GPGPU - Programování pro GPU

Tomáš Halada

4. 5. 2021

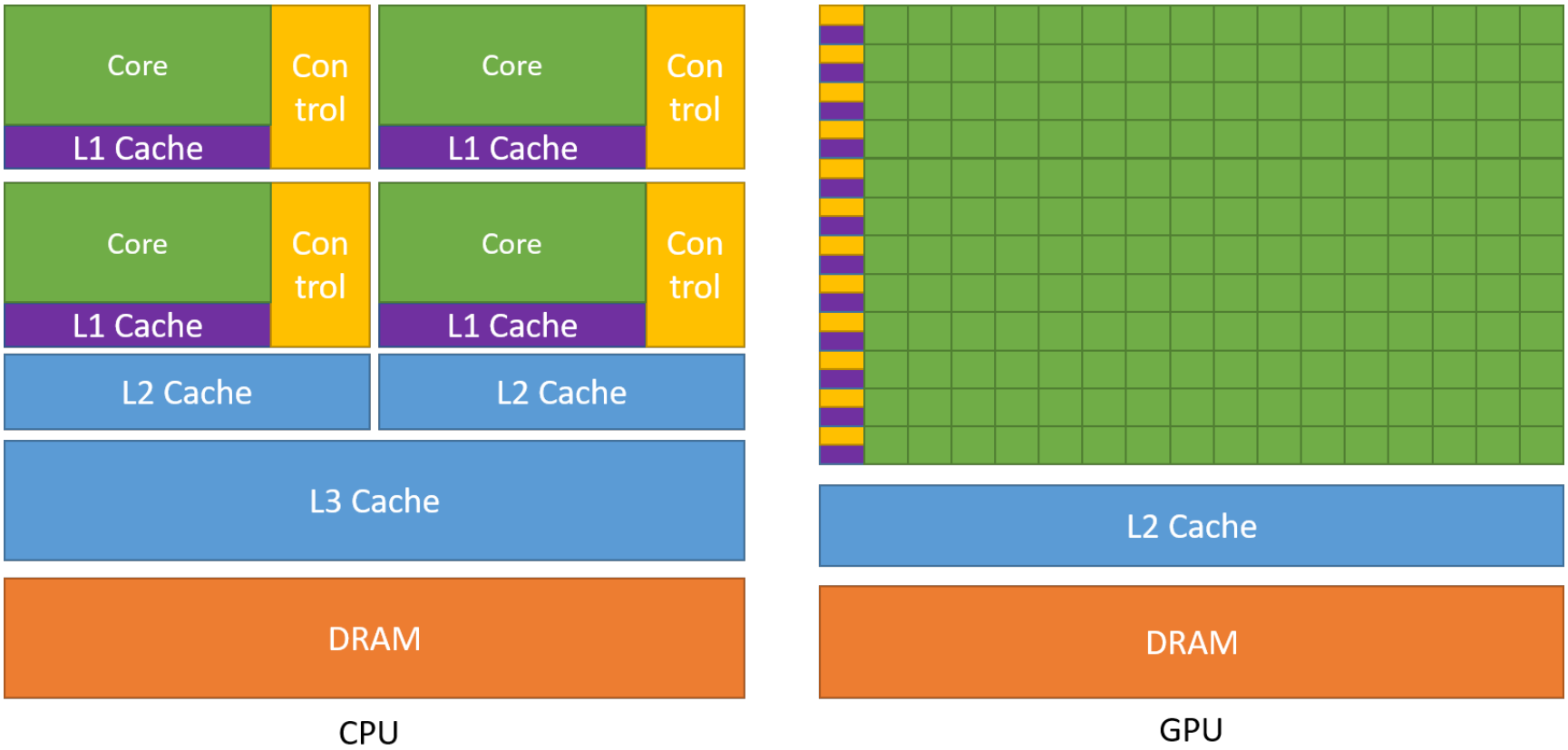
GPU x CPU - parametry

	Nvidia V100	Intel Xeon E5-4660
Cores	5120 @ 1.3GHz	16 @ 3.0GHz
Peak perf.	15.7/7.8 TFlops	0.4 / 0.2 TFlops
Max. RAM	32 GB	1.5 TB
Memory bw.	900 GB/s	68 GB/s
TDP	300 W	120 W

