a także zwiększaniem się ilości przetwarzanych danych, pojawia się coraz większe zapotrzebowanie na moc obliczeniową. W wielu przypadkach, przy zastosowaniu obecnych technik przetwarzania dźwięku, obliczenia w czasie rzeczywistym, stają się nieosiągalne, co mocno utrudnia pracę wielu osobom, a w niektórych przypadkach sprawia, że staje się ona nie możliwa. Wynika z tego, iż ciągły rozwój, a co za tym idzie, ciągły wzrost wymagań oprogramowania, wymusza na użytkownikach okresową wymianę podzespołów na coraz to wydajniejsze. Powstaje pytanie, czy nie istnieje żadne inne rozwiązanie, które pozwoliło by przynajmniej częściowo zniwelować ten problem. W obecny standard komputerów osobistych włączona jest obecność karty graficznej - odmiany procesora, przeznaczonego przede wszystkim do przetwarzania i generowania obrazu. Wraz z rozwojem tej technologii zaczęto zauważać, że karty graficzne posiadają ogromny potencjał obliczeniowy, który można by było wykorzystać w innych dziedzinach informatyki. Wraz z pojawieniem się technologii takich jak CUDA czy OpenCL, rozpowszechniono wykorzystanie karty graficznej do obliczeń ogólnego przeznaczenia. Dziedzina przetwarzania sygnałów dźwiękowych, ze względu na swoją naturę, jest jedną z dziedzin informatyki, która mogłaby skorzystać na wykorzystaniu tych technologii w celu przyspieszenia obliczeń i tym samym umożliwienia szerszemu gronu osób na korzystanie z bardziej zaawansowanych technik przetwarzania dźwięku.

Cel i zakres pracy

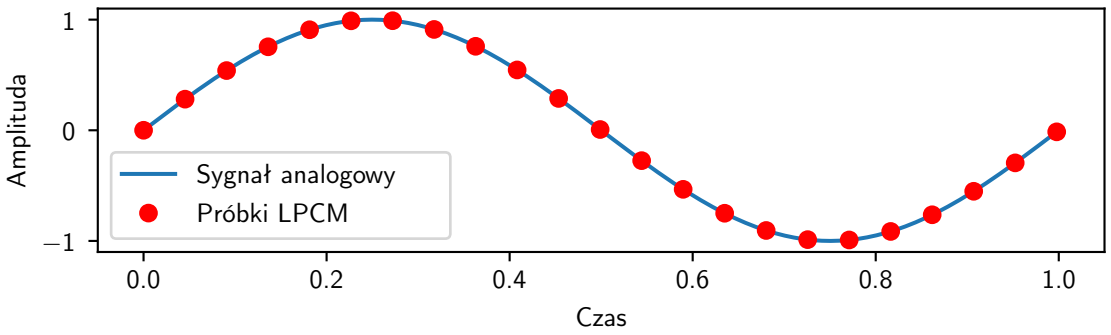
Celem niniejszej pracy jest zaprojektowanie oraz implementacja systemu, pozwalającego na przetestowanie możliwości wykorzystania karty graficznej względem CPU, do przetwarzania sygnałów dźwiękowych. System ma umożliwiać syntezowanie oraz przetwarzanie dźwięku w różnych powszechnie stosowanych formatach (biorąc pod uwagę częstotliwość próbkowania oraz wielkość próbki) zarówno w czasie rzeczywistym, jak i w trybie offline. Tryb offline pozwala na syntezę/przetworzenie sygnału w możliwie najkrótszym dla urządzenia czasie. Na ogół pozwala to na wykonanie skomplikowanych obliczeń, nawet w sytuacji kiedy nie jest możliwe przetworzenie danych w czasie rzeczywistym. W tym przypadku zostanie to wykorzystane w celu porównania czasów wykonania, co nie było by możliwe dla czasu rzeczywistego. System musi udostępniać możliwość gromadzenia danych statystycznych dotyczących wydajności swojej pracy, oraz umożliwiać na przeprowadzanie jednolitych testów wydajnościowych. W swojej uniwersalności powinien pozwalać na wprowadzanie modyfikacji w nieskomplikowany sposób, co umożliwi ekstensywne testowanie algorytmów oraz łączenie ich w bardziej złożonych scenariuszach użycia, jak również zmianę wykorzystanej platformy bez konieczności modyfikacji fundamentów jego działania. Skutkuje to eliminacją możliwie największej liczby czynników zewnętrznych, mogących zaburzyć pomiary wydajności. System powinien przetwarzać dźwięk w formacie PCM, który jest najbardziej powszechnie stosowanym formatem w przemyśle muzycznym oraz być w stanie odczytywać podstawowe sygnały zawarte w MIDI. Biorąc to wszystko pod uwagę, powstały system będzie stanowić podstawę podczas porównania możliwości CPU i GPU dla przedstawionego problemu. Praca ta nie skupia się nadto na optymalizacji powstałych rozwiązań, a na próbie weryfikacji omawianej koncepcji.

Rozdział 1

Cyfrowa reprezentacja dźwięku

1.1 Format LPCM

PCM - pulse-code modulation (modulacja impulsowa) jest popularną metodą reprezentacji sygnałów analogowych w postaci cyfrowej. Polega na próbkowaniu sygnału analogowego w regularnych odstępach czasu, a następnie kwantyzacji wartości próbek. Wartości mogą zostać reprezentowane przy użyciu szerokiej gamy metod. LPCM - linear pulse-code modulation (liniowa modulacja impulsowa) jest jednym z najprostszych sposobów reprezentacji zkwantyfikowanego sygnału. Wykorzystuje on skale liniową, w połączeniu ze stałym punktem odniesienia, względem którego mierzona jest próbka. Reprezentacja wyniku procesu konwersji pomiędzy sygnałem analogowym a sygnałem PCM przedstawia Rysunek 1.1.



