Politechnika Poznańska

Wydział Informatyki

Przetwarzanie Równoległe

Programowanie CUDA na NVIDIA GPU

Autorzy:
Adam Szczepański
Mateusz Czajka

Prowadzący:
dr Rafał Walkowiak

26 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.

Dla każdej wersji podajemy teoretyczną zajętość SM wynikającą z rozmiaru bloku, wykorzystanej liczby rejestrów, rozmiaru pamięci współdzielonej, a także z ograniczeń GPU. 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.

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	8	4	2	8
	Aktywne warpy	16	32	32	32
	Aktywne wątki	512	1024	1024	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	16	16	16	128
	Rejestry/Blok	1024	4096	8192	16384
	Limit bloków	16	4	2	8
Pamięć współdzielona	Pamięć współdzielona/Blok	44	44	44	16384
	Limit bloków	32	32	32	8

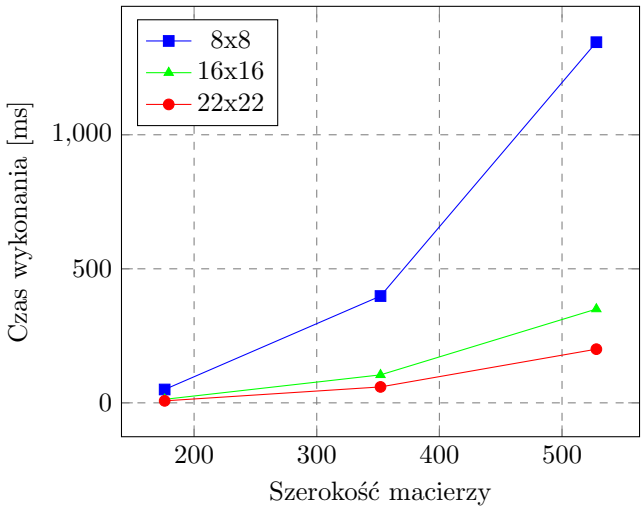
Tablica 2: Teoretyczna zajętość SM.

W przypadku rozmiaru bloku 8x8 limitem okazuje się być maksymalna ilość bloków na SM – 8 bloków po 64 wątki oznacza wykorzystanie 8×2 warpów, czyli zajętość $16/32 = 50\%$.
Dla macierzy 16x16 i 22x22 limitem są warpy i rejestry. Przypada odpowiednio 8 i 16 warpów na blok, co daje limit 4 i 2 bloków. Takie same ograniczenia wynikają z ilości rejestrów. Mimo tego zajętość dla obu tych macierzy wynosi 100%.

3.1.3 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.

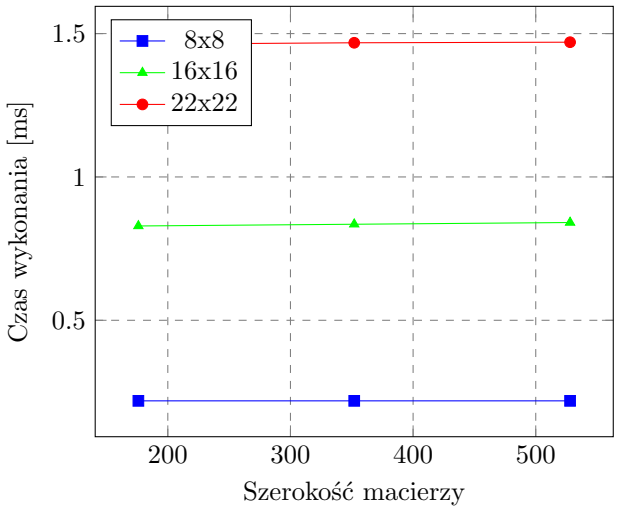


Rysunek 1: Zależność pomiędzy czasem obliczeń a rozmiarem macierzy – wersja 1.

3.1.4 Ilosc 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: Ilosc operacji zmiennoprzecinkowych na sekundę (GFLOPS) – wersja 1.

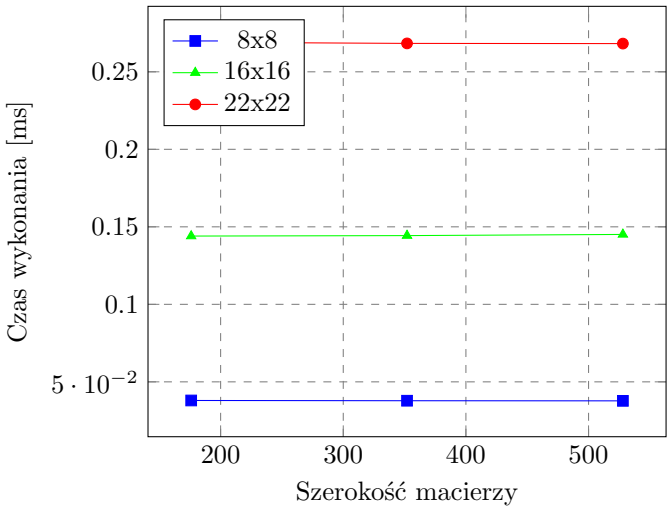


Rysunek 2: Zależność pomiędzy ilością operacji zmiennoprzecinkowychna sekundę a rozmiarem macierzy – wersja 1.

