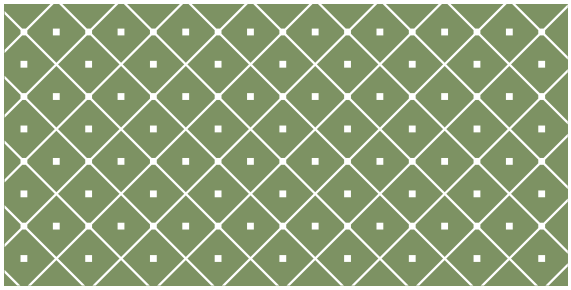
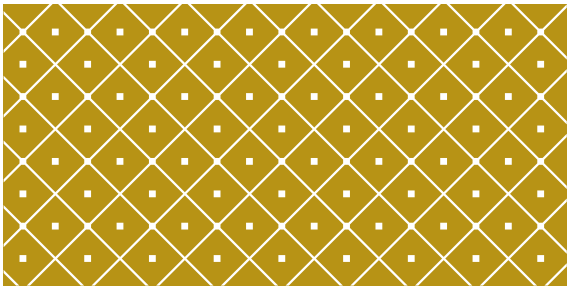


PROGRAMOWANIE WEKTOROWE I RÓWNOLEGŁE

Wykład dla kierunku: Matematyka stosowana i technologie informatyczne



PROGRAMOWANIE GPU – NOWE TYPY DANYCH, STRUKTURY I INSTRUKCJE.

WBUDOWANE TYPY WEKTOROWE

char, short, int, long, longlong, float, double

Są to typy wektorów pochodzące z podstawowych typów liczb całkowitych i zmiennoprzecinkowych. Są to struktury, a komponenty 1., 2., 3. i 4. są dostępne odpowiednio przez pola X, Y, Z i W. Wszystkie są wyposażone w funkcję konstruktora formularza Make_ <Type Nazwa>; Na przykład,

int2 Make_int2 (int x, int y); np.: b= Make_int2(3,7)

który tworzy wektor typu INT2 z wartością (x, y).

Przykład odwołania a=b.y

WBUDOWANE TYPY WEKTOROWE I ICH WYRÓWNANIE

char1, uchar1	1	int1, uint1	4	longlong1, ulonglong1	8	float1	4
char2, uchar2	2	int2, uint2	8			float2	8
char3, uchar3	1	int3, uint3	4	longlong2, ulonglong2	16	float3	4
char4, uchar4	4	int4, uint4	16			float4	16
short1, ushort1	2	long1, ulong1	4	longlong3, ulonglong3	8	double1	8
short2, ushort2	4	long2, ulong2	8			double2	16
short3, ushort3	2	long3, ulong3	4	longlong4, ulonglong4	16	double3	8
short4, ushort4	8	long4, ulong4	16			double4	16

dim3

dim3 jest typem wektorowym opartym na uint3, jest używany do określania wymiarów. Podczas definiowania zmiennej typu dim3 każdy komponent, który nie został określony, jest inicjowany na 1.

gridDim, blockIdx, blockDim, threadIdx i warpSize

gridDim jest typu dim3 i zawiera wymiary siatki.

blockIdx jest typu uint3 i zawiera indeks bloku w siatce.

blockDim jest typu dim3 i zawiera wymiary bloku.

threadIdx jest typu uint3 i zawiera indeks wątku w bloku.

warpSize jest typu int i zawiera rozmiar osnowy podany w wątkach

Funkcje matematyczne

__device__ float max (const float a, const float b)
__device__ double max (const double a, const float b)
__device__ double max (const float a, const double b)
__device__ double max (const double a, const double b)
Oblicz maksymalną wartość argumentów wejściowych.
__device__ float min (const float a, const float b)
__device__ double min (const double a, const float b)
__device__ double min (const float a, const double b)
__device__ double min (const double a, const double b)
Oblicz minimalną wartość argumentów wejściowych.