$\approx 8,000$ \$

zdroj: *tnl-project.org*

GPU x CPU - architektura



zdroj: *docs.nvidia.com*

GPGPU

GPGPU - General-Purpose Computing on Graphics Processing Units
gpgpu.org

CUDA - Compute Unified Device Architecture (Nvidia)
(alternativou OpenCL (univerzální), Vulkan (AMD))

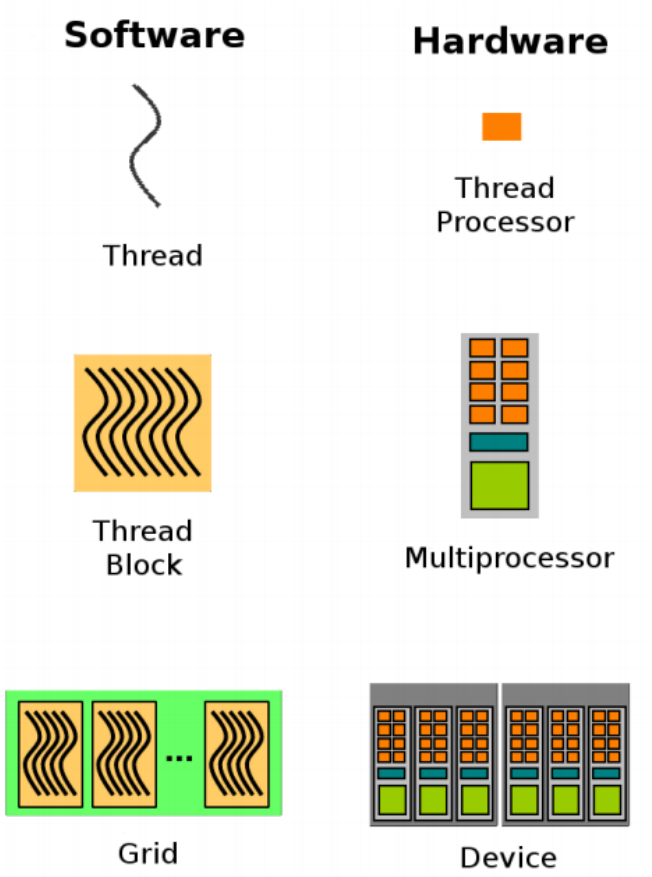
- zjednušení programování GPU (odstraňuje nutnost práce pomocí textur)
- rozšíření jazyka C, C++
- funční pouze na kartách Nvidia