Rysunek 1.1: Próbkowanie LPCM FLOAT32

1.2 Częstotliwość próbkowania

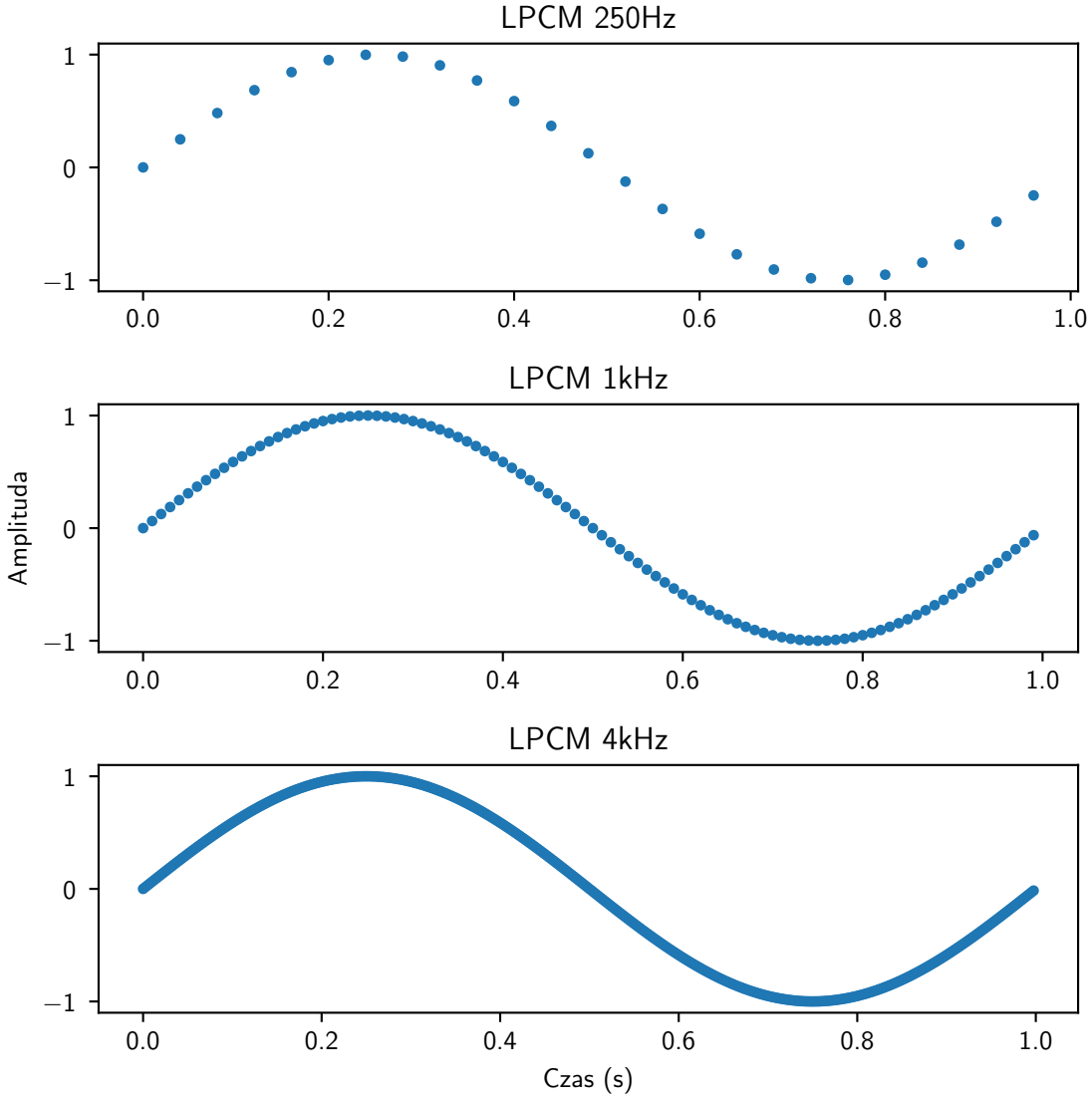
Jakość reprezentacji sygnału zależy do dwóch czynników: częstotliwości próbkowania oraz rozdzielczości pojedynczej próbki. Częstotliwość próbkowania ogranicza odstęp pomiędzy poszczególnymi próbkami. Wyższa częstotliwość próbkowania zmniejsza odstęp pomiędzy próbkami, co pozwala na dokładniejsze odwzorowanie sygnału w dziedzinie czasu, a jednocześnie zwiększa ilość danych, które muszą zostać później przetworzone. Konkretną wartość potrzebną do reprezentacji określonego sygnału można obliczyć przy użyciu twierdzenia Nyquista-Shannona. Głosi ono, iż aby zapisać sygnał o częstotliwości f Hz, należy próbkować go co najmniej z częstotliwością $2f$ Hz. W praktyce jednak, aby uniknąć problemów związanych z aliasingiem, zaleca się próbkowanie z częstotliwością co najmniej $2.2f$ Hz. W przypadku słuchu ludzkiego najwyższy słyszalny dźwięk nie będzie przekraczać 20 kHz. Tak więc, aby zapisać sygnał o takiej częstotliwości, należy próbkować go co najmniej z częstotliwością 44kHz, co pokrywa się z ogólno przyjętym standardem standardem stosowanym w przemyśle muzycznym. Przykład różnicy w częstotliwości próbkowania sygnału przedstawia Rysunek 1.2.

