Politechnika Poznańska

Wydział Informatyki

Przetwarzanie Równoległe

Programowanie CUDA na NVIDIA GPU

Autorzy:
Adam Szczepański
Mateusz Czajka

Prowadzący:
dr Rafał Walkowiak

27 lutego 2014

Spis treści

1 Informacje o projekcie

1.1 Dane autorów

Mateusz Czajka	106596
Adam Szczepański	106593

1.2 Historia projektu

1. Jest to pierwsza wersja projektu. Dokumentacja elektroniczna została przesłana w dniu 25 lutego 2014.

1.3 Parametry karty graficznej

Nazwa	GeForce GT 240
Typ RAM	DDR3
Frame Buffer Bandwidth (GB/s)	25.6
Graphics Clock (MHz)	575
Processor Clock (MHz)	1400
Memory Clock (MHz)	800
SM Count	12
CUDA Cores	96
MAX_THREADS_PER_BLOCK	512
MAX_BLOCK_DIM_X	512
MAX_BLOCK_DIM_Y	512
MAX_BLOCK_DIM_Z	64
MAX_GRID_DIM_X	65535
MAX_GRID_DIM_Y	65535
MAX_GRID_DIM_Z	1
MAX_SHARED_MEMORY_PER_BLOCK	16384
TOTAL_CONSTANT_MEMORY	65536
WARP_SIZE	32
MAX_REGISTERS_PER_BLOCK	16384
MULTIPROCESSOR_COUNT	12
Compute Capability	1.2
MAX_THREADS_PER_MULTIPROCESSOR	1024

Tablica 1: Parametry wykorzystanej karty graficznej GeForce GT 240.

2 Wprowadzenie

Celem projektu było zapoznanie się z możliwościami przetwarzania na kartach graficznych na przykładzie technologi CUDA.

Przygotowaliśmy 6 wersji programu, którego zadaniem było mnożenie macierzy kwadratowych na GPU:

1. wykorzystanie 1 bloku wątków
2. wykorzystanie gridu wieloblokowego wątków
3. wykorzystanie gridu wieloblokowego wątków i pamięci współdzielonej
4. wykorzystanie gridu wieloblokowego wątków i pamięci współdzielonej, zrównoleglone pobranie danych i obliczenia
   (a) pobranie danych do rejestru
   (b) pobranie danych do pamięci współdzielonej
5. wykorzystanie gridu wieloblokowego wątków i pamięci współdzielonej, zrównoleglone pobranie danych i obliczenia, powiększona ilość pracy każdego wątku

Dla każdej wersji podajemy teoretyczną zajętość SM wynikającą z rozmiaru bloku, wykorzystanej liczby rejestrów, rozmiaru pamięci współdzielonej, a także z ograniczeń GPU. Wyjątkiem jest wersja 1. w której wykorzystany jest tylko 1 blok, zatem zajętość SM wynika tylko z jego rozmiaru.
W przypadku pozostałych wersji, dla poszczególnych parametrów podany jest w tabeli limit bloków – oznacza on ile bloków maksymalnie można powiązać z SM przy danych parametrach. Jeśli wszystkie limity są większe od limitu bloków na SM wynikającego z ograniczeń GPU, to limit bloków GPU determinuje ilość aktywnych bloków, a co za tym idzie zajętość SM.

Efektywność programów zbadaliśmy także przy pomocy profilera NVIDIA Visual Profiler. Dla każdej instancji podajemy:

- czas wykonania
- ilość operacji zmienno przecinkowych na sekundę (GFLOPS)
- ilość instrukcji wykonanych na sekundę (GIPS)
- stosunek operacji zmienno przecinkowych do ilości operacji odczytu/zapisu z pamięci globalnej (CGMA)

Ze względu na zupełnie odmienne podejścia w wersjach 1, 2 i 3, oraz modyfikacje wersji 3. w wersjach 4a, 4b i 5 rozmiary macierzy i bloków podzieliliśmy na dwie grupy:

- Wersje 1, 2 i 3 – macierze 176x176, 352x352 oraz 528x528, bloki 8x8, 16x16, 22x22 (wymiary macierzy są podzielne przez 8, 16 i 22).
- Wersje: 3, 4a, 4b, 5 – macierze 128x128, 256x256, 384x384, 512x512, 640x640, bloki 8x8, 16x16 (wymiary macierzy podzielne przez 16 i 32).

Na zakończenie prezentujemy porównanie efektywności wszystkich badanych wersji.