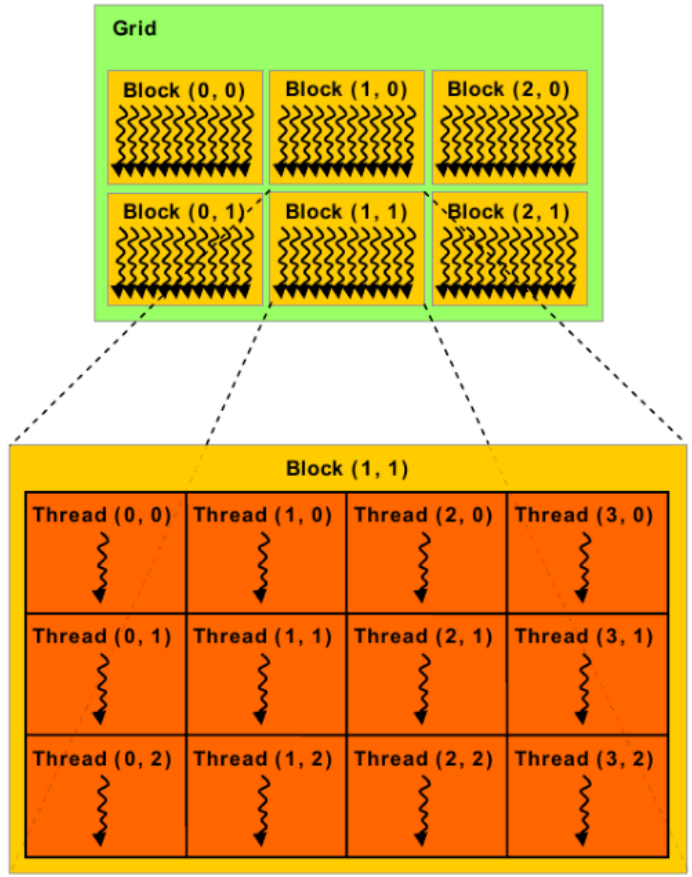
GPGPU - charakteristika

- žádná (velmi malá) datová závislost
- velké množství číselných operací
- současný běh tisíců nezávislých vláken (virtuálně stovky tisíc)
- rychlý sekvenční přístup do paměti
- bez spekulativního zpracování podmínek

