Badanie wpływu organizacji dostępu do pamięci globalnej na efektywność przetwarzania - Nvidia CUDA

Sebastian Maciejewski 132275 i Jan Techner 132332
grupa I1, zajęcia we wtorki o 15:10, tygodnie nieparzyste,
email: maciejewski.torun@gmail.com

11 czerwca 2019

1 Wstęp

Sprawozdanie dotyczy zadania w wariancie 5. - badanie wpływu organizacji dostępu do pamięci globalnej (dostęp łączony i dostęp niełączony) na efektywność przetwarzania.
Warianty kodu:
- grid wieloblokowy, obliczenia przy wykorzystaniu pamięci globalnej,
- grid wieloblokowy, obliczenia przy wykorzystaniu pamięci współdzielonej bloku wątków.

Każdy z tych wariantów był wykorzystywany w dwóch wersjach - z łączonym oraz niełączonym dostępem do pamięci. Zastosowaliśmy dwa rozmiary bloków - 8x8 i 16x16, a pomiary pobraliśmy dla instancji o różnych wielkościach (128x128, 256x256, 512x512, 1024x1024).
Wszystkie pomiary zostały wykonane na komputerach laboratoryjnych w sali 2.7.6. na karcie Nvidia GTX 260.

1.1 Specyfikacja sprzętu i wykorzystanego oprogramowania

- karta graficzna Nvidia GTX 260
  - compute capability - 1.3
  - liczba multiprocesorów - 27
  - maksymalna liczba bloków na jeden multiprocesor - 8
  - maksymalna liczba wątków w bloku - 512
  - warp size - 32
  - half-warp size - 16
- Visual Studio 2013
- Nvidia Visual Profiler

2 Analiza przygotowania eksperymentu

2.1 Kod wykorzystywany przy pomiarach

Obliczenia z wykorzystaniem pamięci globalnej

W pierwszym wariancie kodu do obliczeń wykorzystujemy pamięć globalną - fragmenty pamięci zarezerwowane na każdą z 3 macierzy za pomocą cudaMalloc. Każdy z wątków oblicza jeden element wynikowy dodając wartości w swoim lokalnym rejestrze (C_local), po czym zapisuje wynik takiego przetwarzania do tablicy w pamięci globalnej.

Obliczenia z wykorzystaniem pamięci współdzielonej bloku wątków

W drugim wariancie wykorzystujemy współdzieloną pamięć bloku wątków. Pewien obszar pamięci został zaalokowany na współdzielony obszar pamięci bloku wątków za pomocą __shared__. Dwa takie obszary, o rozmiarze 8x8 lub 16x16 (w zależności od pomiaru) odpowiadają za przechowywanie fragmentów mnożonej macierzy. Dla każdego bloku wątków dane są pobierane do jego pamięci współdzielonej, następnie każdy z wątków wylicza na ich podstawie swój wynik.

Taka organizacja przetwarzania mogłaby prowadzić do problemów z np. nadpisywaniem danych, więc konieczka jest synchronizacja - aby "poczekaćńa zakończenie przetwarzania przez wszystkie wątki, wywoływana jest funkcja __syncthreads. Po zakończeniu przetwarzania, wynik jest zapisywany w pamięci globalnej.

2.1.1 Rodzaje dostępu do pamięci

Dostęp do pamięci w kartach graficznych Nvidia odbywa się za pomocą transkacji o rozmiarach 32, 64 lub 128 bitów. W karcie na której przeprowadzany był eksperyment (CC 1.3) takie dostępy są realizowane dla tzw. half-wrapów - 16 wątkowych grup.
Różnica między dostępem łączonym a niełączonym sprowadza się do tego, że dostępem łączonym nazywamy dostęp do pamięci tych wątków half-wrapu, które sąsiadują ze sobą w pamięci. Dostęp w przeciwnym wypadku (wątki nie siąsiadują ze sobą w pamięci) nazywany jest dostępem niełączonym.

2.2 Istotne fragmenty kodu

Dostęp łączony, pamięć globalna

```
1  template <int BLOCK_SIZE> __global__ void matrixMulCUDA(float *C, float *A,
2  float *B, int wA,
3  int wB) {
4  // Block index
5  int bx = blockIdx.x;
6  int by = blockIdx.y;
7
8  // Thread index
9  int tx = threadIdx.x;
10 int ty = threadIdx.y;
11
12 int row = by * blockDim.y + ty;
13 int col = bx * blockDim.x + tx;
14 float C_local = 0;
15
16 for (int k = 0; k < wA; k++) {
17  C_local += A[row*wA + k] * B[k *wA + col];
18 }
19
20 C[row*wA + col] = C_local;
21 }
```