3 Ocena efektywności przetwarzania

3.1 Wersja 1

3.1.1 Opis rozwiązania

W pierwszej wersji programu wykorzystany został tylko jeden blok wątków. Każdy z wątków oblicza

$$\left(\frac{\text{rozmiar macierzy}}{\text{rozmiar bloku}}\right)^2$$

elementów wyniku. Pamięć współdzielona nie jest wykorzystywana.

```
__global__ void MatrixMulKernel_1(const float* Ad, const float* Bd,
    float* Cd, const int WIDTH) {
  int tx = threadIdx.x;
  int ty = threadIdx.y;
  float C_local;

  for (int i=0; i<WIDTH/blockDim.y; i++) {
    for (int j=0; j<WIDTH/blockDim.x; j++) {
      C_local = 0.0f;
      for (int k = 0; k < WIDTH; ++k) {
        float A_d_element = Ad[i*WIDTH*blockDim.y + ty*WIDTH + k];
        float B_d_element = Bd[j*blockDim.y + k*WIDTH + tx];
        C_local += A_d_element * B_d_element;
      }

      Cd[i*WIDTH*blockDim.y + j*blockDim.y + ty*WIDTH + tx] = C_local;
    }
  }
}
```

Listing 1: Mnożenie macierzy kwadratowych na GPU – wersja 1.

3.1.2 Teoretyczna zajętość SM

Kryterium		Teoretyczna wartość			Limit GPU
		8x8	16x16	22x22	
Zajętość SM	Aktywne bloki	1	1	1	8
	Aktywne warpy	2	8	16	32
	Aktywne wątki	64	256	484	1024
	Zajętość	6.25%	25.00%	50.00%	100.00%
Warpy	Wątki/Blok	64	256	484	512
	Warpy/Blok	2	8	16	16
Rejestry	Rejestry/Wątek	16	16	16	128
	Rejestry/Blok	1024	4096	8192	16384
Pamięć współdzielona	Pamięć współdzielona/Blok	44	44	44	16384

Tablica 2: Teoretyczna zajętość SM – wersja 1.

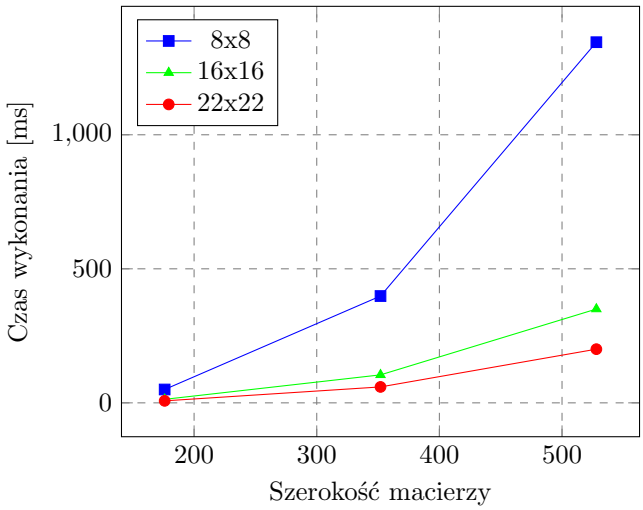
Wykorzystywany jest 1 blok, zatem zajętość jest zależna tylko od rozmiaru tego bloku.

3.1.3 Wyniki pomiarów

1. **Czas trwania obliczeń**

Rozmiar macierzy	Rozmiar bloku		
	8x8	16x16	22x22
176x176	49.860	13.156	7.454
352x352	398.426	104.408	59.424
528x528	1345.355	349.936	200.265

Tablica 3: Czas obliczeń [ms] – wersja 1.

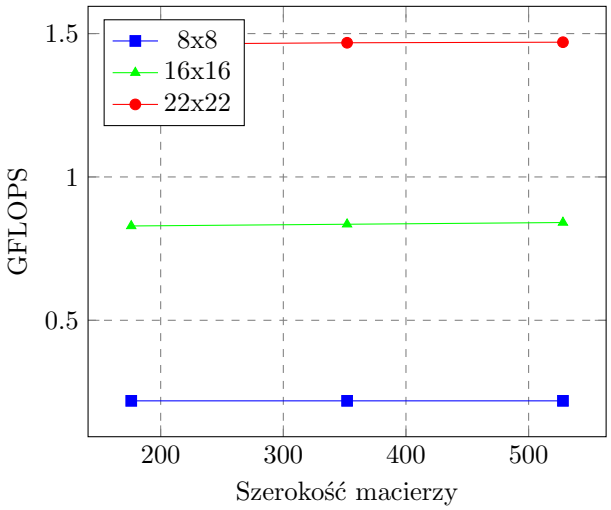


Rysunek 1: Zależność pomiędzy czasem obliczeń a rozmiarem macierzy – wersja 1.

