Grupa 1	Kamil Sacha, Konrad Szwedo	17/11/2016

Zadanie 3 - Mnożenie macierzy GPU

Naszym zadaniem było napisanie programu obliczającego wartość macierzy $C = A \times B$ gdzie macierze A, B, C są kwadratowe oraz posiadają rozmiar $n \times n$ gdzie n jest potęgą 2. Program w celu przyśpieszenia obliczeń miał zostać stworzony w oparciu o środowisko Nvidia® CUDA™

Rozwiązanie problemu

Generowanie oraz proces obliczeń macierzy został przeniesiony z hosta (programu który wywołujemy na CPU) na proces graficzny, w związku z tym potrzebne było napisanie funkcji generujących i obliczających wynik mnożenia na karcie grafiki.

Definiowanie rozmiarów siatki i bloku

Pierwszym ważnym krokiem do rozpoczęcia obliczeń, jest zdefiniowanie odpowiednich rozmiarów siatki, oraz bloku (jako blok definiujemy ilość wątków indeksowanych 2-wymiarowo). W naszym programie zastosowaliśmy taki algorytm wybierania siatki tak aby rozmiar x oraz y (gdzie x to ilość bloków w płaszczyźnie poziomej a y pionowej) był zawsze wielokrotnością 16. Wymiary bloku zostały ustawione na sztywno 16×16 (rozmiar ten został dobrany eksperymentalnie). Blok ten został użyty również do metody generowania macierzy A, B **matrixMultiplyKernel(...)** która to generuje macierze z podanych wzorów zapisując przy użyciu tablic alokowanych na GPU.

```
1
2 dim3 threadsPerBlock(BLOCK_SIZE, BLOCK_SIZE);
3
4 dim3 grid( (int) ceil(cpu_A.matrixSize / (float)threadsPerBlock.x),
5            (int) ceil(cpu_A.matrixSize / (float)threadsPerBlock.y));
```

Listing kodu 1: Algorytm podziału na siatkę i bloki.

Dla przykładu: gdy wejściowe macierze A, B będą wymiarów 12×12 wtedy siatka będzie wymiarów 3×3 ponieważ zaokrąglamy w górę $33 \div 16 = 2.065$ (zaokrąglamy w górę do 3).

Algorytm mnożenia

Funkcja jądra obliczająca wynik mnożenia, została uproszczona do minimum w celu łatwego przeglądania kodu oraz przyśpieszenia obliczeń. Działa ona tak, że każda komórka macierzy C ma swój odpowiednik w utworzonym przez środowisko wątku. Każdy wątek odpowiedzialny jest za wymnożenie całego wiersza macierzy A z odpowiednią kolumną macierzy B. W tym celu są nam potrzebne indeksy dzięki którym możemy zidentyfikować dany wątek w odniesieniu do macierzy. Zostało również zapisane zabezpieczenie, przed wyjściem poza zakres. Taka sytuacja zdarzy się zawszę, gdy rozmiar macierzy nie będzie liczbą podzielną przez 16. W przypadku braku takiego warunku, w sytuacji powyżej,

funkcja jądra próbowała by się odwołać do nie istniejących komórek macierzy co spowodowałoby błąd w czasie uruchomienia. Dodatkowo zostały użyte pomocnicze funkcję **getElement** oraz **setElement** które poprawiają czytelność kodu.

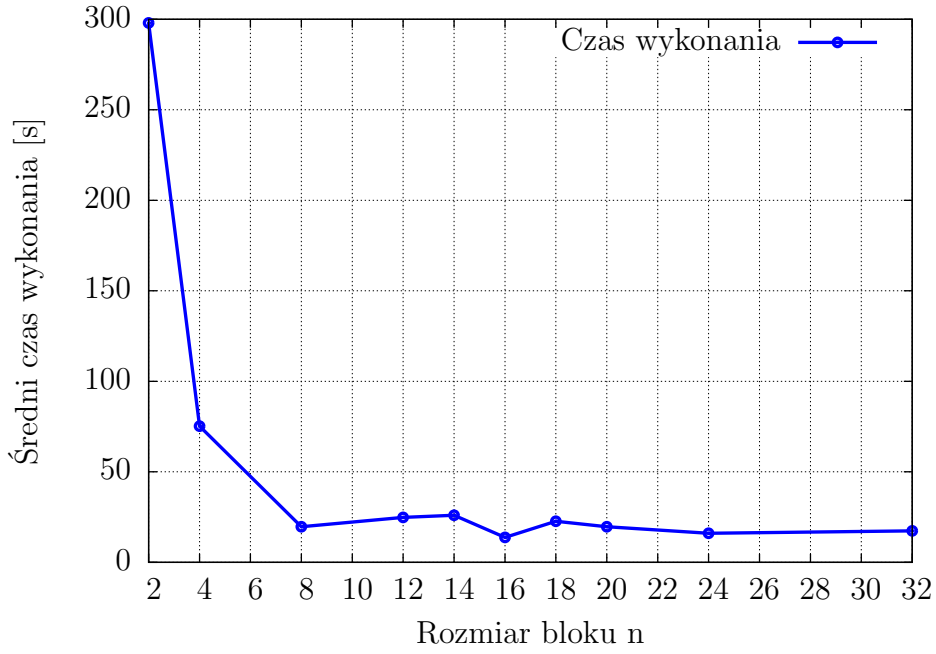
```
__global__ void matrixMultiplyKernel(Matrix A, Matrix B, Matrix C) {

   int row = threadIdx.y + blockIdx.y * blockDim.y;
   int col = threadIdx.x + blockIdx.x * blockDim.x;

   int matrixSize = A.matrixSize;

   float tempCValue = 0.0f;

   if (row < matrixSize && col < matrixSize){

      for(int k = 0; k < matrixSize; k++) {
         tempCValue += GetElement(A, row, k) * GetElement(B, k, col);
      }

      SetElement(C, row, col, tempCValue);
   }
}
```

Listing kodu 2: Główna funkcja jądra GPU - Mnożenie macierzy.

Wyniki i wnioski

Testy zostały przeprowadzone na serwerze cuda.iti.pk.edu.pl, przy użyciu jednej karty graficznej Nvidia®GTX480. Z wykresu możemy zauważyć, że największą wydajność uzyskujemy dla bloku o wymiarach 16×16.

Maksymalny rozmiar bloku jaki możemy utworzyć to taki który posiada maksimum 1024 wątki (np 32×32). Najlepszy czas uzyskaliśmy dla bloku rozmiarów: 16×16 wywołanie samej funkcji jądra odpowiedzialnej za wymnożenie macierzy to około 13772ms czyli 13s. Najgorszy czas został uzyskany dla rozmiaru bloku 2×2, jest to związane z tym, że jednostki procesujące musiały uruchamiać maksimum 4 wątki na blok, co jest niewykorzystywaniem zasobów jednostek streamujących SM. (SM są przystosowane do dużej ilości wątków gdyż są one zarządzane sprzętowo i nie występuje narzut związany z np. wywłaszczaniem, tworzeniem, zarządzaniem).



Rysunek 1: Wykres zależności czasu wykonania od rozmiaru bloku $n \times n$. Dla macierzy A, B, C o wymiarach 8192×8192 elementów.