1.3 Rozdzielczość próbki

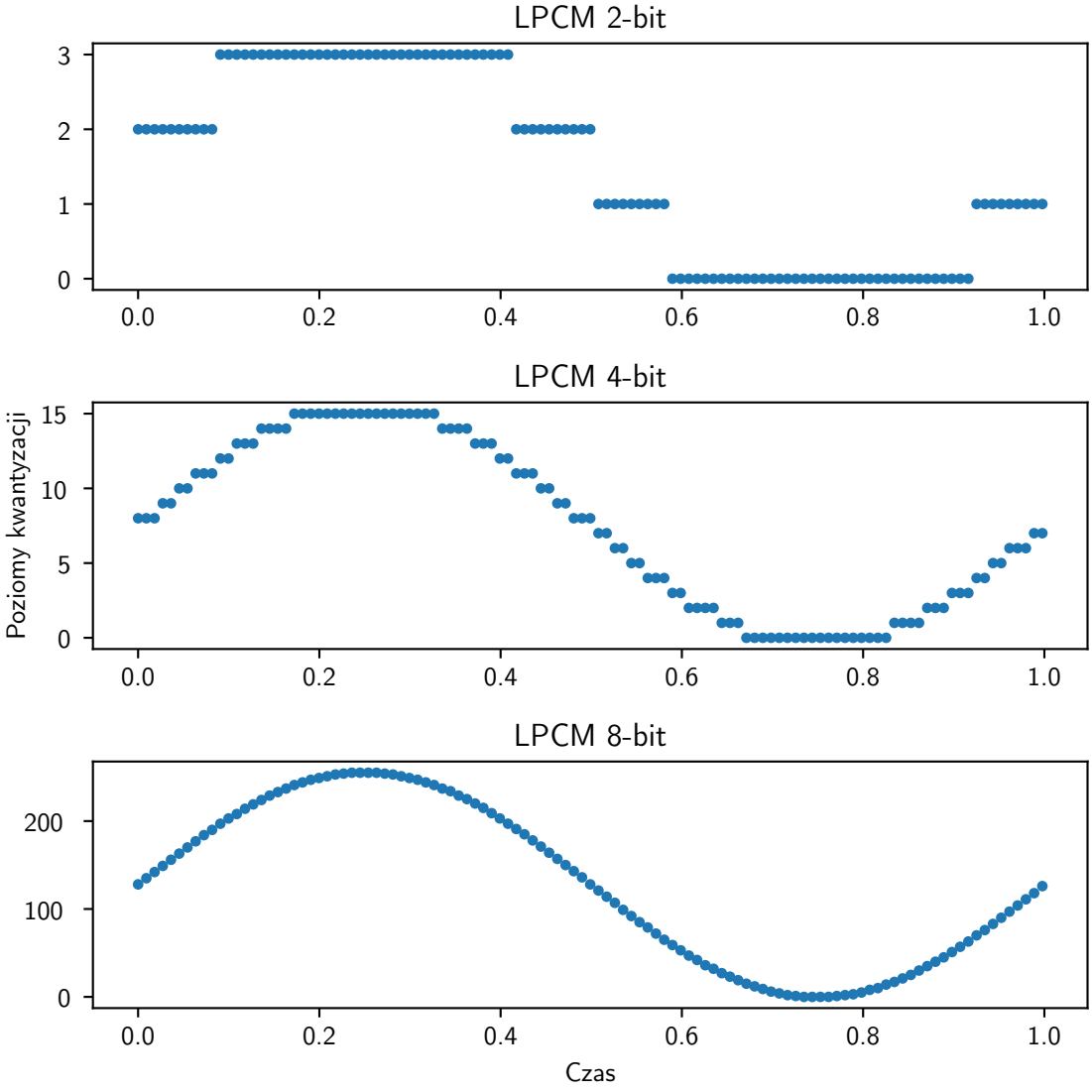
Rozdzielczość próbki, zwana również głębią bitowa / głębią próbki, określa ilość bitów, które są wykorzystywane do jej zapisu. Im większa rozdzielczość, tym mniejszy błąd kwantyzacji oraz większy rozmiar danych do przetworzenia. Determinuje to ilość poziomów kwantyzacji sygnału, która jest równa 2^n, gdzie n to ilość bitów wykorzystywanych do zapisu próbki. W przypadku sygnałów audio najpopularniejszymi wartościami dla głębi są: 16-bit, 24-bit oraz 32-bit. W zależności od sytuacji, wartości te pozwalają na rzetelny zapis sygnału. Wartość ta przeważnie nie posiada bezpośredniego wpływu na czas obliczeń, a jedynie może zwiększyć ilość pamięci koniecznej do zapisu sygnału. Przykład różnicy w rozdzielczości próbki przedstawia Rysunek 1.3.

1.4 Rozmiar bufora

W przeciwieństwie do metod przetwarzania sygnału w sposób analogowy, gdzie sygnał jest ciągły, w przypadku sygnałów cyfrowych, sygnał jest dzielony na porcje o określonej wielkości. Wielkość tej porcji (zazwyczaj jest to wielokrotność liczby 2) określa rozmiar bufora. Bufory pozwalają na przetworzenie większej ilości próbek jednocześnie, co pozwala na zwiększenie wydajności przetwarzania sygnału. Największe znaczenie ma to w przypadku przetwarzania sygnału w czasie rzeczywistym, gdzie każda próbka musi



Rysunek 1.2: Częstotliwość próbkowania LPCM



Rysunek 1.3: Rozdzielczość próbki LPCM

Tabela 1.1: Rozmiar bufora względem czasu przetworzenia

rozmiar bufora	44kHz	48kHz	96kHz	192kHz
32	0.73ms	0.67ms	0.33ms	0.17ms
64	1.45ms	1.33ms	0.67ms	0.33ms
128	2.91ms	2.67ms	1.33ms	0.67ms
256	5.82ms	5.33ms	2.67ms	1.33ms
512	11.64ms	10.67ms	5.33ms	2.67ms
1024	23.27ms	21.33ms	10.67ms	5.33ms
2048	46.55ms	42.67ms	21.33ms	10.67ms
4096	93.09ms	85.33ms	42.67ms	21.33ms
8192	186.18ms	170.67ms	85.33ms	42.67ms

zostać przetworzona w określonym czasie. Im większy rozmiar bufora, tym więcej czasu zostaje przeznaczone na jego przetworzenie. Wartość ta jest zależna od zastosowania, a także od dostępnych zasobów sprzętowych. W wielu przypadkach pożądany rozmiar bufora jest możliwie najmniejszy, tak aby zminimalizować opóźnienie w przetwarzaniu sygnału. Opóźnienie jest szczególnie niepożądanym efektem ubocznym na przykład w sytuacji w której muzyk potrzebuje słyszeć na bieżąco efekt swojego występu. Nie zawsze jest to jednak możliwe ze względu na ograniczenia sprzętowe. Krótszy czas przetwarzania pozostawia mniejszy margines błędu. Chwilowa niedostępność zasobu sprzętowego może spowodować, że próbka nie zostanie przetworzona w określonym czasie, co prowadzi do powstania niechcianych artefaktów w sygnale. Ilość czasu na przetworzenie bufora można wyliczyć na podstawie wzoru: $t = \frac{N}{f}$, gdzie t to czas przetworzenia bufora, N to ilość próbek w buforze, a f to częstotliwość próbkowania. Przykład zależności czasu przetworzenia od wielkości bufora oraz częstotliwości próbkowania przedstawia Tabela 1.1.