2. **Ilość operacji zmiennoprzecinkowych na sekundę**

Rozmiar macierzy	Rozmiar bloku		
	8x8	16x16	22x22
176x176	0.219	0.829	1.463
352x352	0.219	0.835	1.468
528x528	0.219	0.841	1.470

Tablica 4: Ilość operacji zmiennoprzecinkowych na sekundę (GFLOPS) – wersja 1.

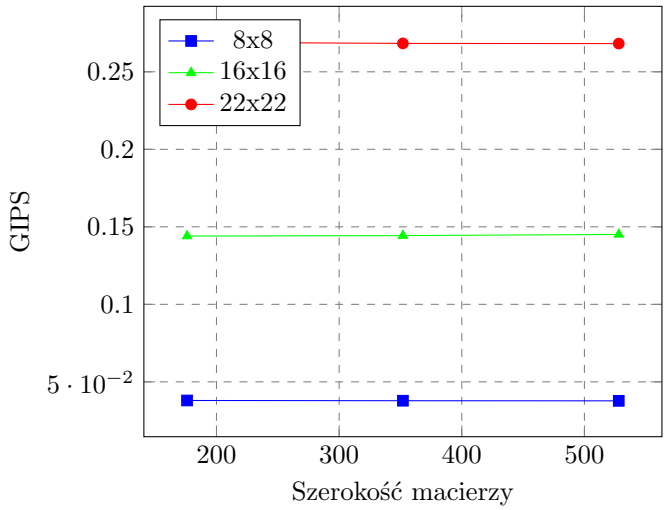


Rysunek 2: Zależność pomiędzy ilością operacji zmiennoprzecinkowychna sekundę a rozmiarem macierzy – wersja 1.

3. **Ilość instrukcji wykonanych na sekundę**

Rozmiar macierzy	Rozmiar bloku		
	8x8	16x16	22x22
176x176	0.03798	0.14404	0.26910
352x352	0.03782	0.14434	0.26832
528x528	0.03774	0.14509	0.26820

Tablica 5: Ilość instrukcji wykonana na sekundę (GIPS) – wersja 1.

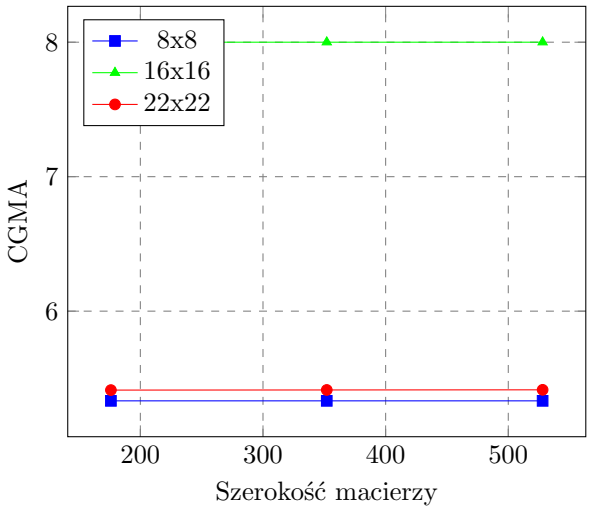


Rysunek 3: Zależność pomiędzy ilością instrukcji wykonanych na sekundę a rozmiarem macierzy – wersja 1.

4. **CGMA**

Rozmiar macierzy	Rozmiar bloku		
	8x8	16x16	22x22
176x176	5.333	8.000	5.413
352x352	5.333	8.000	5.414
528x528	5.333	8.000	5.415

Tablica 6: Stosunek ilości operacji zmiennoprzecinkowych do ilości operacji odczytu/zapisu z pamięci globalnej – wersja 1.



Rysunek 4: Zależność CGMA od rozmiaru macierzy – wersja 1.

3.2 Wersja 2

3.2.1 Opis rozwiązania

W drugiej wersji wykorzystywany jest grid wieloblokowy o rozmiarze

$$\frac{\text{rozmiar macierzy}}{\text{rozmiar bloku}}$$

Każdy wątek oblicza jeden element macierzy wynikowej. Pamięć współdzielona nie jest wykorzystywana.

```
__global__ void MatrixMulKernel_2(const float* Ad, const float* Bd,
    float* Cd, int WIDTH) {
  int Row = blockIdx.y * blockDim.y + threadIdx.y;
  int Col = blockIdx.x * blockDim.x + threadIdx.x;
  float C_local = 0.0f;

  for (int k = 0; k < WIDTH; ++k)
    C_local += Ad[Row*WIDTH + k] * Bd[k*WIDTH + Col];

  Cd[Row*WIDTH + Col] = C_local;
}
```

Listing 2: Mnożenie macierzy kwadratowych na GPU – wersja 2.

