Využití GPU k urychlení učení neuronových sítí

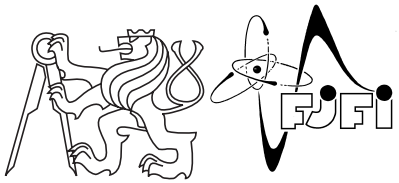
Seminář strojového učení a modelování

V. Španihel

Katedra matematiky

Fakulta jaderná a fyzikálně inženýrská

České vysoké učení technické v Praze



24.10.2013

Obsah

Paralelní programování na GPU

NVIDIA CUDA

- Přehledné rozhranní
- Pouze pro karty od firmy NVIDIA

OpenCL

- Hardwarově nezávislé
- Složitější rozhranní

Co je potřeba

Software

- CUDA Runtime - Nutné pro spuštění CUDA aplikace
- CUDA Toolkit - Pro vývoj(všechny knihovny a dokumentace)
- Podporovaný operační systém s C/C++ překladačem
- Volitelně GPU Computing SDK - Stovky příkladů

Hardware

- Grafická karta NVIDIA podporující CUDA architekturu

Základní pojmy

Pojmy

- Host - Označení pro CPU
- Device - Označení pro GPU
- Kernel - Označení metody spouštěné na GPU
- Vlákno - Jedno volání metody
- Blok - Až trojrozměrná struktura vláken
- Warp - Soubor vláken zpracovávaných paralelně v rámci bloku
- Mřížka - Až trojrozměrná struktura bloků

Výpočetní schopnost (Compute capability)

Vysvětlení

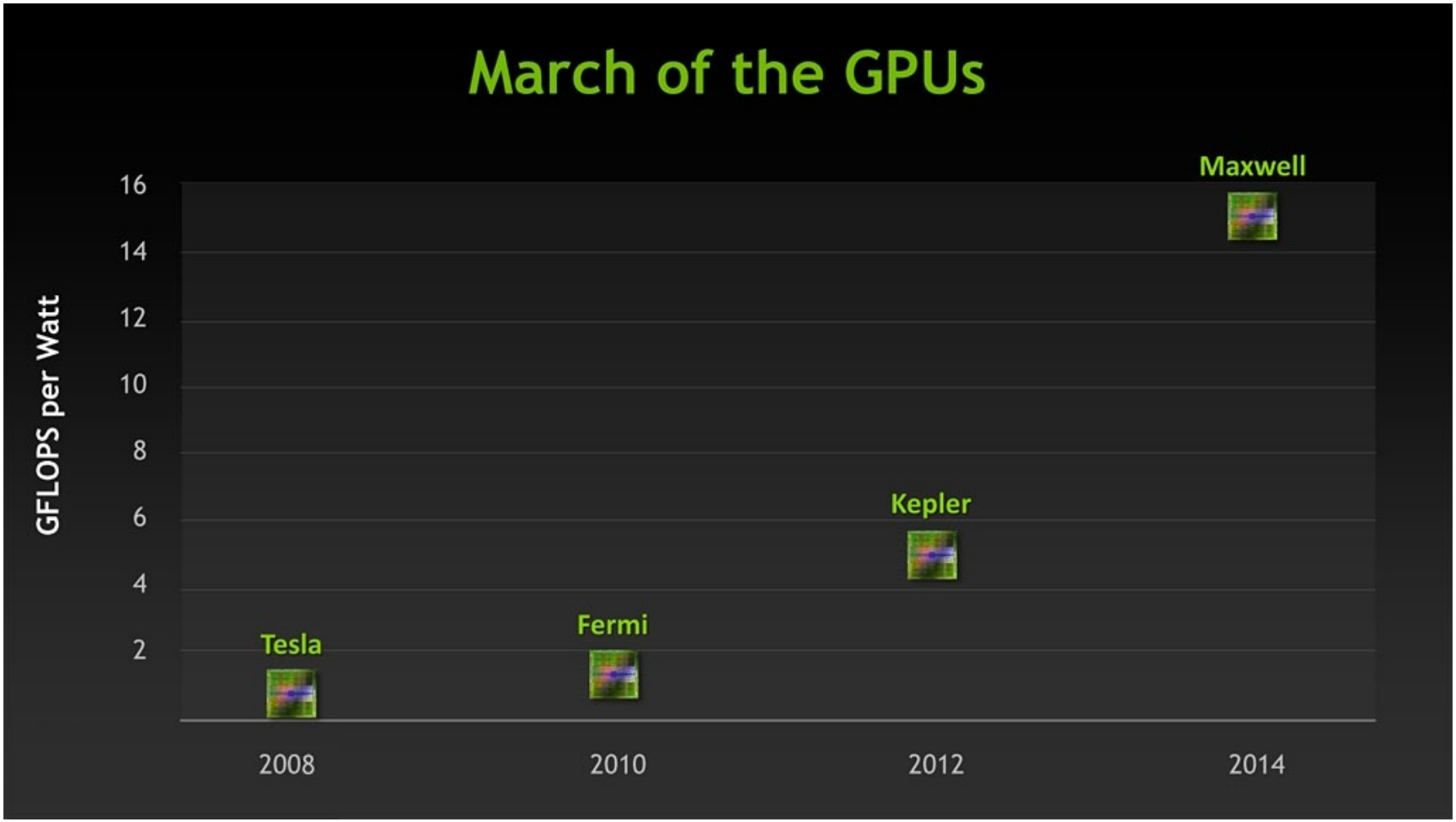
- Specifikace verze CUDA architektury (např. CC = 2.1)
- Příklad odlišností: maximální počet bloků v gridu a vláken v bloku