Dostęp niełączony, pamięć globalna

```
1  template <int BLOCK_SIZE> __global__ void matrixMulCUDA(float *C, float *A,
2  float *B, int wA,
3  int wB) {
4  // Block index
5  int bx = blockIdx.x;
6  int by = blockIdx.y;
7
8  // Thread index
9  int tx = threadIdx.x;
10 int ty = threadIdx.y;
11
12 int col= by * blockDim.y + ty;
13 int row = bx * blockDim.x + tx;
14 float C_local = 0;
15
16 for (int k = 0; k < wA; k++) {
17  C_local += A[row*wA + k] * B[k *wA + col];
18 }
19
20 C[row*wA + col] = C_local;
21 }
```

Dostęp łączony, pamięć współdzielona bloku wątków

```
1  template <int BLOCK_SIZE> __global__ void matrixMulCUDA(float *C, float *A,
2  float *B, int wA,
3  int wB) {
4  // Block index
5  int bx = blockIdx.x;
6  int by = blockIdx.y;
7
8  // Thread index
9  int tx = threadIdx.x;
10 int ty = threadIdx.y;
11
12 int row = by * blockDim.y + ty;
13 int col = bx * blockDim.x + tx;
14 float C_local = 0;
15
16 __shared__ float matAhelp[BLOCK_SIZE][BLOCK_SIZE];
17 __shared__ float matBhelp[BLOCK_SIZE][BLOCK_SIZE];
18 for (int m = 0; m < wA / BLOCK_SIZE; ++m) {
19  matAhelp[tx][ty] = A[row * wA + m*BLOCK_SIZE + tx];
20  matBhelp[tx][ty] = B[(m*BLOCK_SIZE + ty)*wA + col];
21  __syncthreads();
22  for (int k = 0; k < BLOCK_SIZE; ++k)
23   C_local += matAhelp[tx][k] * matBhelp[k][ty];
24  __syncthreads();
```

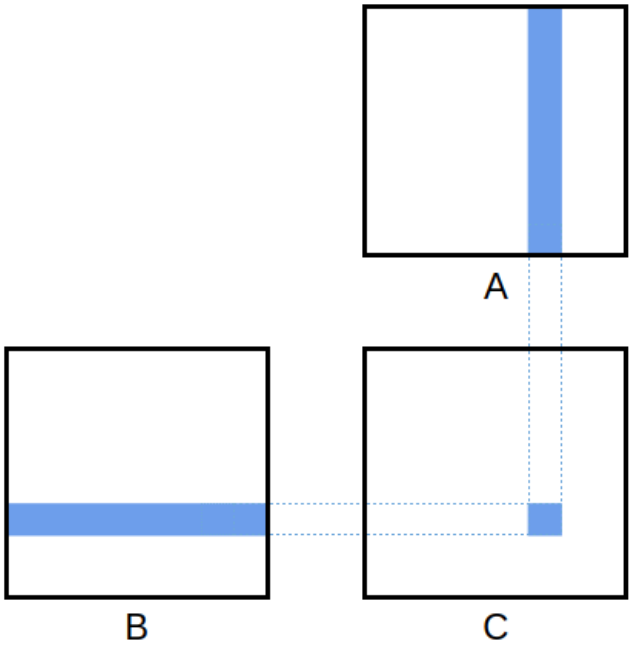
```
25   }
26
27   C[row*wA + col] = C_local;
28  }
```

Dostęp niełączony, pamięć współdzielona bloku wątków

```
1   template <int BLOCK_SIZE> __global__ void matrixMulCUDA(float *C, float *A,
2   float *B, int wA,
3   int wB) {
4   // Block index
5   int bx = blockIdx.x;
6   int by = blockIdx.y;
7
8   // Thread index
9   int tx = threadIdx.x;
10  int ty = threadIdx.y;
11
12  int col = by * blockDim.y + ty;
13  int row = bx * blockDim.x + tx;
14  float C_local = 0;
15
16  __shared__ float matAhelp[BLOCK_SIZE][BLOCK_SIZE];
17  __shared__ float matBhelp[BLOCK_SIZE][BLOCK_SIZE];
18  for (int m = 0; m < wA / BLOCK_SIZE; ++m) {
19   matAhelp[tx][ty] = A[row * wA + m*BLOCK_SIZE + tx];
20   matBhelp[tx][ty] = B[(m*BLOCK_SIZE + ty)*wA + col];
21   __syncthreads();
22   for (int k = 0; k < BLOCK_SIZE; ++k)
23    C_local += matAhelp[tx][k] * matBhelp[k][ty];
24   __syncthreads();
25  }
26
27  C[row*wA + col] = C_local;
28 }
```

