Programowanie CUDA na NVIDIA GPU (PKG)
Termin oddania: 7 zajęcia semestru (30 maja – 12 czerwca w zależności od planu zajęć)
(**projekt jest wykonywany i zaliczany w grupach max 2 osobowych- wersje projektu przydziela prowadzący zajęcia**)

Warianty zadania mnożenia macierzy:

1. badanie prędkości obliczeń w zależności od typu wykorzystywanej pamięci (wersje 2, 3 i 4)
2. badanie prędkości obliczeń w funkcji ilości pracy wątków (wersje 1, 2, 5)
3. badanie prędkości obliczeń w funkcji ilości pracy wątków (wersje 1, 2, 6)
4. badanie prędkości obliczeń w funkcji ilości pracy wątków (wersje 1, 3, 5)
5. badanie prędkości obliczeń w funkcji ilości pracy wątków (wersje 1, 3, 6)
6. badanie wpływu organizacji dostępu do pamięci globalnej na efektywność przetwarzania (wersje 2,3,4) – przygotować kody z dostępami łączonymi (na tyle na ile algorytm pozwala)
7. badanie wpływu organizacji dostępu do pamięci globalnej na efektywność przetwarzania (wersje 1,2,3) – przygotować 2 kody z dostępami do pamięci globalnej: łączonymi i nie łączonymi
8. ukrycie kosztów transferu danych w czasie obliczeń (dla wersji kodu 3 i 7)

Wersje programu mnożenia macierzy dla powyższych wariantów zadania:

1. jeden blok wątków przetwarzania, obliczenia przy wykorzystaniu pamięci globalnej, mnożenie dowolnych tablic o rozmiarach będących wielokrotnością rozmiaru bloku wątków.
2. grid wieloblokowy, jeden wątek oblicza jeden element macierzy wynikowej, obliczenia przy wykorzystaniu pamięci globalnej,
3. grid wieloblokowy, jeden wątek oblicza jeden element macierzy wynikowej, obliczenia przy wykorzystaniu pamięci współdzielonej bloku wątków,
4. grid wieloblokowy, jeden wątek oblicza jeden element macierzy wynikowej, obliczenia danych przy wykorzystaniu pamięci współdzielonej bloku wątków ze zrównolegleniem obliczeń i pobierania danych z pamięci globalnej w ramach każdego bloku wątków,
5. grid wieloblokowy, jeden wątek oblicza 2 lub 4 (podział pracy dwuwymiarowy) sąsiednich elementów macierzy wynikowej, obliczenia przy wykorzystaniu pamięci współdzielonej bloku wątków,
6. grid wieloblokowy, jeden wątek oblicza 2 lub 4 (podział pracy dwuwymiarowy) sąsiednich elementów macierzy wynikowej, obliczenia danych przy wykorzystaniu pamięci współdzielonej bloku wątków ze zrównolegleniem obliczeń i pobierania danych z pamięci globalnej w ramach bloku wątków.
7. grid wieloblokowy, jeden wątek oblicza jeden element macierzy wynikowej, obliczenia przy wykorzystaniu pamięci współdzielonej bloku wątków, zrównoleglenie obliczeń i transferu danych między pamięciami operacyjną procesora, a globalną karty

Warianty zadania sumowanie wektora

Analiza porównawcza efektywności w funkcji organizacji przetwarzania (kod z brakiem rozbieżności ma rozbieżność minimalizowaną do minimum):

9. Warianty z rozbieżnością przetwarzania wątków i bez, użycie wyłącznie pamięci globalnej.
10. Warianty z rozbieżnością przetwarzania wątków i bez, użycie pamięci współdzielonej.
11. Przetwarzanie bez rozbieżności wątków, dostęp efektywny, pamięć globalna i pamięć współdzielona.
12. Przetwarzanie bez rozbieżności wątków, porównanie kodów z dostępem do danych wyłącznie w pamięci globalnej realizowanym z sposób nieefektywny i efektywny.
13. Przetwarzanie bez rozbieżności wątków, dostęp do danych nieefektywny i efektywny - pamięć współdzielona.
14. Przetwarzanie bez rozbieżności wątków, wzrost ilości pracy wątku w pierwszym kroku (porównać przetwarzanie dla 1,2,3,4 sumowań), dostęp do danych efektywny, pamięć globalna.
15. Przetwarzanie bez rozbieżności wątków, wzrost ilości pracy wątku w każdym kroku (porównać przetwarzanie dla 1,2,3,4 sumowań), dostęp do danych efektywny, pamięć globalna.
16. Przetwarzanie bez rozbieżności wątków, wzrost ilości pracy wątku w pierwszym w kroku sumowania (porównać przetwarzanie dla 1,2,3,4 sumowań), dostęp do danych efektywny, pamięć współdzielona.
17. Przetwarzanie bez rozbieżności wątków, wzrost ilości pracy wątku w każdym w kroku sumowania (porównać przetwarzanie dla 1,2,3,4 sumowań), dostęp do danych efektywny, pamięć współdzielona.

