

# Paměťový model OpenCL

Michal Kula

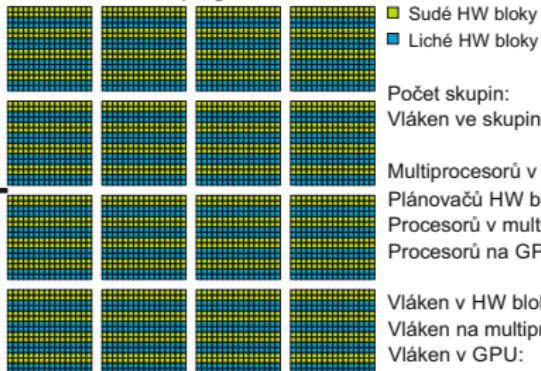
Vysoké učení technické v Brně, Fakulta informačních technologií  
Božetěchova 2, 612 66 Brno  
[www.fit.vutbr.cz/~ikula](http://www.fit.vutbr.cz/~ikula)



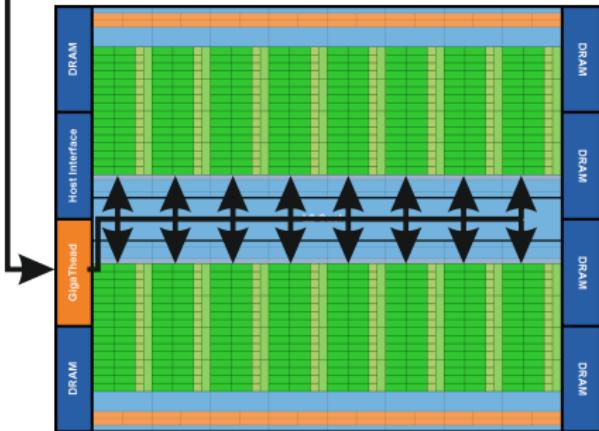
FACULTY  
OF INFORMATION  
TECHNOLOGY

# Princip práce GPU - opakování

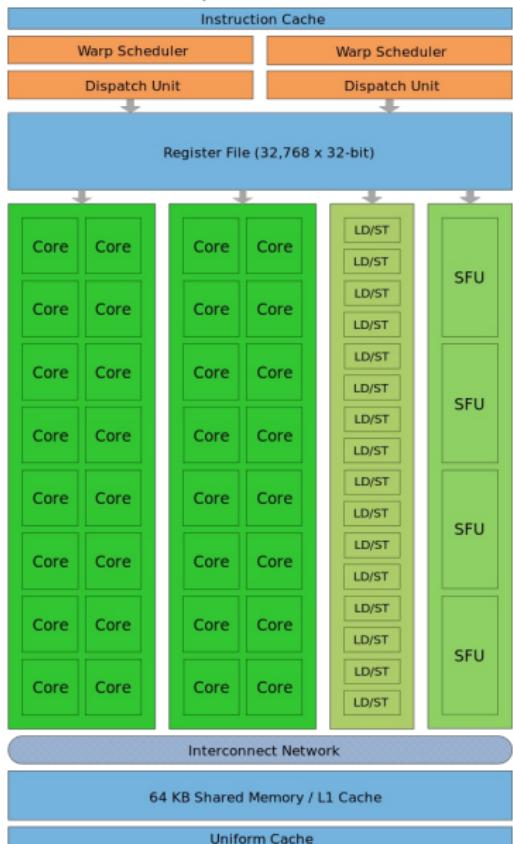
## Vlákna programu



## GPU - Fermi



## Multiprocesor - Fermi



## Sekvenční kód

```
// width = 1023; height = 1025;
// sekvencni kod
void func_2d(int width, int height, ...)
{
    for(int j = 0; j < height; j++)
        for(int i = 0; i < width; i++){
            // funkci kód
        }
}
```