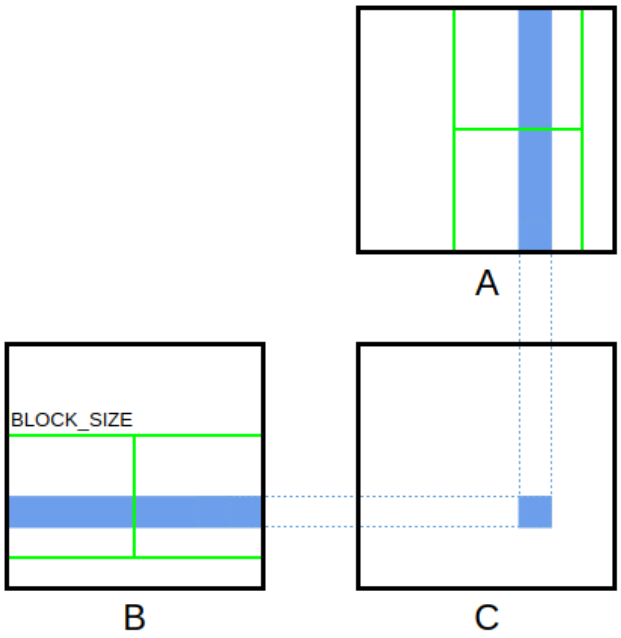
Wizualizacja przetwarzania

Na poniższych grafikach widać różnicę w procesie obliczania macierzy z wykorzystaniem pamięci globalnej i pamięci współdzielonej wątków.



Rysunek 1: Wykorzystanie pamięci globalnej

W przypadku pamięci globalnej każdy wątek odczytuje jeden wiersz macierzy A i jedną kolumnę macierzy B, a następnie oblicza odpowiadający element macierzy C. Wszystkie dane odczytywane są bezpośrednio z pamięci globalnej, co powoduje bardzo dużą liczbę odwołań do pamięci globalnej.



Rysunek 2: Wykorzystanie pamięci podręcznej, dostęp niełączony

W drugim przypadku, dla pamięci współdzielonej bloku wątków, każdy blok oblicza macierz będącą częścią macierzy wynikowej C o rozmiarze równym rozmiarowi bloku. Każdy wątek jest wówczas odpowiedzialny za obliczenie jednego elementu takiego wycinka macierzy. W taki sposób znacząco ograniczamy ilość dostępów do pamięci globalnej, ponieważ macierze A i B są odczytywane tylko $\frac{n}{BLOCK\_SIZE}$ razy, gdzie n to wielkość instancji a $BLOCK\_SIZE$ to wielokść bloku.

3 Analiza wyników eksperymentu pomiarowego

3.1 Miary efektywności

Prędkość obliczeń

$$\frac{ZL}{C} = \frac{2 * n^3}{C} \tag{1}$$

gdzie ZL to złożoność obliczeniowa, C to czas a n to rozmiar instancji.

Przyspieszenie w stosunku do IKJ

$$\frac{C\_IKJ}{CPR} \tag{2}$$

gdzie C_IKJ to czas przetwarzania IKJ a CPR to czas przetwarzania równoległego.

Przyspieszenie w funkcji wielkości instancji

$$\frac{CPI}{CP128} \tag{3}$$

gdzie CPI to czas przetwarzania dla instancji a $CPI128$ to czas przetwarzania dla instancji o rozmiarze 128, będący naszym punktem odniesienia - obliczamy przyspieszenie w stosunku do przetwarzania instancji o rozmiarze 128.

Miara stopnia łączenia dostępów do pamięci

Jest to miara obliczana w następujący sposób: dla każdebo bloku pobierane będą dwie macierze (n^2), dodajemy do tego n^2 zapisów do pamięci globalnej i dzielimy przez sumę transkacji między pamięcią a multiprocesorem (odczytane z wyników z programu Visual Profiler).

Zajętość procesora

Wyliczana za pomocą kalkulatora zajętości SM miara - dla bloków o rozmiarze 8x8 osiąga 50%, zaś dla bloków o rozmiarze 16x16 osiąga 100%, czego można się było spodziewać. Pokazuje to, że pełnię możliwości tej karty wykorzystujemy wtedy, gdy bloki mają rozmiar 16x16.

CGMA - stosunek operacji do dostępów do pamięci

Jest to miara instensywności obliczeń. Zakładamy, że do obliczenia jednego elementu wynikowego potrzebujemy n danych z wiersza macierzy, n danych z kolumny macierzy oraz 1 operację zapisu do pamięci globalnej.
Warto zauważyć, że dla współdzielonej pamięci bloku wątków CGMA będzie bliski 1, ponieważ pomijamy operacje pobierania danych z pamięci globalnej do lokalnej.
gdzie n to wielkość instancji.