1.5 Ilość kanałów

Ilość kanałów określa ilość niezależnych ścieżek dźwiękowych, które mogą reprezentować sygnał w przestrzeni. Najpopularniejszymi wartościami są 1 (mono) oraz 2 (stereo). O ile istnieje więcej standardów reprezentacji sygnału, to nie są one często wykorzystywane. W przetwarzania sygnału zazwyczaj pracuje się używając sygnałów monofonicznych ostatecznie dążąc do uzyskania sygnału stereofonicznego. Wartość ta ma wpływ na ilość danych, które muszą zostać przetworzone, a także nierzadko na ich interpretację, co objawia się w postaci stosowania różnych algorytmów w zależności od

Tabela 1.2: Przesył danych dla różnych formatów przy wykorzystaniu float32

częstotliwość próbkowania	mono	stereo
44kHz	1.76KB/s	3.52KB/s
48kHz	1.92KB/s	3.84KB/s
96kHz	3.84KB/s	7.68KB/s
192kHz	7.68KB/s	15.36KB/s

ilości kanałów.

1.6 Wpływ formatu na czas przetwarzania

Format, w jakim przetwarzany jest sygnał, ma wpływ na jego objętość, a co za tym idzie na czas przetwarzania. Przedstawione powyżej wartości w różnym stopniu przyczyniają się do zwiększenia objętości danych. Najbardziej negatywny wpływ na prędkość obliczeń ma częstotliwość próbkowania. Jest to właściwość uznawana za najważniejszą w kontekście jakości sygnału, a jednocześnie jej zwiększenie pozytywnie wpływa na wielkość opóźnienia (Tabela 1.1). Następną cechą sygnału mającą znaczący wpływ na jego objętość ma ilość kanałów, która to w oczywisty sposób przemnaża liczbę próbek do przetworzenia. Rozdzielczość próbki również ma wpływ na objętość danych, jednak w mniejszym stopniu niż pozostałe cechy. W celu zachowania jakości i zmniejszenia znaczenia błędów kwantyzacji, w trakcie obróbki sygnału najczęściej stosuje się typu float32 i dopiero na sam koniec, w celu zmniejszenia powstałego pliku, konwertuje się sygnał do reprezentacji 24 lub 16 bitowej. Rozmiar bufora nie wpływa na objętość danych. Przeważnie ta wartość służy do uzyskania balansu pomiędzy stabilnością, a wielkością opóźnienia. Nierzadko ma wpływ na wydajność konkretnych algorytmów, jednak nie jest możliwe wyznaczenie stałego współczynnika, bez analizy danego przypadku. Tabela 1.2 przedstawia porównanie potrzebnej przepustowości B/s dla różnych popularnych zestawień parametrów pojedynczego strumienia LPCM. Należy zwrócić uwagę, iż w dziedzinie przetwarzania dźwięku sytuacje w których przetwarza się pojedynczy strumień audio są rzadkością. W większości przypadków przetwarzanych jest wiele strumieni jednocześnie, przy wykorzystaniu sekwencji wielu algorytmów, co zauważalnie zwiększa objętość danych do przetworzenia w raz z potrzebną na to mocą obliczeniową.

1.7 Format MIDI

MIDI - Musical Instrument Digital Interface (Cyfrowy Interfejs Instrumentów Muzycznych) jest standardem komunikacji pomiędzy różnymi urządzeniami muzycznymi, a także służy jako standard zapisu nutowego w postaci cyfrowej. Format ten nie zawiera informacji o dźwięku, a jedynie o sposobie jego syntezy. Dane są zapisane w postaci binarnej, posiadają one zauważalnie mniejsze rozmiary w pamięci w porównaniu z sygnałem PCM. Aby można było odtworzyć dany sygnał w postaci sygnału dźwiękowego, należy go wpierw przetworzyć przy użyciu np. syntezatora. Format MIDI posiada kilka wariantów, rozbudowujących i tak bogaty asortyment możliwości. Nawet w podstawowym wariancie MIDI (General MIDI), pliki MIDI są w stanie przechowywać informacje na różne sposoby. Sposoby te nazywane są formatami. Na rzecz tej prazy w zupełności wystarczającym będzie najprostszy w interpretacji format 0. W tym formacie, wszystkie ścieżki są zapisane w jednym strumieniu danych, zawierającym zdarzenia meta (meta events) oraz zdarzenia MIDI (MIDI events). Zdarzenia meta zawierają między innymi informacje o strukturze pliku, takie jak tempo, nazwa utworu, czy informacja o końcu pliku. Zdarzenia MIDI zawierają przede wszystkim informacje o dźwięku, takie jak: numer kanału, numer dźwięku (głosu), siła uderzenia, czy długość trwania dźwięku. Pojedyncze zdarzenie MIDI przekazują informacje takie jak np.: rozpoczęcie dźwięku, zakończenie dźwięku, zmiana siły uderzenia. Dopiero razem tworzą one pełny obraz informacji o dźwięku, który można syntezować, a następnie odtworzyć.