Výpočty na GPU

Vývoj GPU architektur v čase

- 2008: Tesla - Výpočetní schopnost = 1.x
- 2010: Fermi - Výpočetní schopnost = 2.x
- 2012: Kepler - Výpočetní schopnost = 3.x
- 2014: Maxwell - Výpočetní schopnost = 4.x

Vývoj architektur



Historický přehled vybraných grafických karet

Název	Rok	CC	Počet jader	Paměť	Rozhranní
GeForce GTX 480	2010	2.0	480	1.5GB GDDR5	384-bit
GeForce GTX 680	2012	3.0	1536	2GB GDDR5	256-bit
Tesla M2090	2011	2.0	512	6 GB GDDR5	384-bit
Tesla K20	2012	3.5	2496	5 GB GDDR5	384-bit
Quadro 2000D	2010	2.1	192	1 GB GDDR5	128-bit
Quadro K5000	2012	3.0	1536	4GB GDDR5	256-bit

Název	Rychlost paměti	Výkon SP	Cena
GeForce GTX 480	177.4 GB/s	1.345 TFlops	6 000 Kč
GeForce GTX 680	192.2 GB/s	3.09 TFlops	12 000 Kč
Tesla M2090	177 GB/s	1.33 TFlops	77 000 Kč
Tesla K20	208 GB/s	3.52 TFlops	100 000 Kč
Quadro 2000D	41.6 GB/s	?	10 000 Kč
Quadro K5000	173 GB/s	2.1 TFlops	40 000 Kč

