Grupa 4	Barbara Nowak, Piotr Tomaszewski	25/11/2016

Zadanie 3 - Mnożenie macierzy CUDA

Celem naszego zadania było obliczenie iloczynu dwóch macierzy kwadratowych ale z użyciem karty graficznej i technologii CUDA.

Program uruchamiany jest z argumentem: „rozmiarMac ”- to rozmiar macierzy. Pierwszym etapem jest sprawdzenie poprawności podanego rozmiaru macierzy. Początkowo zaimplementowaliśmy funkcję gpuErrchk(), która sprawdza poprawnosc dzialania GPU i w razie ewentualnych problemów wypisuje adekwatny komunikat z kodem błędu. Następnie podobnie jak w zadaniu 1, deklarujemy potrzebne zmienne oraz macierze. Już w pierwszym etapie zauważamy, że CUDA posiada własny typ do deklaracji zmiennych czasowych. Ważną rzeczą w CUDA jest alokowanie pamięci na karcie graficznej za pomocą funkcji cudaMalloc():

```
1    gpuErrchk(cudaMalloc((void**) &d_A, sizeof(float)*rozmiarMac*rozmiarMac)
         );
2    gpuErrchk(cudaMalloc((void**) &d_B, sizeof(float)*rozmiarMac*rozmiarMac)
         );
3    gpuErrchk(cudaMalloc((void**) &d_C, sizeof(float)*rozmiarMac*rozmiarMac)
         );
```

Deklarując rozmiar grida oraz ilość wątków w bloku, korzystamy ze specjalnej zmiennej typu dim3, która reprezentuje krotkę 3-wymiarową, jakiej używamy do określenia liczby uruchomionych bloków, choć tak naprawdę tworzymy siatkę 2-wymiarową. Ostatni (trzeci) element instrukcji posiada domyślną wartość 1, dzięki czemu program działa poprawnie).

```
1    dim3 grids(rozmiarGrid, rozmiarGrid);
2    dim3 blocks(rozmiarBlok, rozmiarBlok);
```

Wywołując funkcję uzupełniającą macierze, musimy pamiętać aby zmienne typu dim3 przekazać do systemu wykonawczego «¡grids, blocks»¿ a następnie zablokować bieżący wątek aplikacji do czasu zakończenia wszystkich oczekiwanych obliczeń na karcie graficznej.

```
1    uzupelnij <<<grids, blocks>>> (rozmiarMac, d_A, d_B);
2    cudaDeviceSynchronize();
```

Podobnie jak w poprzednich zadaniach musimy przystąpić do zmierzenia czasu wykonywanych obliczeń, tutaj CUDA również posiada własne wbudowane funkcje. Etapy: uruchamiamy pomiar czasu; wywołujemy funkcję odpowiedzialną za obliczenia; ponownie blokujemy bieżący wątek aplikacji do czasu zakończenia wszystkich oczekiwanych obliczeń na karcie graficznej; zatrzymujemy licznik czasu. Po zakończeniu pomiarów musimy skopiować dane między karta graficzną a pamiecia RAM, do czego zastosujemy funkcję cudaMemcpy, gdzie kolejne parametry to odpowiednio: wskaźnik na obszar pamięci, do której nastąpi kopiowanie, wskaźnik na obszar pamięci, z której nastąpi kopiowanie, liczba bajtów do skopiowania oraz wybór kierunku kopiowania - w tym przypadku kopiujemy z karty graficznej do pamięci RAM komputera.

```
1    cudaEventRecord(czasStart, 0);
2    obliczC <<<grids, blocks>>> (rozmiarMac, d_A, d_B, d_C); //
3    cudaDeviceSynchronize();
```

```
4     cudaEventRecord(czasStop, 0);
5     cudaEventSynchronize(czasStop);
6
7     gpuErrchk(cudaMemcpy(C, d_C , sizeof(float)*rozmiarMac*rozmiarMac,
          cudaMemcpyDeviceToHost));
```

Ostatnim krokiem jest zwolnienie wcześniej zaalokowanej pamięci na karcie graficznej oraz obliczenie czasu wykonywanych obliczeń za pomocą funkcji cudaEventElapsedTime.

```
1     cudaFree(d_A);
2     cudaFree(d_B);
3     cudaFree(d_C);
4     free(C);
5     gpuErrchk(cudaEventElapsedTime(&roznica, czasStart, czasStop));
```

Tabela obok przedstawia czasy [ms] w zależności od rozmiaru macierzy i rozmiaru bloku gridu. Dla rozmiaru grid x blok [1x64] czasy są porównywalne, nie zależnie jaki rozmiar macierzy byśmy zastosowali. Jednak przy zmianach w podziale i wzroście rozmiaru macierzy czasy znacznie wzrastają. Może nie jest to wzrost paraboliczny, jednak widać znaczny przyrost czasu między podziałami [8x8] a [16x4].

		Rozmiar macierzy						
		32	*100*	*128*	*1024*	*2048*	*4096*	*8192*
podział grid x blok	*1x64*	6,368	6,272	6,304	6,4	6,432	6,368	6,336
	2x32	22,911	61,728	65,056	489,783	968,927	1935,104	3913,664
	4x16	21,312	50,655	60,063	469,056	935,455	1881,886	3802,239
	8x8	22,848	53,279	65,215	510,88	1021,823	2047,615	4144,416
	16x4	23,68	122,848	183	1227,135	2712,358	6055,231	12227,9
	32x2	34,784	272,864	341,727	2916,479	5868,768	16097,06	32503,39
	64x1	74,208	865,599	1191,871	9453,792	18941,25	54135,14	119196,4

Rysunek 1: Tabela zależności czasów od rozmiaru macierzy i gridu x bloku.

Wykres przedstawia czas obliczeń dwóch wybranych rozmiarów macierzy na procesorze graficznym w zależności od podziału bloku i gridu. Tutaj potwierdzają się słowa które napisaliśmy powyżej, że dla podziału [16x4] czas obliczeń się wydłuża. Im większy rozmiar macierzy, tym czas będzie jeszcze większy. Jeślibyśmy narysowali wykresy z wszystkimi rozmiarami macierzy, to zauważylibyśmy, że macierze o podobnych rozmiarach mają podobne czasy, co w porównaniu z wielkimi rozmiarami powoduje nałożenie na siebie linii wykresu sprawiając, że wykres jest mało czytelny.

Rysunek 2: Wykres porównania czasów 2 macierzy.

Podsumowanie: Podczas wykonywania zadania przy projekcie napotkaliśmy się na problem związany z odczytywaniem błędów. Bywało, że czasy wykonywania wychodziły 0 ms. Na początku problem wyniknął z niepoprawnej konwersji typów zmiennych. Później problem pojawił się przy odczytywaniu macierzy o dużych rozmiarach. Okazało się, że typ double zajmuje zbyt dużo pamięci, czego wynikiem był błąd. Zmieniliśmy typy macierzy z double na float i wyniki zaczęły wychodzić poprawne. Warto zabezpieczyć kod poprzez odczytywanie błędów powstałych w karcie graficznej, ponieważ dzięki temu mogliśmy zobaczyć co tak naprawdę jest nie tak.