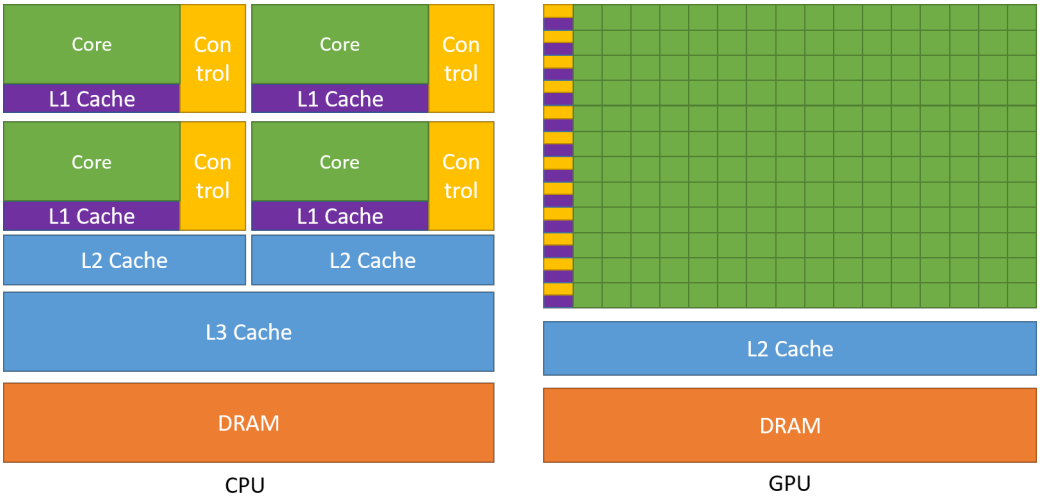
Rozdział 2

Technologia CUDA

CUDA (Compute Unified Device Architecture) to architektura stworzona przez firmę NVIDIA. Pozwala ona na programowanie odpowiednich kart graficznych w sposób umożliwiający wykonywanie obliczeń ogólnego przeznaczenia. Programy napisane z wykorzystaniem technologii CUDA zasadniczo różnią się od standardowych programów, które są wykonywane na zwykłych procesorach. Swoją charakterystyką przypominają one bardziej programy równoległe, które są wykonywane na klastrach obliczeniowych, niż standardowe programy sekwencyjne. CUDA pozwala na wykorzystanie architektury karty graficznej (Rysunek 2.1) w celu wykonania obliczeń w sposób zauważalnie odmienny niż w przypadku procesora.

2.1 Model programowania

W przeciwieństwie do programowania wielowatkowego na CPU, programowanie w technologii CUDA nie polega na zarządzaniu każdym wątkiem oddzielnie, a całym blokiem wątków naraz. Wątki w bloku są grupowane w tzw. *warp*, czyli grupę 32 wątków, które są wykonywane równolegle. Bloki wątków są grupowane w *grid* - siatkę bloków, która jest wykonywana na karcie graficznej. Model programowania w technologii CUDA jest zilustrowany na Rysunku 2.2. Pozwala to wykonywać jednocześnie tę samą operację na wielu elementach danych przy wykorzystaniu rdzeni CUDA. W ten sposób możliwe jest uzyskanie znacznie większej wydajności obliczeń niż w przypadku wykonywania ich sekwencyjnie na procesorze. Architektura GPU narzuca jednak pewne ograniczenia. Możliwość wykonania obliczeń na tysiącach wątków jednocześnie, osiągnięto kosztem możliwości pojedynczego wątku. Wątki na GPU nie mogą komunikować się bezpośrednio między sobą, co w połączeniu z ich ilością, wynosi problem konkurencyjności obliczeń na wyższy poziom abstrakcji niż w przypadku programowania wielowątkowego na CPU. Architektura sprawia również, że operacje warunkowe w kodzie CUDA mogą znacząco

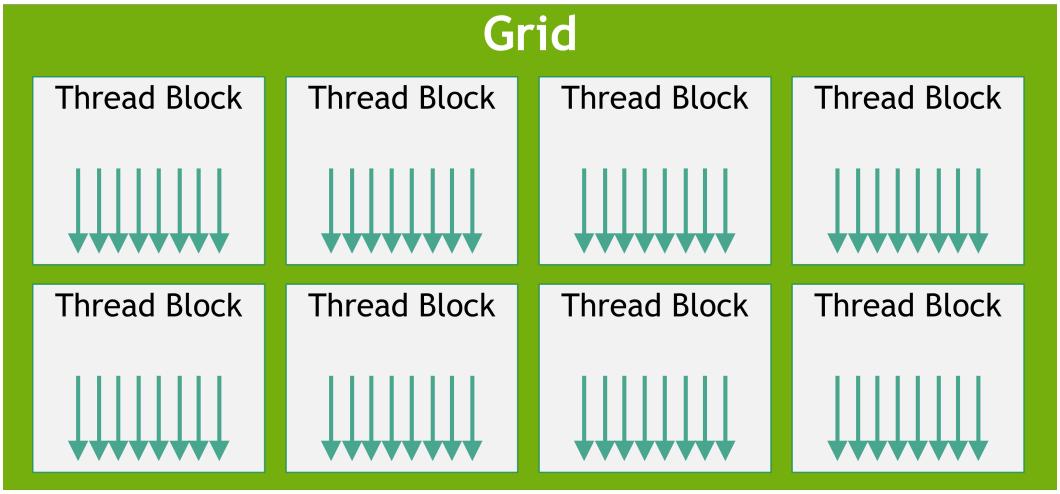


Rysunek 2.1: docs.nvidia.com - "GPU Devotes More Transistors to Data Processing", ilustracja różnicy w architekturze procesora i karty graficznej

obniżyć wydajność urządzenia. W przypadku rozgałęzienia w kodzie CUDA, jeżeli jeden wątek spełni warunek rozgałęzienia, wtedy wszystkie wątki wchodzące w dany warp będą musiały czekać na zakończenie obliczeń przez ten pojedynczy wątek. W pesymistycznym przypadku może to oznaczać, że w danym momencie dysponujemy jedyni 1/32 mocy obliczeniowej karty graficznej.