Funkcje matematyczne

__device__ float fmaxf (float x, float y)
__device__ double fmax (double x, double y)
Określ maksymalną wartość liczbową argumentów.
__device__ float fminf (float x, float y)
__device__ double fmin (double x, double y)
Określ minimalną wartość liczbową argumentów.

Funkcje matematyczne

__device__ float fdividef (float x, float y)
Podziel dwie wartości zmiennoprzecinkowe.
__device__ float fmodf (float x, float y)
__device__ double fmod (double x, double y)
Oblicz resztę z x / y.
__device__ float remainderf (float x, float y)
__device__ double remainder (double x, double y)
Oblicz resztę zmiennoprzecinkową z dzielenia.
__device__ float remquof (float x, float y, int* quo)
__device__ double remquo (double x, double y, int* quo)
Oblicz resztę zmiennoprzecinkową i część całkowitą.
__device__ float frexpf (float x, int* nptr)
__device__ double frexp (double x, int* nptr)
Wyodrębnij mantysę i wykładnik o wartości zmiennoprzecinkowej.

Funkcje matematyczne

__device__ float fabsf (float x)
__device__ double fabs (double x)
Oblicz wartość bezwzględną jego argumentu.

Funkcje matematyczne

__device__ float fmaf (float x, float y, float z)
__device__ double fma (double x, double y, double z)
Oblicz x*y+z jako pojedynczą operację.

Funkcje matematyczne

__device__ __RETURN_TYPE signit (float a)
__device__ __RETURN_TYPE signbit (double a)
Zwróć bit znaku argumentu wejściowego.
__device__ float copysignf (float x, float y)
__device__ double copysign (double x, double y)
Utwórz wartość z daną wielkością, kopiowanie znaku drugiej wartości.

Funkcje matematyczne

__device__ float powf (float x, float y)
__device__ double pow (double x, double y)
Oblicz wartość pierwszego argumentu do potęgi drugiego argumentu.
__device__ float fdimf (float x, float y)
__device__ double fdim (double x, double y)
Oblicz pozytywną różnicę między x i y. if x<=y zwróć 0.

Funkcje matematyczne

__device__ float sqrtf (float x)
__device__ double sqrt (double x)
Oblicz pierwiastek kwadratowy argumentu wejściowego.
__device__ float cbrtf (float x)
__device__ double cbrt (double x)
Oblicz pierwiastek 3 stopnia argumentu wejściowego.

Funkcje matematyczne

__device__ float rsqrtf (float x)
__device__ double rsqrt (double x)
Oblicz odwrotność pierwiastka kwadratowego argumentu wejściowego.
__device__ float rcbrtf (float x)
__device__ double rcbrt (double x)
Oblicz odwrotność pierwiastka sześciennego.

Funkcje matematyczne

__device__ float hypotf (float x, float y)
__device__ double hypot (double x, double y)
Oblicz pierwiastek kwadratowy z sumy kwadratów dwóch argumentów.
__device__ float norm3df (float a, float b, float c)
__device__ double norm3d (double a, double b, double c)
Oblicz pierwiastek kwadratowy z sumy kwadratów trzech współrzędnych.
__device__ float norm4df (float A, float B, float C, float D)
__device__ double norm4d (double a, double b, double c, double d)
Oblicz pierwiastek kwadratowy z sumy kwadratów czterech współrzędnych.
__device__ float normf (int dim, const float* p)
__device__ double norm (int dim, const double* p)
Oblicz pierwiastek kwadratowy suma kwadratów dowolnej liczby współrzędnych.

Funkcje matematyczne

__device__ float rhypotf (float x, float y)
__device__ double rhypot (double x, double y)
Oblicz odwrotność pierwiastka kwadratowego sumy kwadratów dwóch argumentów.
__device__ float rnorm3df (float a, float b, float c)
__device__ double rnorm3d (double a, double b, double c)
Oblicz odwrotność pierwiastka kwadratowego sumy kwadratów trzech współrzędnych.
__device__ float rnorm4df (float a, float b, float c, float d)
__device__ double rnorm4d (double a, double b, double c, double d)
Oblicz odwrotność pierwiastka kwadratowego sumy kwadratów czterech współrzędnych.
__device__ float rnormf (int dim, const float* p)
__device__ double rnorm (int dim, const double* p)
Oblicz odwrotność pierwiastka kwadratowego sumy kwadratów dowolnej liczby współrzędnych.