GLD_EFFICIENCY

Jest to miara efektywności pobierania danych z pamięci globalnej obliczana w sposób następujący:

$$\frac{gld\_reques}{gld\_total/(2 * SM)} \tag{4}$$

gdzie SM to liczba multiprocesorów.

GST_EFFICIENCY

Jest to miara efektywności zapisu danych w pamięci globalnej. Oblicza się ją analogicznie do GLD_EFFICIENCY.

$$\frac{gst\_reques}{(gst\_32 + gst\_64 + gst\_128)/(2 * SM)} \tag{5}$$

gdzie SM to liczba multiprocesorów.

3.2 Wyniki pomiarów

V	BS	INST	GFLOPS	vs IKJ	ΔA	CMGA	JOIN	%	GLD_E	GST_E
G_L	8	128	25,32073626	12,0739	1,0000	0,0039	733,8826	50%	5184	141208
		256	30,63327366	2,7388	6,6126	0,0019	1512,4980	50%	10368	10904
		512	26,32677471	2,6480	61,5543	0,0010	3045,1146	50%	20736	591
		864	22,22919876	2,4298	350,3191	0,0006	5162,9016	50%	34992	59
		1024	24,66069324	5,8681	525,7037	0,0005	6120,9971	50%	41472	35
	16	128	33,95345298	16,1903	1,0000	0,0039	245,8109	100%	1728	282894
		256	32,58490086	2,9133	8,3360	0,0019	552,7462	100%	3456	17551
		512	34,41340787	3,4614	63,1446	0,0010	1136,8067	100%	6912	1141
		864	32,75306701	3,5801	318,8183	0,0006	1930,8138	100%	11664	133
		1024	29,89612411	7,1139	581,4857	0,0005	2286,2996	100%	13824	63
G_NL	8	128	10,11192223	4,8217	1,0000	0,0039	242,4433	50%	3888	77004
		256	9,662213325	0,8639	8,3723	0,0019	501,8743	50%	7776	4603
		512	6,010390691	0,6045	107,6740	0,0010	1012,7101	50%	15552	181
		864	6,303300587	0,6890	493,3749	0,0006	1718,6194	50%	26244	23
		1024	4,254822746	1,0124	1216,8084	0,0005	2037,9817	50%	31104	8
	16	128	5,148311945	2,4549	1,0000	0,0039	29,1321	100%	3672	41510
		256	5,148414636	0,4603	7,9998	0,0019	65,2732	100%	7344	2595
		512	3,569637799	0,3590	92,3040	0,0010	133,9956	100%	14688	112
		864	3,316741109	0,3625	477,3804	0,0006	227,4108	100%	24786	13
		1024	2,167744905	0,5158	1215,9806	0,0005	269,2327	100%	29376	4
S_L	8	128	33,09585582	15,7813	1,0000	1	3879,0936	50%	864	1112210
		256	38,81404571	3,4702	6,8214	1	8493,2578	50%	1728	82978
		512	39,20490797	3,9433	54,0273	1	17661,6646	50%	3456	5254
		864	23,68074399	2,5885	429,8229	1	25701,0879	50%	6912	330
		1024	65,60469794	15,6108	258,2906	1	42639,0980	50%	5832	652
	16	128	17,17252759	8,1885	1,0000	1	2352,7619	100%	0	2155375
		256	20,78304377	1,8581	6,6102	1	6582,7043	100%	0	170279
		512	21,7859839	2,1913	50,4472	1	15496,4703	100%	0	11433
		864	21,82694331	2,3858	241,9650	1	27996,8000	100%	0	1426
		1024	21,80557117	5,1887	403,2150	1	33641,2658	100%	0	724
S_NL	8	128	31,95027271	15,2351	1,0000	1	1429,1397	50%	540	1726347
		256	38,67665172	3,4580	6,6087	1	3247,4221	50%	1080	132073
		512	39,13595265	3,9364	52,2491	1	6899,0877	50%	2160	8390
		864	39,36259325	4,3026	249,6331	1	11969,1021	50%	3645	1041
		1024	35,79431614	8,5174	457,0150	1	14265,8722	50%	4320	487
	16	128	16,64049767	7,9348	1,0000	1	297,6386	100%	229,5	2118706
		256	20,38662765	1,8227	6,5300	1	810,5569	100%	459	164427
		512	21,71864653	2,1845	49,0358	1	1871,3958	100%	918	10961
		864	21,81985743	2,3851	234,5448	1	3348,4545	100%	1549,125	1358
		1024	21,79336548	5,1858	390,9417	1	4014,0147	100%	1836	688

Tablica 1: Miary efektywności dla 6 pętli, przetwarzanie równolegle

4 Wnioski

Po przeanalizowaniu wyników