2.2 Host i Device

Urządzenie GPU można traktować jako oddzielny komputer, posiadający swoje własne podzespoły oraz zasoby. Technologia CUDA właśnie w ten sposób reprezentuje kompatybilną kartę graficzną. Interfejs programistyczny CUDA rozróżnia dwa środowiska: *host* oraz *device*. Host to komputer, na którym uruchamiany jest program, który wykorzystuje technologię CUDA. Device to karta graficzna, na której wykonywane są obliczenia. Najbardziej podstawowym schematem programu wykorzystującego tą technologię jest: przekazanie danych przygotowanych na urządzeniu host do urządzenia device, które wykonuje obliczenia. Po zakończeniu obliczeń, urządzenie host pobiera wyniki z GPU, gdzie możliwa jest ich interpretacja, zapis lub dalsze przetwarzanie. Można zauważyć, iż na moc obliczeniową, wynikającą z połączenia CPU oraz GPU, składa się również szybkość przesyłania danych między tymi dwoma urządzeniami. Tabela 2.1 przedstawia przeciętne możliwości w kategorii kopiowania danych zawartych w konkretnych typach pamięci urządzeń. Należy zauważyć ograniczenia jakie za sobą niesie wykorzystywanie szyny PCIE w przypadku intensywnej komunikacji pomiędzy CPU a GPU. Sugerowanym przez NVIDIA'e rozwiązaniem jest przekazanie jedynie



Rysunek 2.2: docs.nvidia.com - "Grid of Thread Blocks", ilustracja modelu programowania w technologii CUDA

Rysunek 2.3: Przykładowe wywołanie kernela CUDA

niezbędnych danych w celu wykonania obliczeń na GPU, a następnie zwrócenie jedynie samego wyniku obliczeń. W ten sposób można przynajmniej częściowo zniwelować efekt wąskiego gardła, jaki może wynikać z przepustowości szyny PCIE. Alternatywnym rozwiązaniem jest wykonywanie obliczeń asynchronicznie względem przesyłu danych, co pozwala na wykonanie obu operacji w tym samym czasie i zlikwidowanie czasu oczekiwania na zakończenie przesyłu danych.

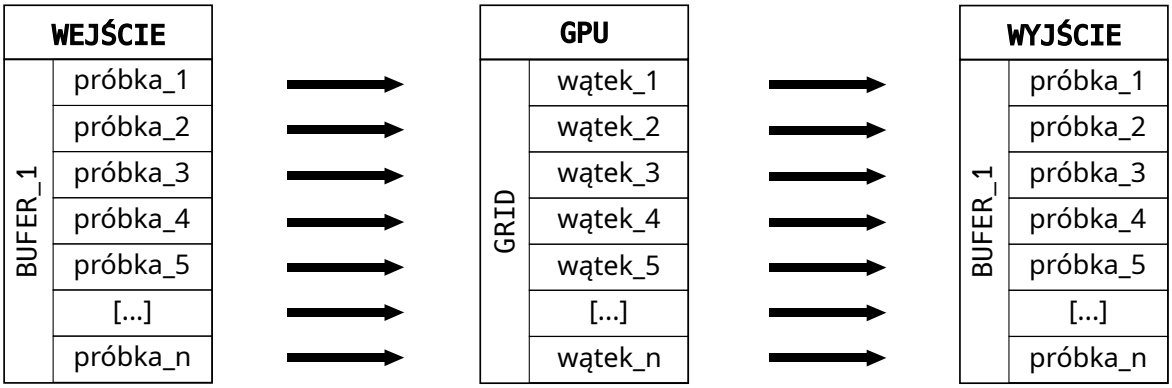
2.3 Zastosowanie dla problematyki przetwarzania sygnałów

Technologia CUDA jest obecnie wykorzystywana do przetwarzania sygnałów cyfrowych. Przy użyciu biblioteki cuFFT możliwe jest wykorzystanie mocy obliczeniowej karty graficznej do przetwarzania sygnałów w dziedzinie częstotliwości. Technologia ta nie jest jednak popularna w przypadku przetwarzania sygnałów dźwiękowych w przypadku w przemyśle rozrywkowym. Nie licząc pojedynczych prób w latach 2000 - 2010, które z powodu małej kompatybilności oprogramowania, zakończyły się niepowodzeniem, nie ma obecnie dostępnych narzędzi, wykorzystujących moc obliczeniową karty graficznej w przemyśle muzycznym. Technologia CUDA oraz karty firmy NVIDIA stały się dużo bardziej rozwinięte i popularne. W związku z tym ponowne podejście do tematu przetwarzania sygnałów dźwiękowych przy użyciu CUDA może przynieść pozytywne rezultaty. Natura popularnego w przemyśle muzycznym

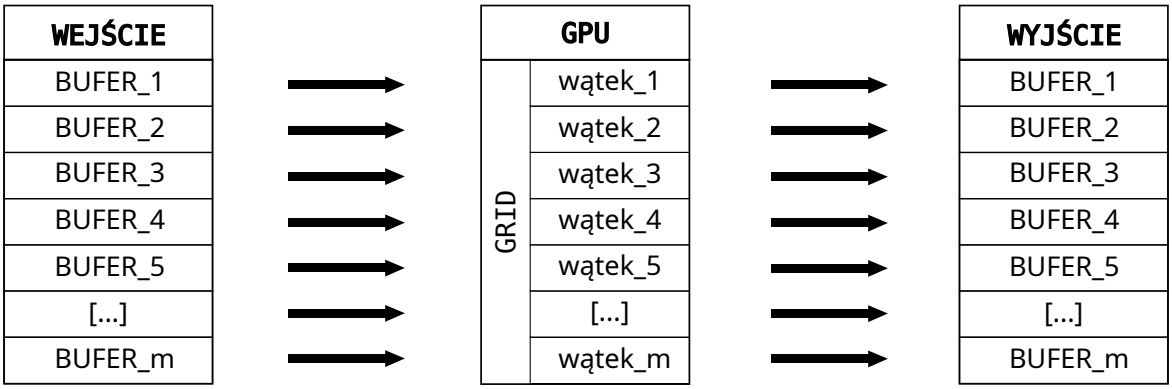
Tabela 2.1: Przeciętna przepustowość przesyłu danych dla danego standardu / urządzenia

urządzenie	typ	przepustowość
CPU	L1 cache	500GB/s - 1TB/s
	L2 cache	200GB/s - 1TB/s
	L3 cache	75GB/s - 400GB/s
RAM	DDR3	10GB/s - 20GB/s
	DDR4	17GB/s - 25GB/s
	DDR5	35GB/s - 50GB/s
GPU	L1 cache	1TB/s - 2TB/s
	L2 cache	500GB/s - 1TB/s
	DRAM	200GB/s - 800GB/s
PCIE	3.0	1GB/s - 16GB/s
	4.0	2GB/s - 32GB/s
	5.0	4GB/s - 64GB/s