Funkcje matematyczne

__device__ float sinf (float x)
__device__ double sin (double x)
Oblicz sinus argumentu wejściowego.
__device__ float cosf (float x)
__device__ double cos (double x)
Oblicz cosinus argumentu wejściowego.
__device__ float tanf (float x)
__device__ double tan (double x)
Oblicz tangens argumentu wejściowego.

Funkcje matematyczne

__device__ float sinpif (float x)
__device__ double sinpi (double x)
Obliczyć sinus argumentu wejściowego razy pi.
__device__ float cospif (float x)
__device__ double cospi (double x)
Oblicz cosinus*pi argumentu wejściowego.

Funkcje matematyczne

__device__ void sincosf (float x, float* sptr, float* cptr)
__device__ void sincos (double x, double* sptr, double* cptr)
Oblicz sinus i cosinus pierwszego argumentu wejściowego.
__device__ void sincospif (float x, float* sptr, float* cptr)
__device__ void sincospi (double x, double* sptr, double* cptr)
Oblicz sinus i cosinus pierwszego argumentu wejściowego razy pi.

Funkcje matematyczne

__device__ float asinf (float X)
__device__ double asin (double x)
Oblicz arcus sinus argumentu wejściowego.
__device__ float acosf (float x)
__device__ double acos (double x)
Oblicz arcus cosinus argumentu wejściowego.
__device__ float atanf (float x)
__device__ double atan (double x)
Oblicz arcus tangens argumentu wejściowego.
__device__ float atan2f (float y, float x)
__device__ double atan2 (double y, double x)
Oblicz arcus tangens stosunku pierwszego i drugiego argumentu wejściowego.

Funkcje matematyczne

__device__ float sinhf (float x)
__device__ double sinh (double x)
Oblicz sinus hiperboliczny argumentu wejściowego.
__device__ float coshf (float x)
__device__ double cosh (double x)
Oblicz cosinus hiperboliczny argumentu wejściowego.
__device__ float tanhf (float x)
__device__ double tanh (double x)
Oblicz tangens hiperboliczny argumentu wejściowego.

Funkcje matematyczne

__device__ float asinhf (float X)
__device__ double asinh (double x)
Oblicz arcus sinus hiperboliczny argumentu wejściowego.
__device__ float acoshf (float x)
__device__ double acosh (double x)
Oblicz nieujemny arcus cosinus hiperboliczny argumentu wejściowego.
__device__ float atanhf (float x)
__device__ double atanh (double x)
Oblicz arcus tangens hiperboliczny argumentu wejściowego.

Funkcje matematyczne

__device__ float expf (float x)
__device__ double exp (double x)
Oblicz f. wykładniczą o podstawie e argumentu wejściowego.
__device__ float exp10f (float x)
__device__ double exp10 (double x)
Oblicz f. wykładniczą o podstawie 10 argumentu wejściowego.
__device__ float exp2f (float x)
__device__ double exp2 (double x)
Oblicz f. wykładniczą o podstawie 2 argumentu wejściowego.
__device__ float expm1f (float x)
__device__ double expm1 (double x)
Oblicz f. wykładniczą o podstawie e argumentu wejściowego, minus 1.

Funkcje matematyczne

__device__ float ldexpf (float x, int exp)
__device__ double ldexp (double x, int exp)
Obliczyć wartość x*2^exp.
__device__ float scalblnf (float x, long int n)
__device__ double scalbln (double x, long int n)
Oblicz x*2^n.
__device__ float scalbnf (float x, int n)
__device__ double scalbn (double x, int n)
Oblicz x*2^n.