Informacje szczegółowe

Celem ćwiczenia jest:

- zapoznanie praktyczne (poprzez przygotowanie kilku wersji kodu) z zasadami programowania równoległego procesorów kart graficznych (PKG),
- zapoznanie z zasadami optymalizacji kodu dla PKG,
- tworzenie prostych funkcji dla PKG dotyczących problemu mnożenia macierzy i sumowania wektora (redukcja) oraz
- ocena prędkości i efektywności przetwarzania na PKG.

Należy przygotować, opisać i zbadać prędkość i efektywność wymaganych wersji programów.

Uwagi do rozmiaru przetwarzanych tablic (mnożenie tablic kwadratowych i sumowanie):

- Minimalny rozmiar powinien zapewnić uzyskanie maksymalnej możliwej zajętości wszystkich multiprocesorów,
- Rozmiar powinien być wielokrotnością wielkości bloku zapewniającą przejrzystą strukturę kodu (bez konieczności sprawdzania dodatkowych warunków brzegowych wynikających z rozmiaru danych – typu „czy proces ma pracę?")
- Maksymalny rozmiar z szeregu testowanych instancji;
  - dla mnożenia 2048,
  - dla sumowania ograniczony zakresem 4 bajtowej zmiennej sterującej typu całkowitego umożliwiającej zapisanie wartości ze znakiem - 2 exp 30
  - ograniczony odpornością systemu operacyjnego na chwilowy brak dostępności do karty graficznej.

Wymiary bloku wątków w zakresie:

- Mnożenie macierzy: 8x8, 16x16, (w miarę możliwości 32x32)
- Redukcja: 32, 128, 512, (1024 w zależności od możliwości karty).

Analizowanie oddzielnie dla wymaganych wersji kodu, wielkości instancji i parametrów gridu:

- **prędkości obliczeń** (złożoność obliczeń/czas),
- **przyspieszenie** (i zmiana przyspieszenia w funkcji wielkości instancji) w stosunku do obliczeń **sekwencyjnych** na komputerze ogólnego przeznaczenia najlepszą dostępną metodą (np. 3 pętle kolejność ikj)
- **stosunek** CGMA (CGMA – obliczane teoretycznie na podstawie kodu, dodatkowo CGMA obliczane przez profiler – podanie wykorzystanych wzorów z uzasadnieniem analizy teoretycznej), na podstawie analizy kodu i wyników z Nvidia profiler.
- **miara stopnia łączenia dostępu** do pamięci globalnej z wyjaśnieniami – określenie średniej liczby transakcji z pamięcią globalną przypadającą na żądanie dostępu do pamięci na poziomie wiązki 32 wątków,
- **zajętość** multiprocresora,
- funkcja skrócenia czasu przetwarzania dzięki zrównolegleniu kopiowania danych (CPU-GPU i GPU-CPU) i przetwarzania (dla porównania wersji 3 i 7 mnożenia macierzy), w tej szczególnej wersji zadania badanymi zależnościami przetwarzania będzie wpływ wielkości bloku macierzy przesyłanej i liczby bloków na wielkość skrócenia czasu przetwarzania.

Różnorodność sprawozdań.

Mimo, że podobieństwa w realizacji projektu przez poszczególne grupy są nieuniknione, zwracam się z apelem o samodzielną pracę nad zadaniami, gdyż pozwoli ona na zrozumienie zagadnień programowania równoległego na PKG. Istnieje wystarczający zakres dowolności działań zarówno w ramach eksperymentu, doborze instancji i parametrów uruchomienia kodu (wymiary tablicy, gridu, bloku) i wyciągania wniosków na podstawie posiadanych wiadomości. Proszę odpowiednio wcześnie podjąć pracę nad projektem, aby można ją było wykonać w wymaganym terminie samodzielnie.

