Sprawozdanie z laboratorium: Przetwarzanie równoległe

Temat: 13. Przetwarzanie bez rozbieżności wątków, dostęp do danych nieefektywny i efektywny - pamięć globalna.

Prowadzący: dr hab. inż. Rafał Walkowiak

Michał Żurkowski indeks: 122480
Paweł Kaczmarek indeks: 122496

1. Wstęp

Celem sprawozdania jest porównanie dwóch metod sumowania wektora. Jedna polega na dostępie nieefektywnym do danych, natomiast druga efektywnym. Należy wspomnieć, że w naszym przypadku korzystamy z pamięci globalnej oraz zakładamy, że przetwarzanie odbywa się bez rozbieżności wątków.

2. Architektura GPU

Testy zostały wykonane na platformie wyposażonej w kartę Nvidia GTX 980 Ti. Poniżej przedstawiamy jej parametry:

- 22 multiprocesorów
- 2816 rdzeni
- 1024 Max wątków na blok
- 2048 Max wątków na multiprocesor
- compute capability: 5.2
- zegar rdzenia: 1420 MHz
- zegar pamięci: 7200 MHz
- Warp Size 32
- 1024 x 1024 x 64 Max rozmiar bloku

Do wykonania testów korzystaliśmy z programu NVIDIA Visual Profiler.

3. Implementacje algorytmów

3.1. Metoda z nieefektywnym dostępem do pamięci

```
template <int BLOCK_SIZE> __global__ void sumKernelStr2(float* c, float* a)  {
  unsigned int tid = 2 * threadIdx.x + 2 * blockDim.x * blockIdx.x;

  // Wyznaczamy które komórki tablicy mają zostać zsumowane w kolejnych iteracjach
  for (unsigned int odstep = 1; odstep < 2 * blockDim.x; odstep *= 2)  {
    if (tid + odstep < arraySize)
      a[tid] += a[tid + odstep]; // Zapisz wynik sumowania do wcześniejszej komórki
    __syncthreads();
  }

  // Wątek o indeksie 0 zapisuje ostateczny wynik.
  if (threadIdx.x == 0) c[blockIdx.x] = a[2 * blockIdx.x * blockDim.x];
}
```

3.2. Metoda z efektywnym dostępem do pamięci

```
template <int BLOCK_SIZE> __global__ void sumKernelStr3(float* c, float* a)  {
  unsigned int tid = threadIdx.x + 2 * blockDim.x * blockIdx.x;

   // Wyznaczamy które komórki tablicy mają zostać zsumowane w kolejnych iteracjach
  for (unsigned int odstep = blockDim.x; odstep > 0; odstep /= 2)  {
    if (tid + odstep < arraySize)
      a[tid] += a[tid + odstep]; // Zapisz wynik sumowania do wcześniejszej komórki
    __syncthreads();
  }

  // Wątek o indeksie 0 zapisuje ostateczny wynik.
  if (threadIdx.x == 0) c[blockIdx.x] = a[2 * blockIdx.x * blockDim.x];
}
```

4. Teoria

W rozważanym przez nas zadaniu korzystamy z pamięci globalnej. Dostęp do niej jest zdecydowanie wolniejszy niż dostęp do pamięci współdzielonej (około 100 razy wolniejszy). Jest ona jednak dużo większa niż pamięć współdzielona. Zmienne przechowywane w pamięci globalnej są dostępne dla wszystkich wątków, przez cały czas życia aplikacji.

4.1. CGMA

CGMA (compute to global memory access ratio) jest to parametr, który określa jak długo zajmujemy procesor pobranymi z pamięci globalnej danymi. Liczony jest on jako stosunek liczby operacji zmiennoprzecinkowych do liczby dostępów do pamięci globalnej. Miara ta jest wykorzystywana do optymalizacji algorytmu. Miarę tą należy maksymalizować w celu zwiększenia przepustowości pamięci. Dlatego warto czasem jest wykonać obliczenia na GPU niż pobierać dane z pamięci globalnej. Jest to spowodowane tym, że GPU dostarcza nam przede wszystkim wiele jednostek arytmetyczno-logicznych (ALU), a nie szybkiej pamięci.