## Sekvenční kód - příprava pro OpenCL

```
// width = 1023; height = 1025;
// global = {1024, 1040};
void func_2d(int width, int height, ...)
{
    size_t local[2] = {16,16}
    size_t global[2] = {align(width,local[0]),
                        align(height, local[1])};
    for(size_t g_y = 0; g_y < global[1]; g_y++)
        for(size_t g_x = 0; g_x < global[0]; g_x++) {
            l_x = g_x % local[0];
            l_y = g_y % local[1];
            l_w = local[0];
            l_h = local[1];
            if((g_x < width) && (g_y < height)){
                // funkcní kod
            }
        }
}
```

```
// width = 1023; height = 1025;
// global = {1024, 1040};
cl::NDRange local(16, 16);
cl::NDRange global(alignTo(width, local[0]),
                    alignTo(height, local[1]));
cl::Kernel kernel(program, "func_2d", ...);
...
queue.enqueueNDRangeKernel(kernel, cl::NullRange, global, local, ...);

// Compiler kod
__kernel void func_2d(...)

{
    size_t g_x = get_global_id(0);
    size_t g_y = get_global_id(1);
    size_t l_x = get_local_id(0);
    size_t l_y = get_local_id(1);
    size_t l_w = get_local_size(0);
    size_t l_h = get_local_size(1);
    if((g_x < width) && (g_y < height)){
        // funkcní kod
    }
}
```

## Omezení

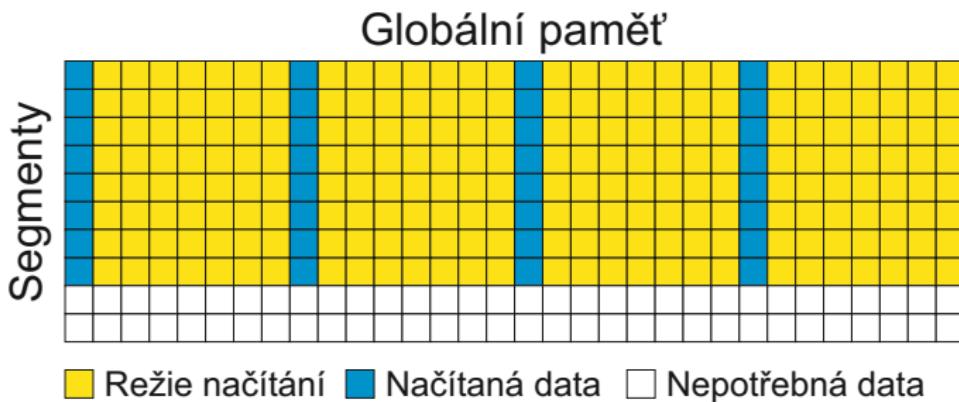
- *global[x]* musí být násobkem *local[x]*
- není zaručeno vyhodnocení pracovních skupin v pořadí
- obousměrná komunikace mezi vlákny - pouze v rámci skupiny pomocí lokální paměti a synchronizace
- jednosměrná komunikace mezi vlákny - pomocí globální paměti a atomických instrukcí

Typ	Umístění	Rychlosť	Cache	Velikost	Dostupnost
global	off-chip	0.125B/tick	1	3072 MB	vše
local	on-chip	1 B/tick	0	0.72 MB	skupina
constant	off-chip	4 B/tick	1	64 kB	vše
register	on-chip	8 B/tick	0	4 MB	vlákno

## Globální paměť

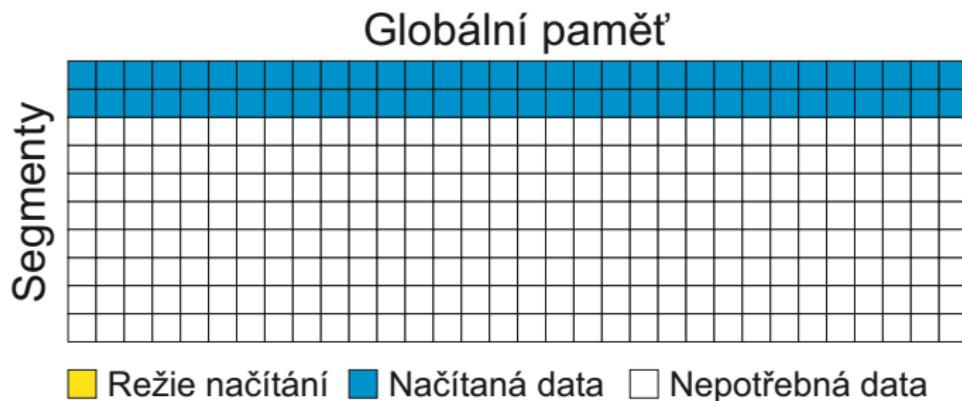
- hlavní paměť grafického adaptéru
- velikost v řádech GB
- latence v řádu stovek cyklů - nutnost běhu mnoha warpů pro překrytí
- desetinová rychlosť oproti lokální paměti nebo registrům
- u novějších architektur cachováno pomocí L2 cache, případně i L1 cache
- adresový prostor `__global`
- paměť rozdělena po segmentech
- pokud je na jednu adresu ze segmentu přistoupeno z warpu přečte se celý segment
- paměťové segmenty se pohybují mezi 32B a 128B
- u starších architektur nezarovnaný přístup - serializace
- pro zachování zarovnanosti je jedním vláknem možné načítat  $\leq 16B$
- problém s nezarovnaností se zmírňuje pomocí cache paměti

```
// příklad nezarovnaného přístupu
__kernel void add(__global int *a, __global int *b)
{
    int g_x = get_global_id(0);
    for(int i = 0; i < 8; i++){
        a[g_x * 8 + i] += b[g_x * 8 + i]
    }
}
```