Opis zadania został opublikowany 23.04.2018 i przydzielony najpóźniej na pierwszych zajęciach majowych – 1 miesiąc przed terminem oddania.

W przypadku potrzeby zapewnienia dostępności laboratorium dla dodatkowego czasu pracy przy komputerze proszę o kontakt z wyprzedzeniem w celu udostępnienia laboratorium.

Dokumentacja
Po wykonaniu zadania proszę dostarczyć wydruk (dwustronny) krótkiego sprawozdania zawierającego:

- opis wykorzystywanej karty graficznej (typ, CC – możliwości obliczeniowe, liczba SM i rdzeni, innych jednostek obliczeniowych, ograniczenia dla CC)
- opis zakresu zrealizowanego zadania,
- kluczowe fragmenty kodów kerneli z **wyjaśnieniami** dotyczącymi znaczenia instrukcji (odwołania do wymaganych szczegółowych kodu zadania),
- wzory zastosowane obliczeń wszystkich prezentowanych miar efektywności,
- najważniejsze wyniki w postaci tabelarycznej (oraz wykresów - opcjonalnie dla lepszej wizualizacji), tabele (i wykresy) należy ponumerować i podpisać w sposób nie budzący wątpliwości co do zawartości, analiza poprawności prezentowanych wartości i ich zależności od wielkości instancji – czy wartości mieszczą się w dopuszczalnym zakresie.
- wnioski z wykonanych eksperymentów z uzasadnieniem obserwowanych wartości (czytelne odwołanie do omawianej wielkości, gdzie się znajduje i jakiego uruchomienia (system, parametry, wersja kodu) dotyczą.

Pozostałe informacje proszę dostarczyć w wersji elektronicznej. Będą to
- pliki źródłowe projektu,
- przykładowe zrzuty kilku przykładowych ekranów z wynikami profilowania,
- plik z pozostałymi informacjami wg uznania: tabele z surowymi wielkościami zebranymi podczas eksperymentu, informacje szczegółowe z pomiarów efektywnościowych przetwarzania.

Zaliczanie projektu będzie wymagało wiadomości z zakresu przetwarzania równoległego na PKG (często zawartość sprawozdania świadczy o posiadaniu lub braku wiadomości), a w szczególności poniżej wymienionych zagadnień (w przypadku braku pomysłu na wnioski z przeprowadzonego eksperymentu opracowanie poniższych zagadnień należy zawrzeć w opracowaniu we wnioskach):
1. Możliwość / brak możliwości łączenia dostępów do pamięci dla wątków w ramach poszczególnych wersji kodów - uzasadnienie.
2. Umiejętność wyznaczenia CGMA dla poszczególnych wersji kodu, znaczenie wielkości CGMA dla poszczególnych wersji kodu – na podstawie kodu.
3. Analiza wpływu na prędkość przetwarzania takich czynników jak: wielkość bloku wątków, wielkość instancji problemu, zajętość multiprocesora, żądania zasobowe związane z wariantem kodu: liczba rejestrów i wymaganie na pamięć współdzieloną
4. Umiejętność oceny liczby dostępów do pamięci globalnej (odczyt i zapis) w funkcji rozmiaru problemu dla poszczególnych wersji kodu.
5. Umiejętność oceny liczby operacji zmiennoprzecinkowych w funkcji rozmiaru problemu.
6. Umiejętność oceny wielkości żądania na wielkość pamięci współdzielonej wątków w ramach poszczególnych wersji kodu.
7. Umiejętność oceny potrzeby, zakresu, zadań i skutków synchronizacji przetwarzania w poszczególnych wersjach kodu.
8. Umiejętność oceny wpływu wielkości bloku wątków na łączenie dostępów do pamięci globalnej w poszczególnych wersjach kodu.

INFORMACJE UZUPEŁNIAJĄCE

1. Eksperyment można przeprowadzić w laboratorium 2.7.6 dla systemu GTX 260. Alternatywnie można przeprowadzić testy również przy użyciu innych kart NVIDIA (jeśli zastosowano inną kartę to proszę w sprawozdaniu opisać kartę: liczba rdzeni, liczba multiprocesorów i parametry jej zdolności obliczeniowych (CC).
2. Przy tworzenie kodu dla GPU proszę bazować na projekcie matrixMul dostępnym w ramach "CUDA SDK code samples" i dołączonym do niniejszego opisu. Dla minimalizacji ilości pracy programistycznej warto skorzystać z tego kodu jako wzorca takich działań jak: pomiar czasu obliczeń, przygotowanie danych i sprawdzenie poprawności obliczeń. Zachęcam do skorzystania również z innych przykładowych rozwiązań problemów w CUDA dostępnych w "CUDA SDK code samples" . Skorzystanie z tych kodów może się wiązać z koniecznością

uzupełnienia właściwości projektu w części VC++ Directories/Include Directories o katalog pakietu NVIDIA zawierający informacje o funkcjach pomocniczych wykorzystywanych w przykładach.