GPU - architektura

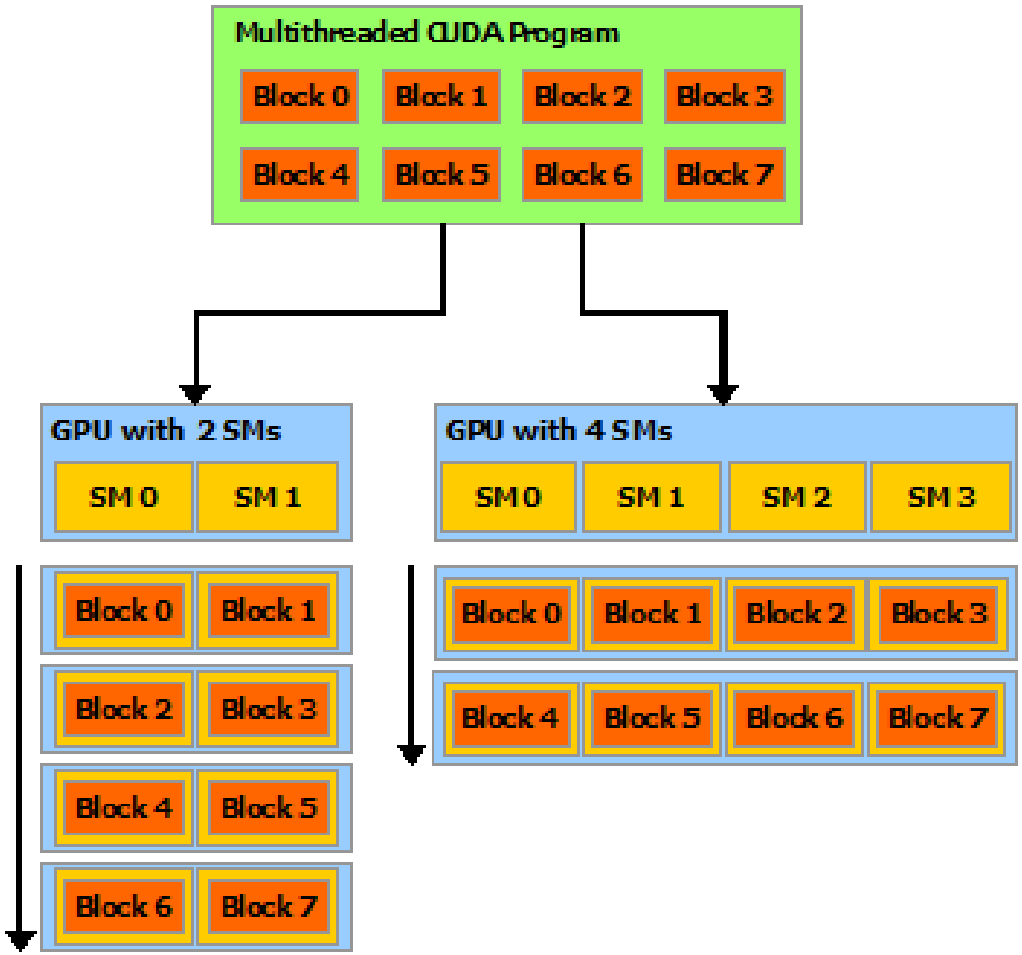
GPU - architektura





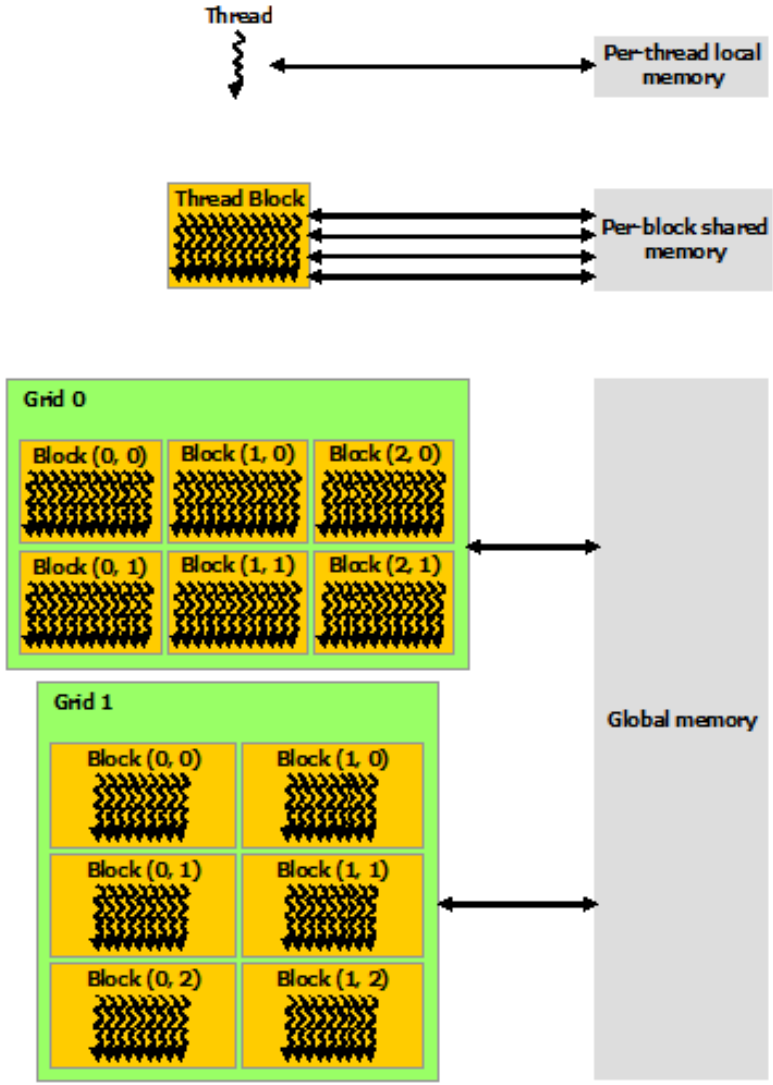
zdroj: *docs.nvidia.com, CUDA by examples*

GPU - architektura



zdroj: *docs.nvidia.com*

GPU - paměť



zdroj: *docs.nvidia.com*

Indexování vláken

```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float* C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}

int main()
{
    ...
    // Kernel invocation with N threads
    VecAdd<<<1, N>>>(A, B, C);
    ...
}
```

Indexování vláken a bloků

```
// Kernel definition
__global__ void MatAdd(float A[N][N], float B[N][N],
                       float C[N][N])
{
    int i = threadIdx.x;
    int j = threadIdx.y;
    C[i][j] = A[i][j] + B[i][j];
}

int main()
{
    ...
    // Kernel invocation with one block of N * N * 1 threads
    int numBlocks = 1;
    dim3 threadsPerBlock(N, N);
    MatAdd<<<numBlocks, threadsPerBlock>>>(A, B, C);
    ...
}
```