## Globální paměť

```
// příklad zarovnaného přístupu
__kernel void add{__global int *a, __global int *b}
{
    int g_x = get_global_id(0);
    int g_w = get_global_size(0);
    for(int i = 0; i < 8; i++){
        a[g_x + i * g_w] += b[g_x + i * g_w]
    }
}
```

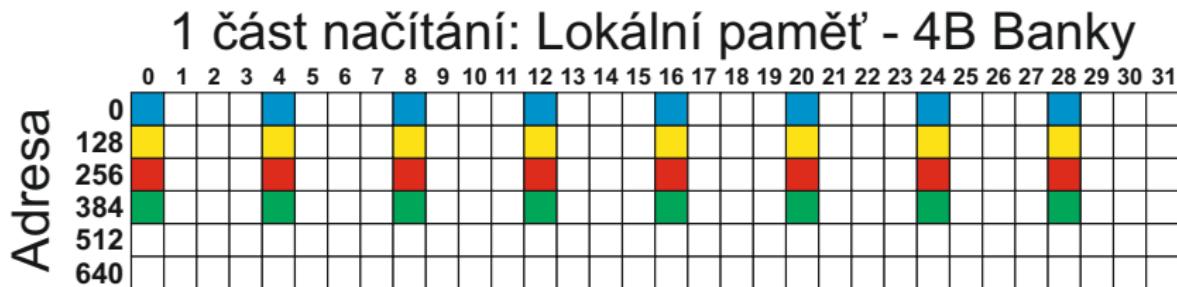


## Lokální paměť

- minimální velikost je pro OpenCL 1.1 32kB
- typicky v multiproc. staré GPU 32-48kB, nové GPU 64 - 96kB - často nelze nalokovat celé/skupinu
- u CPU a některých starších GPU emulovaná pomocí globální paměti - lze nahradit vyhrazeným prostorem v globální paměti
- rozdělena do banků s velikostí buňky typicky 4B
- přístupu do stejného banku více vlákny z warpu - serializace
- novější GPU umějí broadcast a multicast - přístup z více vláken z warpu na stejnou adresu nezpůsobuje bank konflikty
- použití: kooperace mezi vlákny, uživatelské cachování ...
- adresový prostor `__local`

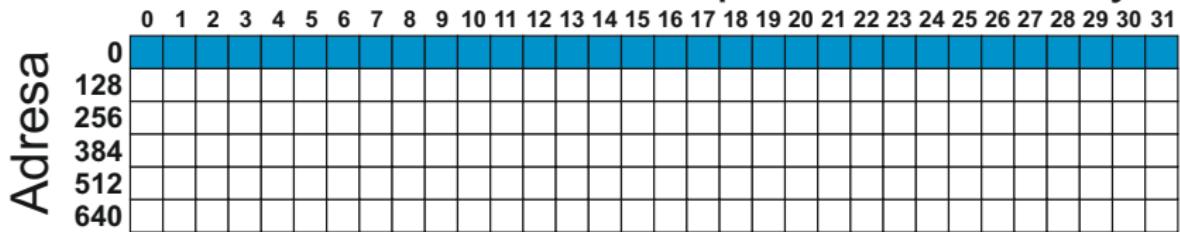
- velikost paměťové buňky banku je 4B
- posun mezi vlákny z warpu je 16B
- rozdělení na 4 nezávislá načítání

```
__kernel void proc_data(__global float4 *data,
                       __local float4 *l_data, ...){
    g_x = get_global_id(0);
    l_x = get_local_id(0);
    l_data[l_x].x = data[g_x].x;
    l_data[l_x].y = data[g_x].y;
    l_data[l_x].z = data[g_x].z;
    l_data[l_x].w = data[g_x].w;
    ...
}
```



```
__kernel void proc_data(__global float4 *data,
                       __local float *l_data, ...){
    g_x = get_global_id(0);
    l_x = get_local_id(0);
    g_w = get_local_size(0);
    float4 tmp_data = data[g_x];
    l_data[g_w * 0 + l_x] = tmp_data.x;
    l_data[g_w * 1 + l_x] = tmp_data.y;
    l_data[g_w * 2 + l_x] = tmp_data.z;
    l_data[g_w * 3 + l_x] = tmp_data.w;
    ...
}
```