3.2.2 Teoretyczna zajętość SM

Kryterium		Teoretyczna wartość			Limit GPU
		8x8	16x16	22x22	
Zajętość SM	Aktywne bloki	8	4	2	8
	Aktywne warpy	16	32	32	32
	Aktywne wątki	512	1024	968	1024
	Zajętość	50%	100%	100%	100%
Warpy	Wątki/Blok	64	256	484	512
	Warpy/Blok	2	8	16	16
	Limit bloków	16	4	2	8
Rejestry	Rejestry/Wątek	10	10	10	128
	Rejestry/Blok	1024	2560	5120	16384
	Limit bloków	16	6	3	8
Pamięć współdzielona	Pamięć współdzielona/Blok	44	44	44	16384
	Limit bloków	32	32	32	8

Tablica 7: Teoretyczna zajętość SM – wersja 2.

Podobnie jak w 1. wersji, dla bloku 8x8 limitem okazuje się być maksymalna ilość bloków na SM, stąd zajętość $16/32 = 50\%$.
Dla macierzy 16x16 i 22x22 limitem są warpy. Przypada odpowiednio 8 i 16 warpów na blok, co daje limit 4 i 2 aktywnych bloków. Zajętość dla obu tych wielkości bloków ponownie wynosi 100%.

3.2.3 Wyniki pomiarów

1. **Czas trwania obliczeń**

Rozmiar macierzy	Rozmiar bloku		
	8x8	16x16	22x22
176x176	1.466	1.381	2.552
352x352	12.934	11.713	21.462
528x528	38.574	36.721	66.967

Tablica 8: Czas obliczeń [ms] – wersja 2.

2. **Ilość operacji zmiennoprzecinkowych na sekundę**

Rozmiar macierzy	Rozmiar bloku		
	8x8	16x16	22x22
176x176	7.440	7.897	4.273
352x352	6.744	7.447	4.064
528x528	7.632	8.017	4.396

Tablica 9: Ilosc operacji zmiennoprzecinkowych na sekundę (GFLOPS) – wersja 2.

3. **Ilość instrukcji wykonanych na sekundę**

Rozmiar macierzy	Rozmiar bloku		
	8x8	16x16	22x22
176x176	0.07851	0.08334	0.04509
352x352	0.07085	0.07776	0.04455
528x528	0.08095	0.08248	0.04777

Tablica 10: Ilość instrukcji wykonana na sekundę (GIPS) – wersja 2.

4. **CGMA**

Rozmiar macierzy	Rozmiar bloku		
	8x8	16x16	22x22
176x176	21.511	35.852	23.178
352x352	21.422	32.538	21.322
528x528	21.178	32.267	21.768

Tablica 11: Stosunek ilości operacji zmiennoprzecinkowych do ilości operacji odczytu/zapisu z pamięci globalnej – wersja 2.

3.3 Wersja 3

3.3.1 Opis rozwiązania

W trzecim podejściu wykorzystana została pamięć współdzielona. W kolejnych iteracjach pętli po blokach najpierw wczytywany jest blok do pamięci współdzielonej (każdy wątek wczytuje jedną komórkę), a następnie wykonywane są obliczenia na dostępnych danych. W tym podejściu niezbędne jest synchronizowanie się wątków dwukrotnie w każdej iteracji.

```
template <int BLOCK_SIZE> __global__ void
MatrixMulKernel_3(float *C, const float *A, const float *B, const int
    arraySize) {
  int bx = blockIdx.x;
```

```
  int by = blockIdx.y;

  int tx = threadIdx.x;
  int ty = threadIdx.y;

  int aBegin = arraySize * BLOCK_SIZE * by;
  int aEnd = aBegin + arraySize - 1;
  int aStep = BLOCK_SIZE;

  int bBegin = BLOCK_SIZE * bx;
  int bStep = BLOCK_SIZE * arraySize;

  float Csub = 0.0f;

  for (int a = aBegin, b = bBegin;
      a <= aEnd;
      a += aStep, b += bStep)
  {
    __shared__ float As[BLOCK_SIZE][BLOCK_SIZE];
    __shared__ float Bs[BLOCK_SIZE][BLOCK_SIZE];

    As[ty][tx] = A[a + arraySize * ty + tx];
    Bs[ty][tx] = B[b + arraySize * ty + tx];

    __syncthreads();

#pragma unroll
    for (int k = 0; k < BLOCK_SIZE; ++k) {
      Csub += As[ty][k] * Bs[k][tx];
    }

    __syncthreads();
  }

  int c = arraySize * BLOCK_SIZE * by + BLOCK_SIZE * bx;
  C[c + arraySize * ty + tx] = Csub;
}
```

Listing 3: Mnożenie macierzy kwadratowych na GPU – wersja 3.