Funkcje matematyczne

__device__ float logf (float x)
__device__ double log (double x)
Oblicz naturalny logarytm argumentu wejściowego.
__device__ float log10f (float x)
__device__ double log10 (double x)
Oblicz logarytm o podstawie 10 argumentu wejściowego.
__device__ float log2f(float x)
__device__ double log2 (double x)
Oblicz logarytm podstawy 2 argumentu wejściowego.
__device__ float log1pf (float x)
__device__ double log1p (double x)
Obliczyć wartość ln(1+x).

Funkcje matematyczne

__device__ int ilogbf (float x)
__device__ int ilogb (double x)
Oblicz wykładnik całkowity argumentu.
__device__ float logbf (float x)
__device__ double logb (double x)
Oblicz reprezentację zmiennoprzecinkową wykładnika argumentu wejściowego.

Funkcje matematyczne

__device__ float ceilf (float x)
__device__ double ceil (double x)
Oblicz sufit argumentu wejściowego.
__device__ float floorf (float x)
__device__ double floor (double x)
Oblicz największą liczbę całkowitą mniejszą lub równą x.
__device__ float roundf (float x)
__device__ double round (double x)
Zaokrąglij do najbliższej wartości całkowitej.
__device__ float truncf (float x)
__device__ double trunc (double x)
Obcinaj argument wejściowy do części całkowitej.

Funkcje matematyczne

__device__ float nextafterf (float x, float y)
__device__ double nextafter (double x, double y)
Zwróć następną reprezentatywną wartość zmiennoprzecinkową pojedynczej precyzji po argumencie X w kierunku Y.
__device__ float modff (float x, float* iptr)
__device__ double modf (double x, double* iptr)
Rozbij argument wejściowy na części ułamkowe i całkowite.

Funkcje matematyczne

__device__ long long int llrintf (float x)
__device__ long long int llrint (double x)
Zaokrąglij wejście do najbliższej wartości całkowitej.
__device__ long long int llroundf (float x)
__device__ long long int llround (double x)
Zaokrąglij wejście do najbliższej wartości całkowitej.
__device__ long int lrintf (float x)
__device__ long int lrint (double x)
Zaokrąglij wejście do najbliższej wartości całkowitej.

Funkcje matematyczne

__device__ long int lroundf (float x)
__device__ long int lround (double x)
Zaokrąglij wejście do najbliższej wartości całkowitej.
__device__ float nearbyintf (float x)
__device__ double nearbyint (double x)
Zaokrąglij argument wejściowy do najbliższej liczby całkowitej.
__device__ float rintf (float x)
__device__ double rint (double x)
Zaokrąglij wejście do najbliższej wartości całkowitej.

Funkcje matematyczne

__device__ float normcdff (float x)
__device__ double normcdf (double x)
Oblicz standardową normalną funkcję rozkładu skumulowanego.
__device__ float normcdfinv f (float x)
__device__ double normcdfinv (double x)
Oblicz odwrotność standardowej normalnej funkcji rozkładu skumulowanego.

Funkcje matematyczne

__device__ __RETURN_TYPE isfinite (float a)
__device__ __RETURN_TYPE isfinite (double a)
Ustal, czy argument jest skończony.
__device__ __RETURN_TYPE isinf (float a)
__device__ __RETURN_TYPE isinf (double a)
Ustal, czy argument jest nieskończony.
__device__ __RETURN_TYPE isnan (float a)
__device__ __RETURN_TYPE isnan (double a)
Ustal, czy argument jest NAN.
__device__ float nanf (const char* tagp)
__device__ double nan (const char* tagp)
Zwraca wartość „nie liczby".

Funkcje matematyczne

__device__ float erff (float x)
__device__ double erf (double x)
Oblicz funkcję błędu argumentu wejściowego.
__device__ float erfcf (float x)
__device__ double erfc (double x)
Oblicz funkcję błędu komplementarnego argumentu wejściowego.
__device__ float erfinvf (float x)
__device__ double erfinv (double x)
Oblicz funkcję błędu odwrotnego argumentu wejściowego.
__device__ float erfcinvf (float x)
__device__ double erfcinv (double x)
Oblicz odwrotną funkcję błędu komplementarnego argumentu wejściowego.
__device__ float erfcxf (float x)
__device__ double erfcx (double x)
Oblicz skalowaną funkcję błędu komplementarnego argumentu wejściowego.