Do obliczenia wartości praktycznej CGMA wykorzystaliśmy wzór:

$$CGMA = \frac{Floating\ Point_{Instructions}}{Global_{Store} + Global_{Load}}$$

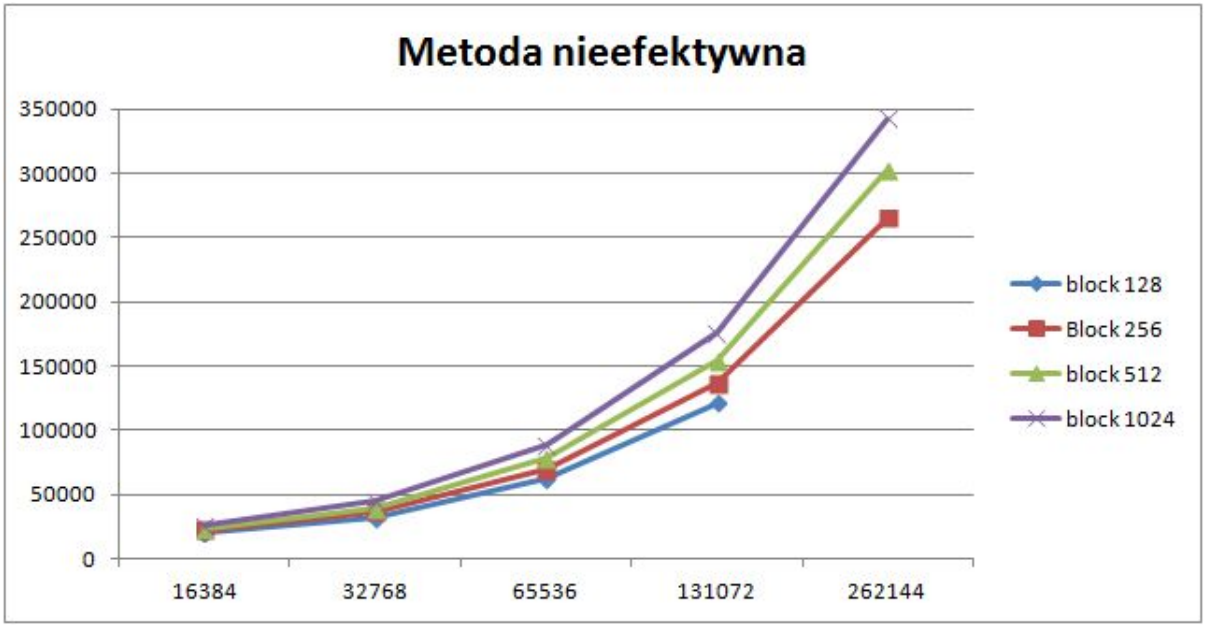
Praktyczne wartości CGMA które otrzymaliśmy wahały się między 10.4 - 10.7 w zależności od wielkości bloku, im większy tym większe CGMA.