3.3.2 Teoretyczna zajętość SM

Kryterium		Teoretyczna wartość			Limit GPU
		8x8	16x16	22x22	
Zajętość SM	Aktywne bloki	8	4	2	8
	Aktywne warpy	16	32	32	32
	Aktywne wątki	512	1024	968	1024
	Zajętość	50%	100%	100%	100%
Warpy	Wątki/Blok	64	256	484	512
	Warpy/Blok	2	8	16	16
	Limit bloków	16	4	2	8
Rejestry	Rejestry/Wątek	12	12	13	128
	Rejestry/Blok	1024	3072	6656	16384
	Limit bloków	16	5	2	8
Pamięć współdzielona	Pamięć współdzielona/Blok	556	2092	3916	16384
	Limit bloków	16	6	4	8

Tablica 12: Teoretyczna zajętość SM – wersja 3.

Podobnie jak dla 1. i 2. wersji, dla bloku 8x8 limitem okazuje się być maksymalna ilość bloków na SM, stąd zajętość $16/32 = 50\%$.
Dla macierzy 16x16 limitem są warpy. Przypada odpowiednio 8 warpów na blok, co daje limit 4 i 2 aktywnych bloków. Zajętość dla tej wielkości bloku wynosi 100%.
Dla macierzy 22x22 limitem są zarówno warpy jak i rejestry. Przypada 16 warpów na blok, co daje limit 2 aktywnych bloków. 6656 rejestrów na blok również daje limit 2 aktywnych bloków. Zajętość dla tej wielkości bloku wynosi 100%.

3.3.3 Wyniki pomiarów

1. **Czas trwania obliczeń**

Rozmiar macierzy	Rozmiar bloku		
	8x8	16x16	22x22
128x128	0.140	0.093	n/d
176x176	0.316	0.181	0.374
256x256	0.930	0.519	n/d
352x352	2.351	1.297	2.343
384x384	3.034	1.642	n/d
512x512	7.001	3.764	n/d
528x528	7.666	4.075	7.389
640x640	13.820	7.301	n/d

Tablica 13: Czas obliczeń [ms] – wersja 3.

2. **Ilość operacji zmiennoprzecinkowych na sekundę**

Rozmiar macierzy	Rozmiar bloku		
	8x8	16x16	22x22
128x128	29.857	45.291	n/d
176x176	34.491	60.169	29.138
256x256	36.086	64.603	n/d
352x352	37.106	67.273	37.231
384x384	37.322	68.970	n/d
512x512	38.341	71.314	n/d
528x528	38.402	72.246	39.844
640x640	37.938	71.812	n/d

Tablica 14: Ilosc operacji zmiennoprzecinkowych na sekundę (GFLOPS) – wersja 3.

3. **Ilość instrukcji wykonanych na sekundę**

Rozmiar macierzy	Rozmiar bloku		
	8x8	16x16	22x22
128x128	0.1639	0.1562	n/d
176x176	0.16910	0.23896	0.12664
256x256	0.1754	0.2622	n/d
352x352	0.17807	0.28408	0.15949
384x384	0.1767	0.2926	n/d
512x512	0.1802	0.2948	n/d
528x528	0.18043	0.30390	0.17317
640x640	0.1775	0.2989	n/d

Tablica 15: Ilość instrukcji wykonana na sekundę (GIPS) – wersja 3.

4. **CGMA**

Rozmiar macierzy	Rozmiar bloku		
	8x8	16x16	22x22
128x128	120.471	546.133	n/d
176x176	129.067	516.267	354.253
256x256	128.502	520.127	n/d
352x352	128.000	507.803	382.302
384x384	128.000	515.580	n/d
512x512	127.626	514.008	n/d
528x528	127.765	510.593	380.668
640x640	127.760	510.723	n/d

Tablica 16: Stosunek ilości operacji zmiennoprzecinkowych do ilości operacji odczytu/zapisu z pamięci globalnej – wersja 3.