$\frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}dt$

$1 - erf(x)$

$erf^{-1}(x)$

$erfc^{-1}(x)$

$e^{x^2}erfc(x)$

Funkcje matematyczne

__device__ float tgammaf (float x)
__device__ double tgamma (double x)
Oblicz funkcję gamma argumentu wejściowego.
__device__ float lgammaf (float x)
__device__ double lgamma (double x)
Oblicz naturalny logarytm wartości bezwzględnej funkcji gamma argumentu wejściowego.

$\int_0^\infty e^{-t}t^{x-1}dt$

$\ln\left|\int_0^\infty e^{-t}t^{x-1}dt\right|$

Funkcje matematyczne

__device__ float cyl_bessel_i0f (float x)
__device__ double cyl_bessel_i0 (double x)
Oblicz wartość regularnej zmodyfikowanej funkcji cylindrycznej Bessela rzędu 0 dla argumentu wejściowego.
__device__ float cyl_bessel_i1f (float x)
__device__ double cyl_bessel_i1 (double x)
Oblicz wartość regularnej zmodyfikowanej funkcji cylindrycznej Bessela rzędu 1 dla argumentu wejściowego.

Funkcje matematyczne

__device__ float j0f (float x)

__device__ double j0 (double x)

Oblicz wartość funkcji Bessela pierwszego rodzaju rzędu 0 dla argumentu wejściowego.

__device__ float j1f (float x)

__device__ double j1 (double x)

Oblicz wartość funkcji Bessela pierwszego rodzaju rzędu 1 dla argumentu wejściowego.

__device__ float jnf (int n, float x)

__device__ double jn (int n, double x)

Oblicz wartość funkcji Bessela pierwszego rodzaju rzędu n dla argumentu wejściowego.

Funkcje matematyczne

__device__ float y0f (float x)

__device__ double y0 (double x)

Oblicz wartość funkcji Bessela drugiego rodzaju rzędu 0 dla argumentu wejściowego.

__device__ float y1f (float x)

__device__ double y1 (double x)

Oblicz wartość funkcji Bessela drugiego rodzaju rzędu 1 dla argumentu wejściowego.

__device__ float ynf (int n, float x)

__device__ double yn (int n, double x)

Oblicz wartość funkcji Bessela drugiego rodzaju rzędu n dla argumentu wejściowego.

Zbiór Julii

Zbiór Julii to granica pewnej klasy funkcji w zbiorze liczb zespolonych.

Granica ta tworzy fraktal.

Należy iteracyjnie rozwiązać równanie:

$Z_{n+1} = Z_n^2 + C$

ZBIÓR JULII CPU

```
17  #include "../common/book.h"
18  #include "../common/cpu_bitmap.h"
19
20  #define DIM 1000
21
22  struct cuComplex {
34
35  int julia( int x, int y ) {
52
53  void kernel( unsigned char *ptr ){
66
67  int main( void ) {
68      CPUBitmap bitmap( DIM, DIM );
69      unsigned char *ptr = bitmap.get_ptr();
70
71      kernel( ptr );
72
73      bitmap.display_and_exit();
74  }
```


ZBIÓR JULII CPU

```
53  void kernel( unsigned char *ptr ){
54      for (int y=0; y<DIM; y++) {
55          for (int x=0; x<DIM; x++) {
56              int offset = x + y * DIM;
57
58              int juliaValue = julia( x, y );
59              ptr[offset*4 + 0] = 255 * juliaValue;
60              ptr[offset*4 + 1] = 0;
61              ptr[offset*4 + 2] = 0;
62              ptr[offset*4 + 3] = 255;
63          }
64      }
65  }
```