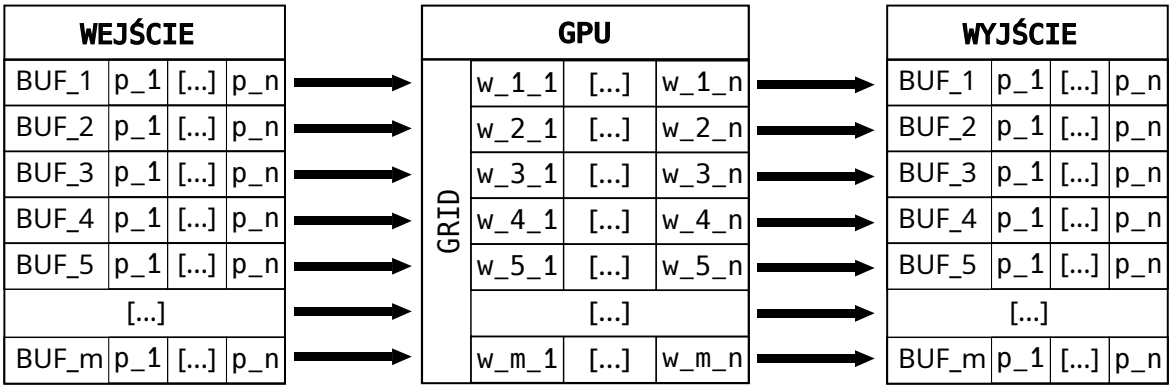
formatu PCM sprawia, że przetwarzanie sygnałów dźwiękowych jest zadaniem, które można wykonać przy użyciu algorytmów korzystających z równoległości. Najprostszym przykładem jest zastosowanie wykorzystanie n wątków GPU do przetworzenia n próbek sygnału zawartych w buforze. Sytuacja przedstawiona na Rysunku 2.4 ilustruje metodę, którą można by bez problemu wykorzystać przy zrównoleglaniu nieskomplikowanych algorytmów. W przypadku algorytmów bardziej złożonych lub wymagających sekwencyjnego przetwarzania danych dla kolejnych próbek, może się stać niemożliwym wykorzystanie tego podejścia. Rozwiązaniem tego problemu może być zastosowanie m wątków do przetworzenia m buforów w sposób liniowy, zakładając, iż dany algorytm zostaje wywołany jednocześnie m-krotnie. Przykład takiego podejścia przedstawiono na Rysunku 2.5. O ile te rozwiązanie nie jest tak samo wydajne, co pierwsze z przedstawionych, nie koniecznie musi implementować cały algorytm, a jedynie jego problematyczną część. Idealną sytuacją byłoby wywołanie algorytmu niesekwencyjnego m-krotnie. Taki przypadek pozwalałby na wykorzystanie pełni mocy obliczeniowej karty graficznej. Przykład takiego podejścia przedstawiono na Rysunku 2.6. Dobór podejścia może okazać się kluczowy dla uzyskania zadowalających wyników. Niewykluczone, iż w wielu przypadkach konieczne będzie zastosowanie hybrydowego podejścia, łączącego przedstawione metody.



Rysunek 2.4: Wykorzystanie n wątków do przetworzenia n próbek



Rysunek 2.5: Wykorzystanie m wątków do przetworzenia m buforów w sposób liniowy



Rysunek 2.6: Wykorzystanie m*n wątków do przetworzenia m buforów po n próbek

Rozdział 3

Wnioski

Przedstawiona implementacja systemu syntezującego i przetwarzającego sygnały dźwiękowe udowadnia, iż jest to możliwe przy urzyciu karty graficznej. Problematyka przedstawionego zagadnienia jest wysoce kompatybilna z metodami równoległego przetważania danych. Wiele algorytmów przetwarzania sygnałów dźwiękowych, które są obecnie wykonywane na CPU, może być bezproblemowo przeniesione na GPU przy użyciu takich platform jak CUDA. Platforma ta zapewnia łatwy dostęp do zasobów karty graficznej, co pozwala na wykorzystanie jej mocy obliczeniowej i przeniesienie kodu na GPU w sposób stosunkowo nieskomplikowany, nieingerujący nadto w strukturę systemu. Jedynym mankamentem okazał się brak pełnego wsparcia dla polimorfizmu w języku CUDA, co wymusiło zmianę sposobu zastosowania wzorca strategii. Na rynku dostępne jest obecnie wiele innych techologii, pozwalających na wykorzystanie mocy obliczeniowej GPU, takich jak OpenCL, OpenACC, czy SYCL. Każda z nich mogła by być równie dobrze wykorzystane do tego rodzaju zadań w zależności od indywidualnych preferencji programisty oraz wymagań stawianych przed projektem. Powstały system bazujący na GPU bezproblemowo dorównuja, a niekiedy nawet przewyższa wydajnością swój odpowiednik powstały na CPU, pomimo iż kwestje optymalizacji nie zostały poruszone w tej pracy. Implementacja bardziej złożonych algorytmów, a w szczególności algorytmów sekwencyjnych w sposób optymalny może okazać się jednak trudniejsza i wymagać głębszej analizy problemu.

Rozdział 4

Możliwości rozwoju

4.1 Optymalizacja obecnego rozwiązania

Ideą stojącą za wykorzystanie karty graficznej było zagospodarowanie dodatkowej mocy obliczeniowej w celu umożliwienia przetwarzania większej ilości danych w czasie rzeczywistym.

4.1.1 Przystosowanie algorytmów do wykorzystania GPU

Przedstawiona implementacja kodu w języku CUDA (przystosowana do wykorzystania GPU) przedstawia algorytmy, których zasada działania jest bliźniaczo podobna do implementacji w języku C++ (przystosowanej do wykorzystania CPU). W związku z tym, wiele z nich obarczonych jest koniecznością wykonywania sekwencyjnego, co nie pozwala na pełne wykorzystanie mocy obliczeniowej karty graficznej. Rozwiązaniem może być całkowita rezygnacja z sekwencyjności, co prawdopodobnie wiązałoby się z brakiem możliwości uzyskania identycznych wyników do klasycznych algorytmów. Nie koniecznie jest to wada, w wielu przypadkach ludzka percepcja nie byłaby wystarczająca aby nawet dostrzec różnicę.