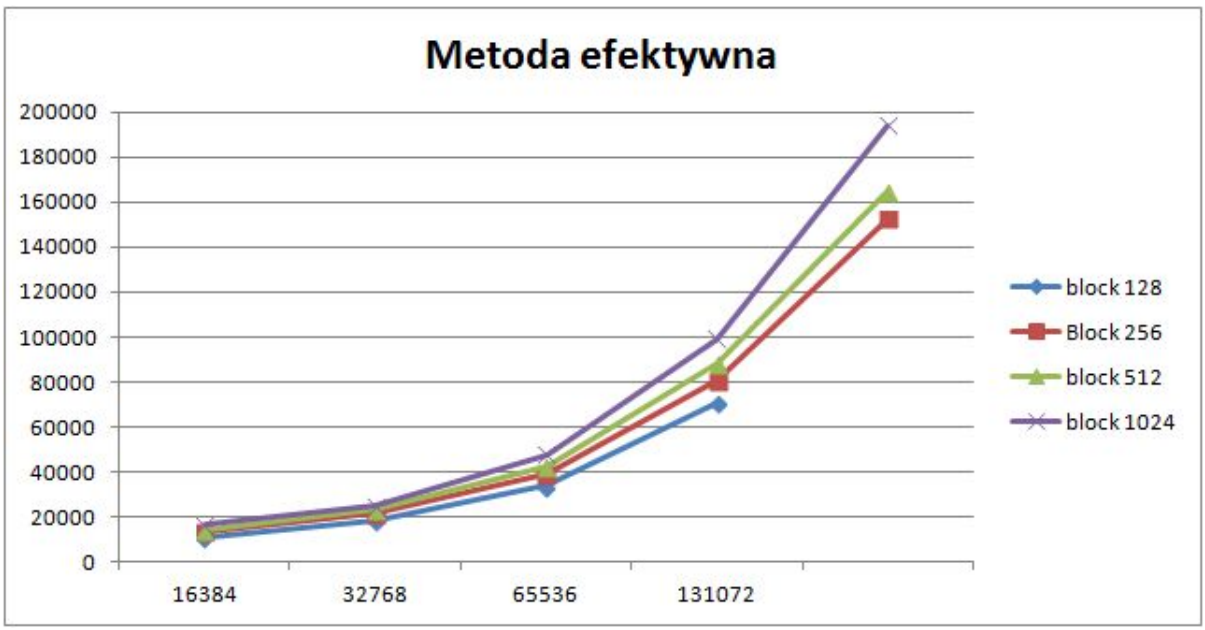
5. Pomiary

Ta część sprawozdania przeznaczona jest na analizę parametrów otrzymanych podczas badań wykonanych przy pomocy Profilera.

5.1. Analiza czasowa

Poniższe wykresy przedstawiają czasy przetwarzania obu metod w zależności od instancji.





Na podstawie zaprezentowanych wykresów możemy stwierdzić, że czas przetwarzania dla metody z efektywnym dostępem do danych jest znacznie lepszy niż metody z nieefektywnym. Kolejnym zaobserwowanym faktem jest to, że wraz ze wzrostem wielkości bloku czas przetwarzanie zwiększa się. Ostatnim, a zarazem najbardziej oczywistym uzyskanym wnioskiem jest wzrost czasu przetwarzania wraz ze wzrostem wielkości tablicy.

5.2. Przepustowość przetwarzania

Kolejnym istotnym badanym przez nas parametrem jest przepustowość przetwarzania. Badaliśmy ją dla kilku różnych wymiarów bloków oraz tablic. Wyniki eksperymenty zostały przedstawione w poniższej tabeli.

Block	Array	Performance[Gflop/s]	
		nieefektywny	efektywny
128	1024	7,58	12,19
128	512	6,65	10,97
128	256	5,5	8,09
128	128	3,84	5,35
256	2048	7,47	12,61
256	1024	6,92	11,1
256	512	6,05	9,76
256	256	5,44	8,51
256	128	3,98	5,79
512	2048	6,58	11,79
512	1024	6,23	10,92
512	512	5,89	9,93
512	256	5,11	8,17
512	128	3,75	5,57
1024	2048	5,93	10,73
1024	1024	5,63	9,81
1024	512	5,49	9,22
1024	256	4,7	7,71
1024	128	3,44	5,23

Na podstawie tabeli można zauważyć, że wzrost wielkości Array powoduje wzrost przepustowości. Jest to pierwszy wniosek jaki uzyskaliśmy na podstawie tej tabeli. Kolejnym faktem jest to, że wraz ze wzrostem wymiaru bloku wątków przepustowość maleje. Obie powyższe relacje zachodzą zarówno dla metody efektywnym dostępem do danych jak i dla tej z nieefektywnym dostępem. Powyższa tabela potwierdza również, że metoda efektywna jest znacznie lepsza jeżeli chodzi o przepustowość przetwarzania.