3.1.5 Ilość instrukcji wykonana 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.

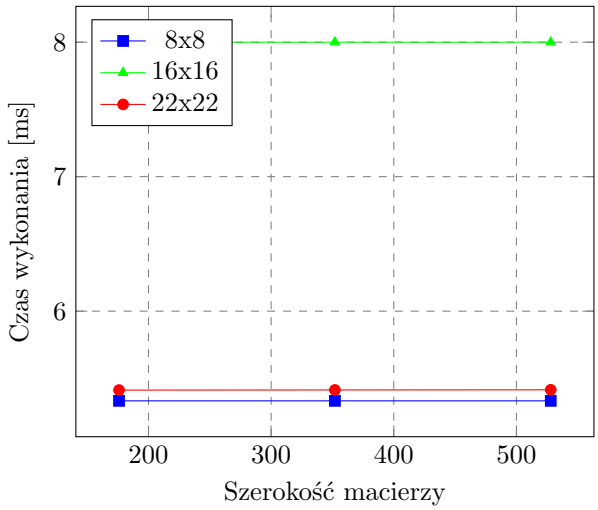


Rysunek 3: Zależność pomiędzy ilością instrukcji wykonanych na sekundę a rozmiarem macierzy – wersja 1.

3.1.6 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

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(float* Ad, 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.

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 7: Czas obliczeń [ms] – wersja 2.

3.2.1 Ilosc 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 8: Ilosc operacji zmiennoprzecinkowych na sekundę (GFLOPS) – wersja 1.

3.2.2 Ilość instrukcji wykonana 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 9: Ilość instrukcji wykonana na sekundę (GIPS) – wersja 1.

3.2.3 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 10: Stosunek ilości operacji zmiennoprzecinkowych do ilości operacji odczytu/zapisu z pamięci globalnej – wersja 1.

3.3 Wersja 3

W trzecim podejściu wykorzystana została pamięć współdzielona. W kolejnych iteracjach pętli po blokach najpierw wczytywany jest blok do pamięci współ-

dzielonej (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;

    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.

Rozmiar macierzy	Rozmiar bloku		
	8x8	16x16	22x22
176x176	0.316	0.181	0.374
352x352	2.351	1.297	2.343
528x528	7.666	4.075	7.389

Tablica 11: Czas obliczeń [ms] – wersja 3.

3.3.1 Ilosc operacji zmiennoprzecinkowych na sekundę

Rozmiar macierzy	Rozmiar bloku		
	8x8	16x16	22x22
176x176	34.491	60.169	29.138
352x352	37.106	67.273	37.231
528x528	38.402	72.246	39.844

Tablica 12: Ilosc operacji zmiennoprzecinkowych na sekundę (GFLOPS) – wersja 1.

3.3.2 Ilość instrukcji wykonana na sekundę

Rozmiar macierzy	Rozmiar bloku		
	8x8	16x16	22x22
176x176	0.16910	0.23896	0.12664
352x352	0.17807	0.28408	0.15949
528x528	0.18043	0.30390	0.17317

Tablica 13: Ilość instrukcji wykonana na sekundę (GIPS) – wersja 1.

3.3.3 CGMA

Rozmiar macierzy	Rozmiar bloku		
	8x8	16x16	22x22
176x176	129.067	516.267	354.253
352x352	128.000	507.803	382.302
528x528	127.765	510.593	380.668

Tablica 14: Stosunek ilości operacji zmiennoprzecinkowych do ilości operacji odczytu/zapisu z pamięci globalnej – wersja 1.

Rozmiar macierzy	Rozmiar bloku	
	8x8	16x16
128x128	0.140	0.093
256x256	0.930	0.519
384x384	3.034	1.642
512x512	7.001	3.764
640x640	13.820	7.301

Tablica 15: Czas obliczeń [ms] – wersja 3.

3.3.4 Ilosc operacji zmiennoprzecinkowych na sekundę

Rozmiar macierzy	Rozmiar bloku	
	8x8	16x16
128x128	29.857	45.291
256x256	36.086	64.603
384x384	37.322	68.970
512x512	38.341	71.314
640x640	37.938	71.812

Tablica 16: Ilosc operacji zmiennoprzecinkowych na sekundę (GFLOPS) – wersja 1.