## 1 část načítání: Lokální paměť - 4B Banky



- Každé vlákno načítá hodnoty s rozestupem skupiny
- Zarovnaný přístup do globální paměti
- Bez bankových konfliktů v lokální paměti

```
__kernel void proc_data(__global int *data,
                      __local int *tmp_data,
                      int data_w) {
    g_x = get_global_id(0);
    l_x = get_local_id(0);
    l_w = get_local_size(0);
    for(int i = l_x; i < data_w; i += l_w) {
        tmp_data[i] = data[i];
    }
    barrier(CLK_LOCAL_MEM_FENCE);
}
```

Lokální paměť

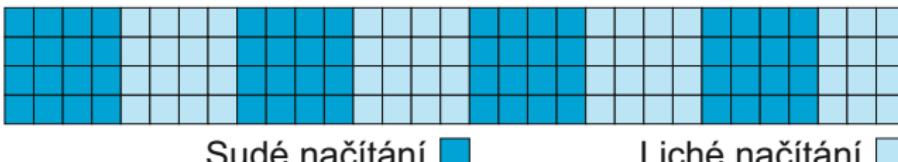


- Každé vlákno načítá hodnoty s rozestupem skupiny
- Zarovnaný přístup do globální paměti
- Bez bankových konfliktů v lokální paměti

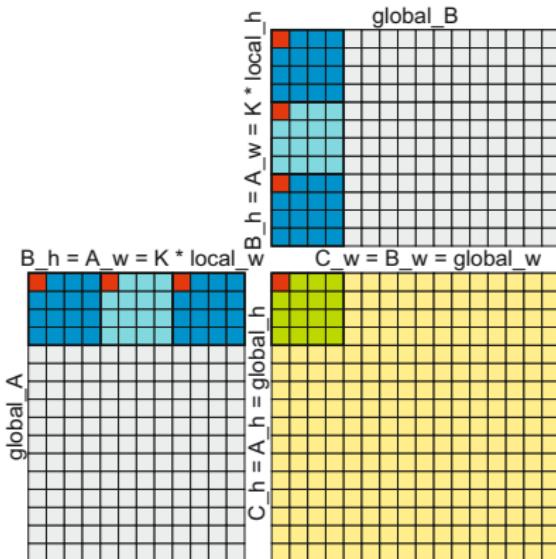
Příklad: Načítání horizontálního pásu maticce pomocí skupiny

```
__kernel void proc_data(__global int *global_data,
                       __local int *local_data, int data_w) {
    g_x = get_global_id(0);
    g_y = get_global_id(1);
    l_x = get_local_id(0);
    l_y = get_local_id(1);
    l_w = get_local_size(0);
    for(int i = 0; i < data_w; i += l_w) {
        local_data[l_x + l_y * l_w] = global_data[i + l_x + g_y * data_w];
        barrier(CLK_LOCAL_MEM_FENCE);
        ... // pouziti lokalni pameti local_data
        barrier(CLK_LOCAL_MEM_FENCE);
    }
}
```

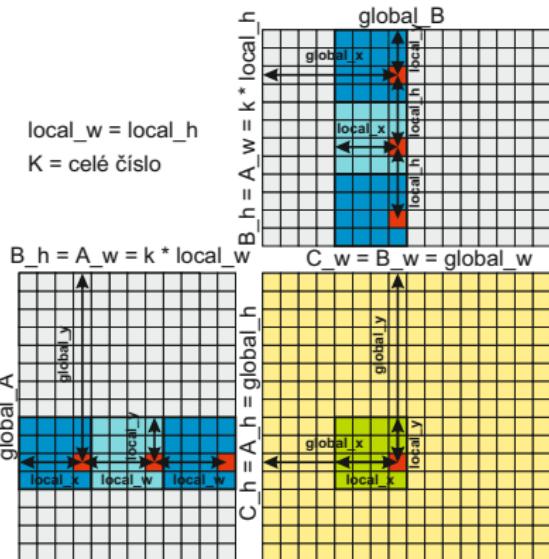
Lokální paměť



pracovní skupina [0,0] vlákno [0,0]



pracovní skupina [1,2], vlákno [3,2]