5.3. Pamięć globalna

Kolejnymi parametrami uznanymi przez nas za istotne dla naszego zadanie są efektywność pobierania danych z pamięci globalnej oraz efektywność zapisywania danych w tejże pamięci. Uzyskane wyniki zostały zaprezentowane w poniższej tabeli.

Block	Array	Gl. Mem. L. Eff.(%)		Gl. Mem. S. Eff.(%)	
		nieefektywny	efektywny	nieefektywny	efektywny
128	1024	49,16	95,214	49,854	99,322
128	512	49,16	95,214	49,854	99,322
128	256	49,16	95,214	49,854	99,322
128	128	49,16	95,214	49,854	99,322
256	2048	49,284	95,861	49,935	99,697
256	1024	49,284	95,861	49,935	99,697
256	512	49,284	95,861	49,935	99,697
256	256	49,283	95,861	49,935	99,697
256	128	49,283	95,861	49,935	99,697
512	2048	49,368	96,322	49,971	99,863
512	1024	49,368	96,322	49,971	99,863
512	512	49,368	96,322	49,971	99,863
512	256	49,368	96,322	49,971	99,863
512	128	49,368	96,322	49,97	99,863
1024	2048	49,432	96,674	49,987	99,938
1024	1024	49,432	96,674	49,987	99,938
1024	512	49,432	96,674	49,987	99,938
1024	256	49,431	96,674	49,987	99,938
1024	128	49,431	96,674	49,986	99,938

Na podstawie tabeli możemy stwierdzić, że wielkość instancji nie ma większego wpływu na te dwa parametry. Jednak najważniejszym wnioskiem uzyskanym na podstawie powyższych danych jest to, że metoda z efektywnym dostępem do pamięci jest prawie dwukrotnie lepsza od tej nieefektywnej na obu rozważanych parametrach.

5.4. Zajętość multiprocesora

Pierwszym wykonanym przez nas badaniem w tym punkcie było teoretyczne przeliczenie zajętości multiprocesora w zależności od wielkości bloku. Dla każdej rozważanej przez nas wartości wyszła nam zajętość multiprocesora równa 100%. Obliczenia teoretyczne wykonaliśmy przy pomocy CUDA Occupancy Calculator.
Kolejnym etapem było zbadanie zajętości multiprocesora przy pomocy Profiliera. Wyniki naszych eksperymentów przedstawione zostały poniżej.

Block	Array	Warp Execution Efficiency(%)		Multiprocessor Activity(%)	
		nieefektywny	efektywny	nieefektywny	efektywny
128	1024	98,77	98,59	95,55	94,25
128	512	98,77	98,57	91,94	89,79
128	256	98,77	98,59	86,27	80,29
128	128	98,77	98,59	68,19	71,71
256	2048	99,44	99,36	98,34	97,23
256	1024	99,44	99,36	96,52	94,35
256	512	99,44	99,36	92,17	91,71
256	256	99,44	99,36	88,59	83,95
256	128	99,44	99,36	69,75	70,16
512	2048	99,75	99,71	98,08	97,42
512	1024	99,75	99,71	95,82	95,41
512	512	99,75	99,71	92,42	90,99
512	256	99,75	99,71	89,29	84,39
512	128	99,74	99,71	70,62	66,26
1024	2048	99,88	99,87	98,29	97,31
1024	1024	99,88	99,87	96,00	95,52
1024	512	99,88	99,87	92,56	91,57
1024	256	99,89	99,87	89,53	83,49
1024	128	99,88	99,87	71,06	69,05

Z tabeli zawierającej wyniki widać że zajętość multiprocesora rośnie z każdym wzrostem wielkości sumowanej tablicy.

6. Wnioski końcowe

Tematem naszego zadanie było porównanie metod z efektywnym oraz nieefektywnym dostępem do danych. Najważniejszym wnioskiem uzyskanym na podstawie tego zadania jest to że efektywna metoda dostępu jest dużo lepsza na wszystkich omawianych przez nas parametrach. Minusem jednak okazuje się korzystanie z pamięci globalnej, przez co parametr CMGA jest dosyć niski. Można byłoby go znacznie poprawić poprzez zastosowanie pamięci współdzielonej.