3.4 Wersja 4

3.4.1 Opis rozwiązania

Jest to rozszerzona wersja 3 o równoległe z obliczeniami pobranie kolejnych danych (na poziomie bloku). Ma to spowodować złagodzenie kosztów synchronizacji.

(a) **Pobranie do rejestru**

```
template <int BLOCK_SIZE> __global__ void
MatrixMulKernel_4a(float *C, const float *A, const float *B, const
    int arraySize) {
  int bx = blockIdx.x;
  int by = blockIdx.y;

  int tx = threadIdx.x;
  int ty = threadIdx.y;

  int aBegin = arraySize * BLOCK_SIZE * by;
  int aEnd = aBegin + arraySize - 1;
  int aStep = BLOCK_SIZE;

  int bBegin = BLOCK_SIZE * bx;
  int bStep = BLOCK_SIZE * arraySize;

  float Csub = 0.0f;

  float fetchA, fetchB;
  fetchA = A[aBegin + arraySize * ty + tx];
```

```
  fetchB = B[bBegin + arraySize * ty + tx];

  for (int a = aBegin, b = bBegin;
      a <= aEnd;
      a += aStep, b += bStep)
  {
    __shared__ float As[BLOCK_SIZE][BLOCK_SIZE];
    __shared__ float Bs[BLOCK_SIZE][BLOCK_SIZE];

    As[ty][tx] = fetchA;
    Bs[ty][tx] = fetchB;

    __syncthreads();

    fetchA = A[a + aStep + arraySize * ty + tx];
    fetchB = B[b + bStep + arraySize * ty + tx];

#pragma unroll
    for (int k = 0; k < BLOCK_SIZE; ++k) {
      Csub += As[ty][k] * Bs[k][tx];
    }

    __syncthreads();
  }

  int c = arraySize * BLOCK_SIZE * by + BLOCK_SIZE * bx;
  C[c + arraySize * ty + tx] = Csub;
}
```

Listing 4: Mnożenie macierzy kwadratowych na GPU – wersja 4 z pobraniem do rejestru.

(b) **Pobranie do pamięci współdzielonej**

```
template <int BLOCK_SIZE> __global__ void
MatrixMulKernel_4b(float *C, const float *A, const float *B, const
    int arraySize) {
  int bx = blockIdx.x;
  int by = blockIdx.y;

  int tx = threadIdx.x;
  int ty = threadIdx.y;

  int aBegin = arraySize * BLOCK_SIZE * by;
  int aEnd = aBegin + arraySize - 1;
  int aStep = BLOCK_SIZE;

  int bBegin = BLOCK_SIZE * bx;
  int bStep = BLOCK_SIZE * arraySize;
```

```
  float Csub = 0.0f;

  __shared__ float fetchA[BLOCK_SIZE][BLOCK_SIZE];
  __shared__ float fetchB[BLOCK_SIZE][BLOCK_SIZE];

  fetchA[ty][tx] = A[aBegin + arraySize * ty + tx];
  fetchB[ty][tx] = B[bBegin + arraySize * ty + tx];

  for (int a = aBegin, b = bBegin;
      a <= aEnd;
      a += aStep, b += bStep)
  {
    __shared__ float As[BLOCK_SIZE][BLOCK_SIZE];
    __shared__ float Bs[BLOCK_SIZE][BLOCK_SIZE];

    As[ty][tx] = fetchA[ty][tx];
    Bs[ty][tx] = fetchB[ty][tx];

    __syncthreads();

    fetchA[ty][tx] = A[a + aStep + arraySize * ty + tx];
    fetchB[ty][tx] = B[b + bStep + arraySize * ty + tx];

#pragma unroll
    for (int k = 0; k < BLOCK_SIZE; ++k) {
      Csub += As[ty][k] * Bs[k][tx];
    }

    __syncthreads();
  }

  int c = arraySize * BLOCK_SIZE * by + BLOCK_SIZE * bx;
  C[c + arraySize * ty + tx] = Csub;
}
```

Listing 5: Mnożenie macierzy kwadratowych na GPU – wersja 4 z pobraniem do pamięci współdzielonej.