ZBIÓR JULII CPU

```
35  int julia( int x, int y ) {
36      const float scale = 1.5;
37      float jx = scale * (float)(DIM/2 - x)/(DIM/2);
38      float jy = scale * (float)(DIM/2 - y)/(DIM/2);
39
40      cuComplex c(-0.8, 0.156);
41      cuComplex a(jx, jy);
42
43      int i = 0;
44      for (i=0; i<200; i++) {
45          a = a * a + c;
46          if (a.magnitude2() > 1000)
47              return 0;
48      }
49
50      return 1;
51  }
```

ZBIÓR JULII CPU

```
struct cuComplex {
    float   r;
    float   i;
    cuComplex( float a, float b ) : r(a), i(b)  {}
    float magnitude2( void ) { return r * r + i * i; }
    cuComplex operator*(const cuComplex& a) {
        return cuComplex(r*a.r - i*a.i, i*a.r + r*a.i);
    }
    cuComplex operator+(const cuComplex& a) {
        return cuComplex(r+a.r, i+a.i);
    }
};
```


ZBIÓR JULII GPU

```
#include "../common/book.h"
#include "../common/cpu_bitmap.h"

#define DIM 1000

struct cuComplex {
...
__device__ int julia( int x, int y ) {
...
__global__ void kernel( unsigned char *ptr ) {
...
// Wartości wymagane przez procedurę aktualizującą
struct DataBlock {
    unsigned char   *dev_bitmap;
};

int main( void ) {
    DataBlock   data;
    CPUBitmap bitmap( DIM, DIM, &data );
    unsigned char    *dev_bitmap;

    HANDLE_ERROR( cudaMalloc( (void**)&dev_bitmap, bitmap.image_size() ) );
    data.dev_bitmap = dev_bitmap;

    dim3    grid(DIM,DIM);
    kernel<<<grid,>>>( dev_bitmap );

    HANDLE_ERROR( cudaMemcpy( bitmap.get_ptr(), dev_bitmap,
                              bitmap.image_size(),
                              cudaMemcpyDeviceToHost ) );

    HANDLE_ERROR( cudaFree( dev_bitmap ) );

    bitmap.display_and_exit();
}
```


ZBIÓR JULII GPU

```
__global__ void kernel( unsigned char *ptr ) {
    // Odwzorowanie z blockIdx na położenie piksela
    int x = blockIdx.x;
    int y = blockIdx.y;
    int offset = x + y * gridDim.x;

    // Obliczenie wartości dla tego miejsca
    int juliaValue = julia( x, y );
    ptr[offset*4 + 0] = 255 * juliaValue;
    ptr[offset*4 + 1] = 0;
    ptr[offset*4 + 2] = 0;
    ptr[offset*4 + 3] = 255;
}

// Wartości wymagane przez procedurę aktualizującą
struct DataBlock {
    unsigned char   *dev_bitmap;
};
```


ZBIÓR JULII GPU

```
__device__ int julia( int x, int y ) {
    const float scale = 1.5;
    float jx = scale * (float)(DIM/2 - x)/(DIM/2);
    float jy = scale * (float)(DIM/2 - y)/(DIM/2);

    cuComplex c(-0.8, 0.156);
    cuComplex a(jx, jy);

    int i = 0;
    for (i=0; i<200; i++) {
        a = a * a + c;
        if (a.magnitude2() > 1000)
            return 0;
    }

    return 1;
}
```


ZBIÓR JULII GPU

```
struct cuComplex {
    float   r;
    float   i;
    cuComplex( float a, float b ) : r(a), i(b)  {}
    __device__ float magnitude2( void ) {
        return r * r + i * i;
    }
    __device__ cuComplex operator*(const cuComplex& a) {
        return cuComplex(r*a.r - i*a.i, i*a.r + r*a.i);
    }
    __device__ cuComplex operator+(const cuComplex& a) {
        return cuComplex(r+a.r, i+a.i);
    }
};
```


ZBIÓR JULII REZULTAT