- **Przetwarzanie przy użyciu procesorów kart graficznych (PKG) w środowisku CUDA**
- **ĆWICZENIA laboratoryjne rok ak.2016/2017 PR studia niestacjonarne (opracowano 12.04.2017, zmiana 17.07.2017 dotyczy przepustowości obliczeń)**

Informacje wstępne:

Tworzenie projektu CUDA dlla Visual Studio.

Tworzenie projektu dla PKG w Visual studio wymaga takiego jego skonfigurowania aby posiadał zasady kompilacji projektu CUDA (Build Customizations for CUDA Projects). Mozna to zrealizować korzystając z opcji File-> New | Project... NVIDIA-> CUDA->, a nastepnie wybranie wzorca CUDA Toolkit version - np. "CUDA X.Y Runtime". Wzorzec skonfiguruje projekt do wykorzystania CUDA X.Y Toolkit. Nowy projekt jest projektem C++ lecz skonfigurowanym do użycia NVIDIA's Build Customizations. Wszystkie własności projektów Visual Studio C++ projects pozostają dostępne. Proszę w opcjach kompilacji **wybrać** odpowiednią dla używanej karty **wersję generowanego kodu** wynikowego.

Zadania do wykonania;

- ZADANIA 1 i 2 na potrzeby zapoznania się z systemem,
- Zadanie 3 dla przeprowadzenia eksperymentu obliczeniowego i jego omówienia w świetle posiadanych wiadomości o systemach PKG **w sprawozdaniu z projektu**.

ZADANIE 1: Przeanalizować zawartość kodu przykładowego projektu powstającego przy wykorzystanu wzorca projektowego CUDA

1. Określić części składowe kodu i ich zadania.
2. Określić zadania wykonywane przez wątki w ramach funkcji kernela.
3. Skompilować kod i uruchomić program.

Zadanie 2: korzystanie ze stałych automatycznych wątków określających ich identyfikator wątku w bloku i identyfikator globalny w gridzie

Korzystając z kodu z zadania pierwszego uruchomić kod, którego zadaniem jest wypełnienie (przez wątki uruchomione na PKG) tablicy (int) identyfikatorami wątków – każdy wątek zapisuje w „swoim" elemencie tablicy (o numerze będącym globalnym identyfikatorem wątku w bloku jednowymiarowym idx = blockIdx.x*blockDim.x + threadIdx.x)
„swój identyfikator" równy 1000*blockIdx.x + threadIdx.x

A) Przygotować kod dla **jednego bloku wątków** dla tablicy o rozmiarze ograniczonym wielkością jednowymiarowego bloku wątków. **KOD 2.1.**

B) Przygotować kod dla dowolnej wielkości tablicy uruchomiony z **gridem** o wielkości zależnej od **wielkości tablicy i wielkości bloku**. **KOD2.2.**

Uwaga – wyjaśnienia: grid składa się z bloków jednakowej wielkości, liczba wątków musi być wystarczająca do zainicjowania całej tablicy, lecz część wątków nie będzie zapisywać wartości w tablicy.

Jeśli tablica jest 20 elementowa o blok wątków ma wielkość blockDim.x =7 to kolejne elementy tablicy powinny mieć wartość:

0,1,2,3,4,5,6,1000,1001,1002,1003,1004,1005,1006,2000,2001,2002,2003,2004,2005

Proszę wykorzystać do obliczeń różne wielkości bloków (32, 64, 256,512).

Zadanie 3: Scalanie wartości w określonym sąsiedztwie (uwzględniane wsprawozdniu z projketu)

Wektor wynikowy zawiera wartości będące wynikiem scalenia wartości wejściowych pobieranych w określonym obszarze (praca na dostarczonym szkielecie pliku kodu)

Przykład obliczeń :

Wejście: 1,2,3,4,5,6 PROMIEŃ = 1, scalenie operacją dodawania

Wyjście: 6,9,12,15 wyliczone jako sumy 3 (ze względu na PROMIEŃ=1) wartości sąsiednio umieszczonych w tabelicy: 1+2+3=6, 2+3+4=9, 3+4+5=12,4+5+6=15

Architektura procesów: grid i bloki jednowymiarowe

Dane wejściowe: tablica jednowymiarowa o rozmiarze: N+2*PROMIEN.

Wyniki: tablica jednowymiarowa o rozmiarze N.

Do wykonania – analiza kodu programu:

Części kodu aplikacji:

A) Alokacja danych w RAM i pamięci karty
B) Zainicjowanie danych wejściowych
C) Kopiowanie danych do pamięci karty
D) Uruchomienie kernela i realizacja obliczeń na PKG
E) Kopiowanie wyników obliczeń
F) Sprawdzenie poprawności obliczeń

Do wykonania eksperymenty dla różnych wartości PROMIEŃ = 1, 10 i 100 i rozmiarów instancji (dane typu zmiennoprzecinkowego pojedynczej precyzji – FLOAT) zapewniających nasycenie obliczeniami wykorzystywanego systemu karty graficznej. Proszę uruchomić obliczenia, zmierzyć czas, miary efektywności dostępów do pamieci globalnej i obliczyć przepustowość obliczeń używając bloków WĄTKÓW o wielkości 32,64,128,256,512 i tablic danych 10exp4,10exp6,10exp9.

1. Proszę przygotować wersje **kodu jednoblokowego** – OGRANICZENIE INSTANCJI: rozmiar tablicY wyniku = wielkość bloku wątków – **KOD 3.1.**
2. Proszę przygotować wersje **kodu wieloblokowego** – **KOD 3.2.**
3. Proszę przygotować wersje **kodu wieloblokowego z wykorzystaniem pamięci współdzielonej bloku wątków** – **KOD 3.3.**.
4. Dla każdej wersji kodu proszę sprawdzić poprawność obliczeń na różnych danych (omówić w sprawozdaniu sposób przeprowadzenia testów poprawności i jego wyniki).
5. Dla przygotowanych wariantów kodu proszę zmierzyć czas obliczeń i parametry kernela (liczba rejestrów i wielkość pamięci współdzielonej) za pomocą Nvidia Visual Profiler.