3.4.2 Teoretyczna zajętość SM

(a) **Pobranie do rejestru**

Kryterium		Teoretyczna wartość		Limit GPU
		8x8	16x16	
Zajętość SM	Aktywne bloki	8	4	8
	Aktywne warpy	16	32	32
	Aktywne wątki	512	1024	1024
	Zajętość	50%	100%	100%
Warpy	Wątki/Blok	64	256	512
	Warpy/Blok	2	8	16
	Limit bloków	16	4	8
Rejestry	Rejestry/Wątek	13	13	128
	Rejestry/Blok	1024	3584	16384
	Limit bloków	16	4	8
Pamięć współdzielona	Pamięć współdzielona/Blok	556	2092	16384
	Limit bloków	16	6	8

Tablica 17: Teoretyczna zajętość SM – wersja 4. z pobraniem do rejestru.

Podobnie jak dla poprzednich wersji, dla bloku 8x8 limitem okazuje się być maksymalna ilość bloków na SM, stąd zajętość $16/32 = 50\%$.
Dla macierzy 16x16 limitem są warpy i rejestry. Przypada odpowiednio 8 warpów na blok, co daje limit 4 aktywnych bloków. 3584 rejestrów na blok również daje limit 4 aktywnych bloków. Zajętość dla bloku 16x16 wynosi 100%.

(b) **Pobranie do pamięci współdzielonej**

Kryterium		Teoretyczna wartość		Limit GPU
		8x8	16x16	
Zajętość SM	Aktywne bloki	8	3	8
	Aktywne warpy	16	24	32
	Aktywne wątki	512	768	1024
	Zajętość	50%	75%	100%
Warpy	Wątki/Blok	64	256	512
	Warpy/Blok	2	8	16
	Limit bloków	16	4	8
Rejestry	Rejestry/Wątek	14	14	128
	Rejestry/Blok	1024	3584	16384
	Limit bloków	16	4	8
Pamięć współdzielona	Pamięć współdzielona/Blok	1068	4140	16384
	Limit bloków	16	3	8

Tablica 18: Teoretyczna zajętość SM – wersja 4. z pobraniem do pamięci współdzielonej.

Podobnie jak dla poprzednich wersji, dla bloku 8x8 limitem okazuje się być maksymalna ilość bloków na SM, stąd zajętość $16/32 = 50\%$.
Dla macierzy 16x16 limitem jest pamięć współdzielona. 4140 bajtów na blok daje limit 3 aktywnych bloków. W porównaniu z poprzednimi wersjami zajętość SM dla bloku 16x16 spada i wynosi 75%.

3.4.3 Wyniki pomiarów

(a) **Pobranie do rejestru**

1. **Czas trwania obliczeń**
2. **Ilość operacji zmiennoprzecinkowych na sekundę**
3. **Ilość instrukcji na sekundę**
4. **CGMA**

(b) **Pobranie do pamięci współdzielonej**

1. **Czas trwania obliczeń**

2. **Ilość operacji zmiennoprzecinkowych na sekundę**

3. **Ilość instrukcji na sekundę**

4. **CGMA**

3.5 Wersja 5

3.5.1 Opis rozwiązania

Ostatnia wersja rozszerza wersję 4 – każdy wątek wykonuje większą pracę. Zostało to zrealizowane przez zwiększenie pobieranych i obliczanych danych z jednej do czterech.

```
template <int BLOCK_SIZE> __global__ void
MatrixMulKernel_5(float *C, const float *A, const float *B, const int
    arraySize) {
  int bx = blockIdx.x;
  int by = blockIdx.y;

  int tx = threadIdx.x;
  int ty = threadIdx.y;

  int aBegin = 2 * arraySize * BLOCK_SIZE * by;
  int aEnd = aBegin + arraySize - 1;
  int aStep = BLOCK_SIZE;

  int bBegin = 2 * BLOCK_SIZE * bx;
  int bStep = BLOCK_SIZE * arraySize;

  float Csub00=0.0f, Csub01=0.0f, Csub10=0.0f, Csub11=0.0f;
  float fetchA0, fetchA1;
  float fetchB0, fetchB1;
  fetchA0 = A[aBegin + arraySize * 2*ty + tx];
  fetchA1 = A[aBegin + arraySize * (2*ty+1) + tx];
  fetchB0 = B[bBegin + arraySize * ty + 2*tx];
  fetchB1 = B[bBegin + arraySize * ty + 2*tx+1];

  for (int a = aBegin, b = bBegin;
      a <= aEnd;
      a += aStep, b += bStep)
  {
    __shared__ float As[2 * BLOCK_SIZE][BLOCK_SIZE];
    __shared__ float Bs[BLOCK_SIZE][2 * BLOCK_SIZE];

    As[2*ty+0][tx] = fetchA0;
    As[2*ty+1][tx] = fetchA1;
    Bs[ty][2*tx+0] = fetchB0;
```

```
    Bs[ty][2*tx+1] = fetchB1;

    __syncthreads();

    fetchA0 = A[a + aStep + arraySize * 2*ty + tx];
    fetchA1 = A[a + aStep + arraySize * (2*ty+1) + tx];
    fetchB0 = B[b + bStep + arraySize * ty + 2*tx];
    fetchB1 = B[b + bStep + arraySize * ty + 2*tx+1];

#pragma unroll
    for (int k = 0; k < BLOCK_SIZE; ++k) {
      Csub00 += As[2*ty][k] * Bs[k][2*tx];
    }
#pragma unroll
    for (int k = 0; k < BLOCK_SIZE; ++k) {
      Csub01 += As[2*ty][k] * Bs[k][2*tx+1];
    }
#pragma unroll
    for (int k = 0; k < BLOCK_SIZE; ++k) {
      Csub10 += As[2*ty+1][k] * Bs[k][2*tx];
    }
#pragma unroll
    for (int k = 0; k < BLOCK_SIZE; ++k) {
      Csub11 += As[2*ty+1][k] * Bs[k][2*tx+1];
    }

    __syncthreads();
  }

  int c = 2 * arraySize * BLOCK_SIZE * by + 2 * BLOCK_SIZE * bx;
  C[c + arraySize * (2*ty) + 2*tx] = Csub00;
  C[c + arraySize * (2*ty) + 2*tx+1] = Csub01;
  C[c + arraySize * (2*ty+1) + 2*tx] = Csub10;
  C[c + arraySize * (2*ty+1) + 2*tx+1] = Csub11;

}
```