Prvky výstupní matice c aktuální pracovní skupiny

Prvky výstupní matice c ostatních pracovních skupin

Prvky vstupní matice aktuální pracovní skupiny - sudé načítání do lokální paměti

Prvky vstupní matice aktuální pracovní skupiny - liché načítání do lokální paměti

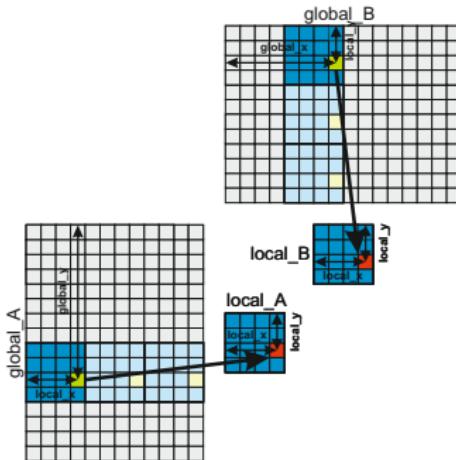
Prvky vstupní matice nepoužité v této pracovní skupině

Prvky vstupní matice zapisované aktuálním vláknem do lokální paměti

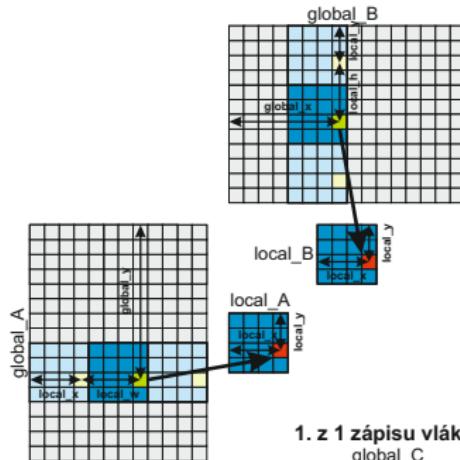


$$c_{x,y} = a_{0,y}b_{x,0} + a_{1,y}b_{x,1} + \dots + a_{N-1,y}b_{x,N-1}$$

## 1. z 3 načtení skupinou do lokální paměti



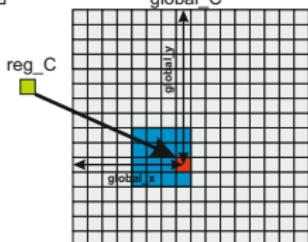
## 2. z 3 načtení skupinou do lokální paměti



## 1. z 4 přičtení do registru vláknem 2. z 4 přičtení do registru vláknem



## 1. z 1 zápisu vláknem



Paměť zapisovaná aktuálním vláknem u aktuální operace

Paměť zapisovaná aktuálním vláknem v ostatních krocích aktuální operace

Paměť čtená aktuálním vláknem u aktuální operace

Paměť čtená aktuálním vláknem v ostatních krocích aktuální operace

Paměť čtená/zapisovaná ostatními vláknami ze skupiny u aktuální operace

Paměť čtená/zapisovaná ostatními vláknami ze skupiny v ostatních krocích aktuální operace



Příklad: násobení matic o rozměrech:

- Matice A: 512x512
- Matice B: 512x512
- Matice C: 512x512

Pracovní skupiny o rozměrech:

- 16x16

Načítání z globální paměti bez použití lokální paměti:

- $512 \times 512 \times 2 \times 512 \times 4B = 1024MB$

Načítání z globální paměti s použitím lokální paměti:

- $512 \times 512 \times 32 \times 2 \times 4B = 64MB$

Vaším úkolem je

- doplnit kernel `matrix_mul_local` pro násobení matic:  $C = A \times B$  pomocí lokální paměti - 3 body
- velikost skupiny je vždy čtvercová
- šířka matice A a výška matice B je zarovnaná na násobek velikosti skupiny

Testy

- ① matice C:  $512 \times 512$  šířka A a výška B 16
- ② matice C:  $512 \times 512$  šířka A a výška B 32
- ③ matice C:  $512 \times 512$  šířka A a výška B 512
- ④ matice C:  $511 \times 513$  šířka A a výška B 512
- ⑤ matice C:  $1024 \times 1024$  šířka A a výška B 1024

Odkazy:

- khronos - OpenCL 1.2 - Reference pages
- khronos - OpenCL 1.2 - Specification
- khronos - OpenCL 1.2 C++ binding - Specification

Děkuji za pozornost!