4.1.2 Grupowanie wywołań kerneli

Rezygnacja z sekwencyjności w wielu przypadkach nie jest jednak konieczna. Można by zastosować system przypominający relację klient-serwer: klient (instancja wykorzystująca konkretny algorytm) wysyła swoje dane do serwera (jednostki opartej o wzorzec singletonu), gdzie są one przetwarzane jednocześnie dla wszystkich możliwych klientów jednocześnie. Algorytm decydujący o możliwym połączeniu wywołań algorytmów mógłby bazować na obecnej implementacji zawartej w klasie ExecutionQueue - algorytmu przechodzenia po drzewie. Różnicą była by konieczność

powiązania ze sobą gałęzi drzewa, które mogą być przetwarzane jednocześnie, a następnie pogrupowanie ich w kolejności pozwalającej na uzyskanie najmniejszej liczby wywołań kerneli. Równocześnie rozwiązanie to pozwalało by w większości przypadków na obliczenia asynchroniczne. Zastosowanie tego rozwiązanie przy zachowaniu sekwencyjności było by najskuteczniejsze w przypadków w których dany algorytm jest wykorzystany wielokrotnie w tym samym czasie. Za przykład może posłużyć synteza dźwięku dla wielu syntezatorów, gdzie każdy z nich syntezuje wiele głosów (wciśniętych klawiszy) jednocześnie. Ten samo rozwiązanie mogło by również przynieść korzyści dla algorytmów nie sekwencyjnych, poprzez zwiększenie rozmiaru bloku bądź gridu.

4.1.3 Implementacja systemu hybrydowego

Niezaprzeczalnie zarówno CPU jak i GPU posiadają swoje mocne oraz słabe strony, wynikające z ich architektury. Tę wiedzę można wykorzystać w celu doboru odpowiedniego urządzenia do konkretnego zadania. Skutecznym rozwiązaniem może okazać się system hybrydowy, który posiadał by implementację zarówno na GPU, jak i CPU. W zależności od potrzeb, system mógłby decydować o wyborze jednej z opcji biorąc pod uwagę zarówno zysk w wykorzystaniu zasobów biorący się z konkretnego rozwiązania oraz czas potrzebny na ewentualne przesłanie danych pomiędzy urządzeniami. W tym przypadku skutecznym może okazać się asynchroniczny przesył danych pomiędzy urządzeniami. Pozwoliło by to na równomierne wykorzystanie mocy obliczeniowej obu urządzeń w najopłacalniejszy w danym kontekście sposób przy jednoczesnym zachowaniu możliwości zachowania algorytmów sekwencyjnych, które preferowane są dla implementacji CPU.

4.2 Wykorzystanie innych technologii

Możliwości karty graficznej nie kończą się na przyspieszaniu obliczeń wykonywanych uprzednio na CPU. Nowoczesne karty graficzne posiadają wieli technologii, które mogą być wykorzystane do utworzenia zupełnie nowych algorytmów, które nie mogły by zostać skutecznie zastosowane w przypadku CPU.

4.2.1 Wykorzystanie rdzeni ray-tracingu

Ray-tracing to technika generowania obrazów poprzez śledzenie promieni światła. Jest to technika stosowana w grafice komputerowej, która pozwala na uzyskanie bardzo realistycznych obrazów. W przeciwieństwie do tradycyjnego renderowania,

gdzie obliczenia są wykonywane dla każdego piksela, ray-tracing pozwala na uzyskanie obrazu poprzez śledzenie promieni światła od obserwatora do obiektów na scenie. Można by wykorzystać tę technologię to śledzenia fal dźwiękowych od źródła dźwięku przez obiekty na scenie, aż do obserwatora. Take wykorzystanie tej technologii może pozwolić na uzyskanie bardziej realistycznych efektów, takich jak echa, czy innych zjawisk związanych z propagacją fal dźwiękowych w przestrzeni. Z kolei takie zastosowanie pozwoliło by na dokładne symulowanie akustyki konkretnych pomieszczeń (sal koncertowych, filharmonii, katedr, itp.) oraz ostatecznie takie narzędzie mogło by okazać się przydatne dla architektów, dając im w ten sposób możliwość symulowania akustyki budynków/pomieszczeń jeszcze w fazie projektu.

4.2.2 Wykorzystanie rdzeni tensorowych

Rdzenie tensorowe to jednostki obliczeniowe, które są specjalnie zaprojektowane do wykonywania operacji na tensorach - wielowymiarowych tablic, które mogą przechowywać dane w dowolnym wymiarze. Rdzenie tensorowe są powszechnie wykorzystywane w technologiach związanych z uczeniem maszynowym, takich jak sieci neuronowe. System przetwarzania dźwięku, który pozwala na wykorzystanie karty graficznej, umożliwia na prostą integrację wykorzystania rdzeni tensorowych w algorytmach przetwarzających sygnał dźwiękowy, a co za tym idzie, na wykorzystanie zaawansowanych technik uczenia maszynowego w celu uzyskania lepszych rezultatów niż było to możliwe w przypadku samego CPU.

4.2.3 Przyszłe technologie

Karty graficzne są jednym z najszybciej rozwijających się obszarów technologii komputerowych. Są one obecnie odbiciem rozwoju szeroko pojętej informatyki, zaczynając od przetwarzania dużych zbiorów danych, poprzez branżę rozrywkową, aż po techniki uczenia komputerowego. W związku z tym jest to obszar, w którym można się spodziewać ciągłych innowacji, nowych rozwiązań i wzrostu mocy obliczeniowej. Wykorzystanie karty graficznej do przetwarzania dźwięku to dziedzina, która ma ogromny potencjał, zarówno w celu przyspieszenia obliczeń dla algorytmów obecnie implementowanych na CPU, jak i potencjalnie w celu uzyskania zupełnie nowych sposobów na przetwarzanie sygnałów dźwiękowych.

Spis listingów

Spis tabel

Spis rysunków

Bibliografia