3.3.5 Ilość instrukcji wykonana na sekundę

Rozmiar macierzy	Rozmiar bloku	
	8x8	16x16
128x128	0.1639	0.1562
256x256	0.1754	0.2622
384x384	0.1767	0.2926
512x512	0.1802	0.2948
640x640	0.1775	0.2989

Tablica 17: Ilość instrukcji wykonana na sekundę (GIPS) – wersja 1.

3.3.6 CGMA

Rozmiar macierzy	Rozmiar bloku	
	8x8	16x16
128x128	120.471	546.133
256x256	128.502	520.127
384x384	128.000	515.580
512x512	127.626	514.008
640x640	127.760	510.723

Tablica 18: Stosunek ilości operacji zmiennoprzecinkowych do ilości operacji odczytu/zapisu z pamięci globalnej – wersja 1.

3.4 Wersja 4

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

```
template <int BLOCK_SIZE> __global__ void
MatrixMulKernel_4(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 = A[aBegin + arraySize * ty + tx];
   float 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();

    if (a < aEnd) {
      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.

Rozmiar macierzy	Rozmiar bloku
	16x16
64x64	0.02
128x128	0.10
256x256	0.60
384x384	1.89
512x512	4.44

Tablica 19: Czas obliczeń [ms] – wersja 4.

3.5 Wersja 5

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 = 2 * BLOCK_SIZE;

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

float Csub00=0.0f, Csub01=0.0f, Csub10=0.0f, Csub11=0.0f;
float fetchA00, fetchA01, fetchA10, fetchA11;
float fetchB00, fetchB01, fetchB10, fetchB11;

fetchA00 = A[aBegin + arraySize * (2*ty+0) + 2*tx+0];
fetchA01 = A[aBegin + arraySize * (2*ty+0) + 2*tx+1];
fetchA10 = A[aBegin + arraySize * (2*ty+1) + 2*tx+0];
fetchA11 = A[aBegin + arraySize * (2*ty+1) + 2*tx+1];
fetchB00 = B[bBegin + arraySize * (2*ty+0) + 2*tx+0];
fetchB01 = B[bBegin + arraySize * (2*ty+0) + 2*tx+1];
fetchB10 = B[bBegin + arraySize * (2*ty+1) + 2*tx+0];
fetchB11 = B[bBegin + arraySize * (2*ty+1) + 2*tx+1];

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

  As[2*ty+0][2*tx+0] = fetchA00;
  As[2*ty+0][2*tx+1] = fetchA01;
  As[2*ty+1][2*tx+0] = fetchA10;
  As[2*ty+1][2*tx+1] = fetchA11;
  Bs[2*ty+0][2*tx+0] = fetchB00;
  Bs[2*ty+0][2*tx+1] = fetchB01;
  Bs[2*ty+1][2*tx+0] = fetchB10;
  Bs[2*ty+1][2*tx+1] = fetchB11;

      __syncthreads();

  fetchA00 = A[a + aStep + arraySize * (2*ty+0) + 2*tx+0];
  fetchA01 = A[a + aStep + arraySize * (2*ty+0) + 2*tx+1];
  fetchA10 = A[a + aStep + arraySize * (2*ty+1) + 2*tx+0];
  fetchA11 = A[a + aStep + arraySize * (2*ty+1) + 2*tx+1];
  fetchB00 = B[b + bStep + arraySize * (2*ty+0) + 2*tx+0];
  fetchB01 = B[b + bStep + arraySize * (2*ty+0) + 2*tx+1];
  fetchB10 = B[b + bStep + arraySize * (2*ty+1) + 2*tx+0];
  fetchB11 = B[b + bStep + arraySize * (2*ty+1) + 2*tx+1];

  for (int k = 0; k < (2*BLOCK_SIZE); ++k) {
    Csub00 += As[2*ty+0][k] * Bs[k][2*tx+0];
  }
```

```
    for (int k = 0; k < (2*BLOCK_SIZE); ++k) {
      Csub01 += As[2*ty+0][k] * Bs[k][2*tx+1];
    }
    for (int k = 0; k < (2*BLOCK_SIZE); ++k) {
      Csub10 += As[2*ty+1][k] * Bs[k][2*tx+0];
    }
    for (int k = 0; k < (2*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+0) + 2*tx+0] = Csub00;
  C[c + arraySize * (2*ty+0) + 2*tx+1] = Csub01;
  C[c + arraySize * (2*ty+1) + 2*tx+0] = Csub10;
  C[c + arraySize * (2*ty+1) + 2*tx+1] = Csub11;

}
```

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

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 3.

3.5.1 Ilosc 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 1.

3.5.2 Ilość instrukcji wykonana 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 1.

3.5.3 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 1.

4 Podsumowanie

5 Załączniki

1. Kody źródłowe

Spis rysunków

Spis tablic

Listingi