Klasifikace GPU

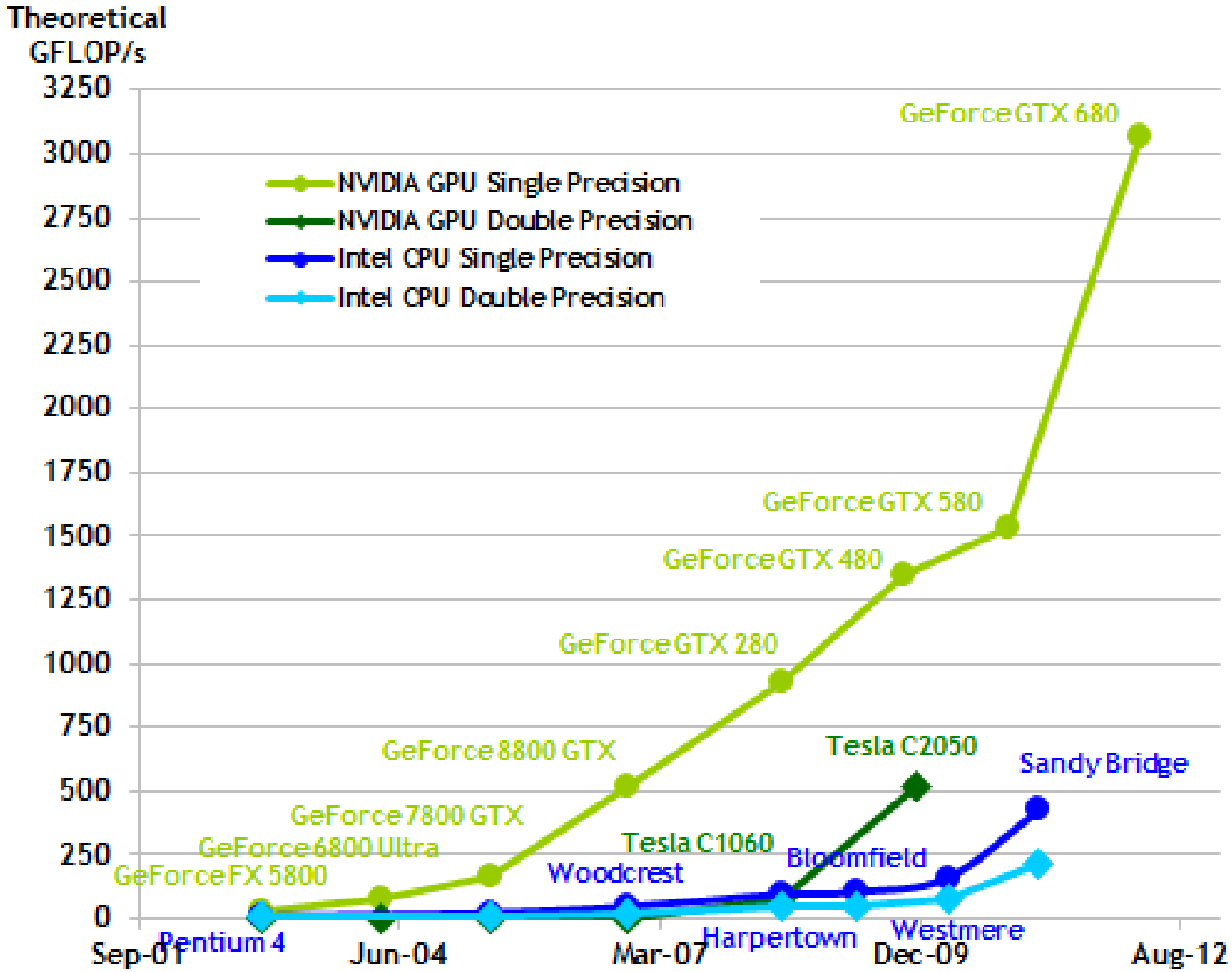
Flynnova taxonomie - Klasifikace počítačových architektur

	Single Instruction	Multiple Instruction
Single Data	SISD	MISD
Multiple Data	**SIMD**	MIMD

Řady karet NVIDIA

- GeForce - pomalá ve dvojité přesnosti
- Tesla - profesionální, rychlá dvojitá přesnost
- Quadro - určena spíše pro grafiku