Funkce na CPU

```
// SIMULATION upwind2D;

// for(int x = 0; x < steps_x; x ++)
// for(int y = 0; y < steps_y; y ++)

template <typename SCHEME>
void Step(SIMULATION &sim, int x, int y)
{

        if (x != 0 && y != 0)
                SCHEME::update(sim, x, y);
}
```

Funkce na CPU/GPU

```
template <typename SCHEME>
#ifdef USE_CUDA
__global__ void cudaStep(SIMULATION &sim)
#else
void Step(SIMULATION &sim, int x, int y)
#endif
{
        #ifdef USE_CUDA
        int x = threadIdx.x + blockIdx.x * blockDim.x;
        int y = threadIdx.y + blockIdx.y * blockDim.y;
        #endif

        if (x != 0 && y != 0)
                SCHEME::update(sim, x, y);
}
```

Příklad - 2D advekce

$$u_t + 1u_x + 2u_y = 0, \quad \Omega = [0,1] \times [0,1]$$
$$u(x,y,0) = 0, \quad \Gamma_1 : u(x,0,t) = 1, \quad \Gamma_2 : u(0,y,t) = -1$$

Metoda konečných objemů:

$$\mathrm{u}_i^{n+1} = \mathrm{u}_i^n - \frac{\Delta t}{|\Omega_i|} \sum_{j \in \mathcal{N}_i} \mathrm{f}(\mathrm{u}_i^n, \mathrm{u}_j^n, \vec{n}_{ij}) |\Gamma_{ij}|$$

Protiproudé schéma:

$$\mathrm{u}_{i,j}^{n+1} = \mathrm{u}_{i,j}^n - \frac{\Delta t}{\Delta x \Delta y} \left[a\left(\mathrm{u}_{i,j}^n - \mathrm{u}_{i-1,j}^n\right)\Delta y + b\left(\mathrm{u}_{i,j}^n - \mathrm{u}_{i,j-1}^n\right)\Delta x \right]$$

Časový krok:

$$\Delta t = 0.95 \cdot \mathbf{min}\left(\frac{\Delta x}{a}, \frac{\Delta y}{b}, \frac{\Delta x \Delta y}{a\Delta y + b\Delta x}\right)$$

Template Numerical Library

"TNL aims to be STL for HPC."

- BLAS-like functions enclosed to templated vectors and expression templates for simple and efficient vector operations
- Flexible parallel reduction for GPUs and multicore CPUs
- Matrix formats - dense, diagonal, tridiagonal, multidiagonal, COO (CPU only), CSR, Ellpack, Sliced Ellpack
- Solvers - CG, BiCGStab,GMRES, parallel CWYGMRES, TFQMR, Jacobi, SOR (CPU only)
- Preconditioners - Jacobi, ILU0 (CPU only), ILUT(CPU only)
- Runge-Kutta solvers - Euler (first order), Runge-Kutta-Merson (fourth order adaptive)
- Orthogonal numerical grids and unstructured numerical meshes

zdroj: *tnl-project.org*

TNL vector

Co máme k dispozici?

- std::vector< Type, Allocator>
- TNL::Vector< RealType, DeviceType, IndexType, Allocator>

Co získáváme?

- sčítání, násobení
- optimalizovanou redukci (skalární součin, norma, suma...)

GPU pro všední den - python a GPU

Numpy a SciPy na GPU?

- Numba
- Cupy
- (Pytorch)

CuPy

Odkazy a reference

[1] Cuda Toolkit documentation:
https://docs.nvidia.com/cuda/cuda-c-programming-guide/#abstract
[2] Cuda by examples:
http://www.mat.unimi.it/users/sansotte/cuda/CUDA_by_Example.pdf

[3] Template Numerical Library:
https://tnl-project.org/
[4] Podklady KM FJFI