Zawartość sprawozdania z ZADANIA 3:

1. **Informacje o użytej karcie: model, compute capability, ograniczenia dla compute capability, liczba rdzeni, liczba multiprocesorów.**
2. Kody przygotowanych efektywnych kerneli, w których wątki współpracują przy wykonaniu zadania.
3. Opis sposobu i wyniku testowania poprawności przetwarzania.
4. **PRZEPUSTOWOŚĆ**: W przepustowości systemu dla badanego problemu uwzględniamy:
    a. **czas obliczeń kernela**, zmierzony za pomocą profilera lub za pomocą zdarzeń wprowadzonych do strumienia operacji CUDA
    b. **liczbę operacji arytmetycznych** na przetwarzanych danych (bez uwzględnienia operacji na zmiennych sterujących pętlami), jest ona równa: N*2*R gdzie N jest liczbą wyników w tablicy wyjściowej, a R promieniem scalania
    c. przepustowość = **liczbę operacji arytmetycznych /czas obliczeń kernela.**
5. Zajętość teoretyczna SM i jej znaczenie: Proszę za pomocą CUDA occupancy calculator wyznaczyć zajętość teoretyczną SM (multiprocesora strumieniowego) dla KOD2.2 i KOD 2.3.

6. Miary efektywności dostępu do pamięci globalnej: Wykorzystać **NVIDIA Visual Profiler** do porównania efektywności dla wersji kodu bez pamięci współdzielonej i z pamięcią współdzieloną pod względem liczby dostępów do pamięci globalnej i możliwości łączenia dostępów do pamięci globalnej (por. miary efektywności poniżej i dostępne w NVIDIA VP dla badanej karty).
7. CGMA: Wyznaczyć (i wyjaśnić) TEORETYCZNIE wielkość parametru CGMA dla przygotowanego kodu.
8. Synchronizacja wątków: Omówić jaka jest przyczyna zastosowania synchronizacji w kodzie, kiedy jest potrzebna, jakie są efekty wprowadzenia funkcji synchronizacji do kodu (pozytywne i negatywne).
9. Przyczyny różnic prędkości: Porównać wartości przepustowości obliczeń, określić-wyjaśnić przyczynę różnic, określić-wyjaśnić możliwy wpływ : liczby bloków, wielkości bloku wątków, zajętości multiprocesora, CGMA, synchronizacji wątków.
10. Tabele: Wszystkie pomierzone i wyliczone wartości parametrów proszę umieścić w czytelnych tabelach pozwalających na ich porównanie. Każda tabela powinna zawierać informacje o zawartości (wersja kodu, wielkość instancji, użyte parametry).
11. Sprawozdanie powinna zakończyć informacja: „Przed oddaniem sprawozdania sprawdzono i uzupełniono sprawozdanie tak, aby zawierało wszystkie wymienione w opisie wymagania". Wymagane informacje dla czytelności prezentacji proszę przedstawiać w kolejnych punktach sprawozdania.

UWAGA:
Do zaliczenia wymagana jest znajomości i rozumienie takich pojęć jak: kernel, grid, blok wątków, konfiguracja uruchomienia kernela, multiprocesor, osnowa, WARP - wiązka 32 watków o sąsiednich identyfikatorach modulo 32, łączenie dostępów do pamięci globalnej – lokalność przestrzenna dostępu do danych, pomiar czasu przetwarzania na karcie graficznej, synchronizacja pracy procesora i karty graficznej, zajętość multiprocesora (znaczenie i wzór), CGMA (znaczenie i wzór).

NVIDIA Visual Profiler

Praca z NVIDIA Visual Profiler wymaga:
- utworzenia kodu wynikowego badanej aplikacji,
- uruchomienia NVIDIA Visual Profiler,
- utworzenia sesji profilowania i wskazania pliku uruchamianego kodu i
- uruchomienia aplikacji.

Podczas wielokrotnych uruchomień programu zbierane będą statystyki i mierzone czasy realizacji poszczególnych funkcji przetwarzania dotyczącego PKG.

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).

Miary efektywności przetwarzania do zmierzenia i przeanalizowania w uruchamianych kodach.

Karta GTX 260 (laboratorium 2.7.6), miary efektywności obliczeń (i ich znaczenie) dostępne w Visual Profiler dla innych kart graficznych należy sprawdzić w dokumentacji oprogramowania Visual Profiler.

sm_cta_launched – liczba bloków wątków przetwarzana na multiprocesorze 0
cta_launched - liczba bloków przetwarzana na TPC(thread processing cluster) 0
global store requests i **global load requests** – dostępy do pamięci na poziomie kodu, liczone jednokrotnie dla całego warp, zdarzenia liczone dla wszystkich bloków przetwarzanych 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)))
#SM liczba muliprocesorów
Gst_64 - liczba transferów (zapisów) 64 bajtowych (np. 16 watków 4 bajty (integer)) realizowane przez 3 SM w TPC 0 . W przypadku dostępów do słowa 4 bajtowego, łączonychsąsiednich
dla 16 wątków **Gst_64** jest 6 x większa niż global store requests. Zatem dla GTX 260 z 27 SM maksymalna wartość gst_efficiency np. dla wszystkich transferów (zapisów) łączonych 64 bajtowych = 9 (wg wzoru: gld_request/ (6xgld_request)/(2*27)).