Listing 6: Mnożenie macierzy kwadratowych na GPU – wersja 5.

3.5.2 Teoretyczna zajętość SM

Kryterium		Teoretyczna wartość		Limit GPU
		8x8	16x16	
Zajętość SM	Aktywne bloki	8	1	8
	Aktywne warpy	16	8	32
	Aktywne wątki	512	256	1024
	Zajętość	50%	25%	100%
Warpy	Wątki/Blok	64	256	512
	Warpy/Blok	2	8	16
	Limit bloków	16	4	8
Rejestry	Rejestry/Wątek	23	34	128
	Rejestry/Blok	1536	8704	16384
	Limit bloków	10	1	8
Pamięć współdzielona	Pamięć współdzielona/Blok	1068	4140	16384
	Limit bloków	10	3	8

Tablica 19: Teoretyczna zajętość SM – wersja 5.

Podobnie jak dla poprzednich wersji, dla bloku 8x8 limitem okazuje się być maksymalna ilość bloków na SM, stąd zajętość $16/32 = 50\%$.
Dla macierzy 16x16 limitem są rejestry. 8704 rejestrów na blok daje limit 1 aktywnych bloków. Zajętość dla bloku 16x16 wynosi zaledwie 25%.

3.5.3 Wyniki pomiarów

1. **Czas trwania obliczeń**

Rozmiar macierzy	Rozmiar bloku	
	8x8	16x16
128x128	0.118	0.139
256x256	0.877	0.763
384x384	2.622	2.287
512x512	6.248	5.471
640x640	12.047	10.632

Tablica 20: Czas obliczeń [ms] – wersja 5.

2. **Ilość operacji zmiennoprzecinkowych na sekundę**

Rozmiar macierzy	Rozmiar bloku	
	8x8	16x16
128x128	35.396	30.229
256x256	38.248	43.995
384x384	43.189	49.512
512x512	42.966	49.063
640x640	43.520	49.314

Tablica 21: Ilosc operacji zmiennoprzecinkowych na sekundę (GFLOPS) – wersja 5.

3. **Ilość instrukcji na sekundę**

Rozmiar macierzy	Rozmiar bloku	
	8x8	16x16
128x128	0.1481	0.1856
256x256	0.1423	0.1641
384x384	0.1543	0.1951
512x512	0.1574	0.1894
640x640	0.1538	0.1909

Tablica 22: Ilość instrukcji wykonana na sekundę (GIPS) – wersja 5.

4. **CGMA**

Rozmiar macierzy	Rozmiar bloku	
	8x8	16x16
128x128	240.941	1129.931
256x256	256.250	1236.528
384x384	250.776	1276.675
512x512	253.050	1297.743
640x640	255.393	1310.720

Tablica 23: Stosunek ilości operacji zmiennoprzecinkowych do ilości operacji odczytu/zapisu z pamięci globalnej – wersja 5.

4 Podsumowanie

5 Załączniki

1. Kody źródłowe

Spis rysunków

Spis tablic

Listingi