3. Miary dostępne w Nvidia profiler są zależne od CC i dla CC 1.X zawierają przykładowo:

- global store requests i global load requests – zdarzenia polegające na dostępie do pamięci na poziomie kodu, liczone **jednokrotnie** dla całej wiązki 32 wątkowej, zdarzenia te liczone są dla wszystkich bloków przetwarzanych **tylko na 0 multiprocesorze**
- gld_efficiency – efektywnośc pobierania danych z pamięci globalnej wg wzoru (gld_request /((gld_32 + gld_64 + gld_128) / (2* #SM)))
- gst_efficiency – efektywność zapisu danych do pamięci globalnej wg wzoru (gst_request /((gst_32 + gst_64 + gst_128) / (2* #SM)))

Gdzie:

- #SM - liczba muliprocesorów
- Gst_64 - liczba transferów (typu zapis 64 bajtowych (np. 16 watków 4 bajty (integer)) realizowane przez 3 SM w TPC 0, inne parametry gst_* są liczone analogicznie.

Przykład obliczeń parametrów:
Maksymalna wartość gst_efficiency i przykład obliczania dla CC= 1.* Sytuacja analizowana: występują 4 bajtowe (zmienna float) dostępy łączone – to znaczy, że 16 wątków wymaga - każdy jednego słowa 4 bajtowego, a słowa te znajdują się w bloku wyrównanym do granicy 64 bajtów.

- Występują transfery 64 bajtowe (16*4) - Gst_64 jest niezerowy
- Gst_64 jest większy 6 razy od gst_request (zakładamy, że każdy z multiprocesorów wykonuje tyle samo warpów) gdyż:
- Gst_64 jest wielkością większą od gst_request: 3 razy ze względu na liczbę procesorów (3) i dodatkowo Gst_64 jest 2 razy większy od gst_request ze względu na łączenie dostępów dla połowy warp (w wiązce gst_request liczone raz, a jedna wiązka 32 elementowa musi mieć 2 transfery).
- Zatem dla GTX 260 z 27 SM maksymalna wartość gst_efficiency wynosi 9 (po podstawieniu 2*27/6=9) ; tak jest dla wszystkich transferów (zapisów) łączonych 64 bajtowych (wg wzoru: 2*27*gst_request/ (6*gst_request)).

Tworzenie projektu CUDA dlla Visual Studio.
Tworzenie projektu dla PKG w Visual studio wymaga takiego skonfigurowania Visual studio, aby posiadał zasady kompilacji projektu CUDA (Build Customizations for CUDA Projects). Można to zrealizować korzystając z opcji File-> New | Project... NVIDIA-> CUDA->, a następnie wybranie wzorca CUDA Toolkit version - np. "CUDA 5.0 Runtime". Wzorzec skonfiguruje projekt do wykorzystania CUDA 5.0 Toolkit. Nowy projekt jest projektem C++ lecz skonfigurowanym do użycia NVIDIA's Build Customizations. Wszystkie własności projektów Visual Studio C++ projects pozostają dostępne.

NVIDIA Visual Profiler
Praca z NVIDIA Visual Profiler wymaga:

- utworzenia kodu wynikowego badanej aplikacji,
- uruchomienia NVIDIA Visual Profiler,
- utworzenia sesji oceny efektywności (profilowania) i wskazania pliku uruchamianego kodu i
- uruchomienia procesu oceny aplikacji.
- Podczas wielokrotnych uruchomień programu zbierane będą statystyki i mierzone czasy realizacji poszczególnych funkcji przetwarzania dotyczącego GPU. Wyniki profilowania dostępne są w poszczególnych widokach: Czasu przetwarzania (Timeline View), widoku analiz (Analysis View), widoku szczegółów (miar efektywności) Details View, Detail Graphs View, Properties View, Console View (standardowe wyjście przetwarzania kodu) , Settings View (określenie ścieżki kodu i parametrów linii uruchomienia).

Ostatnie zmiany zawartości dokumentu: 23.04.2018.