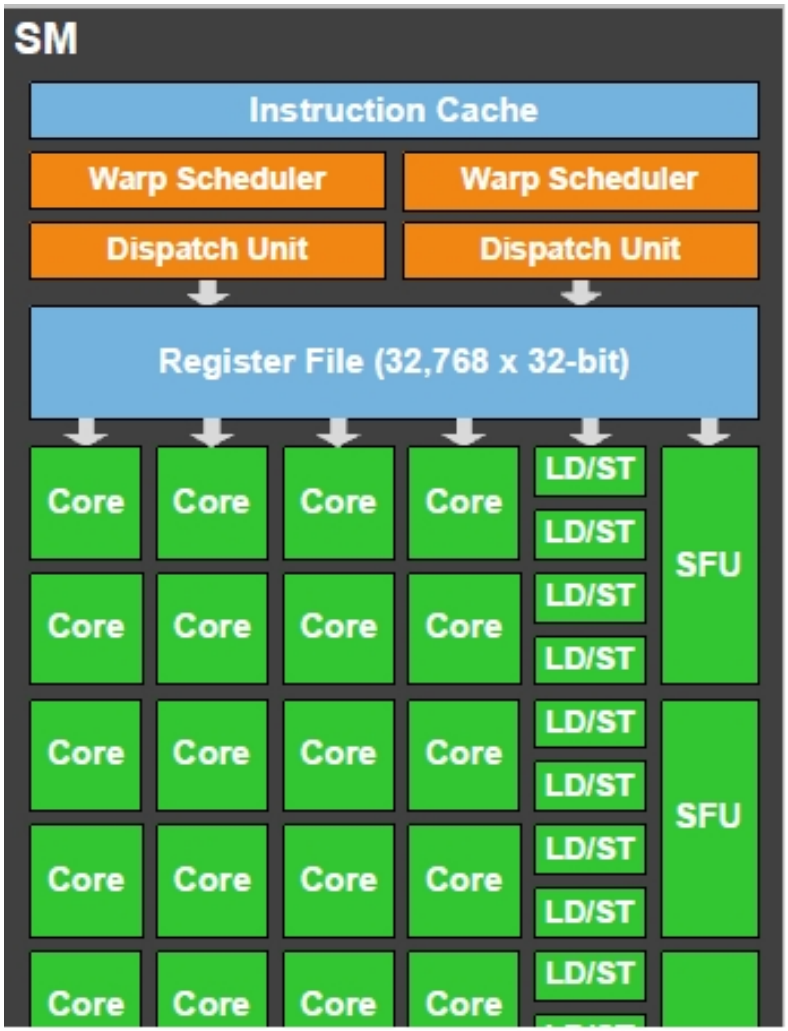
Porovnání výkonu



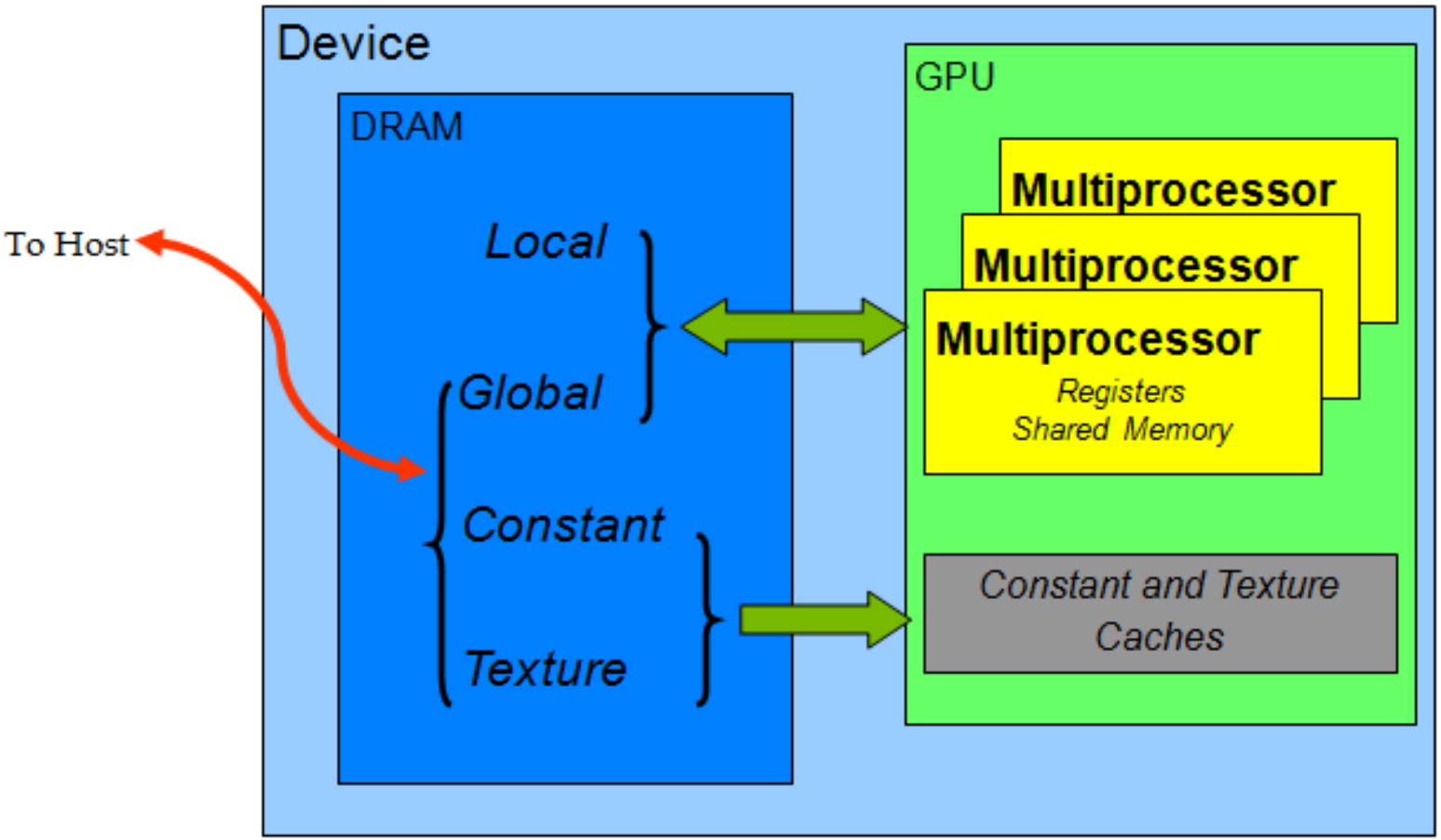
Architektura GPU

Fermi

- SM - Multiprocesor s CUDA jádry
- SFU - Zpracování složitých operací
- LD/ST - Načítání dat



Typy pamětí

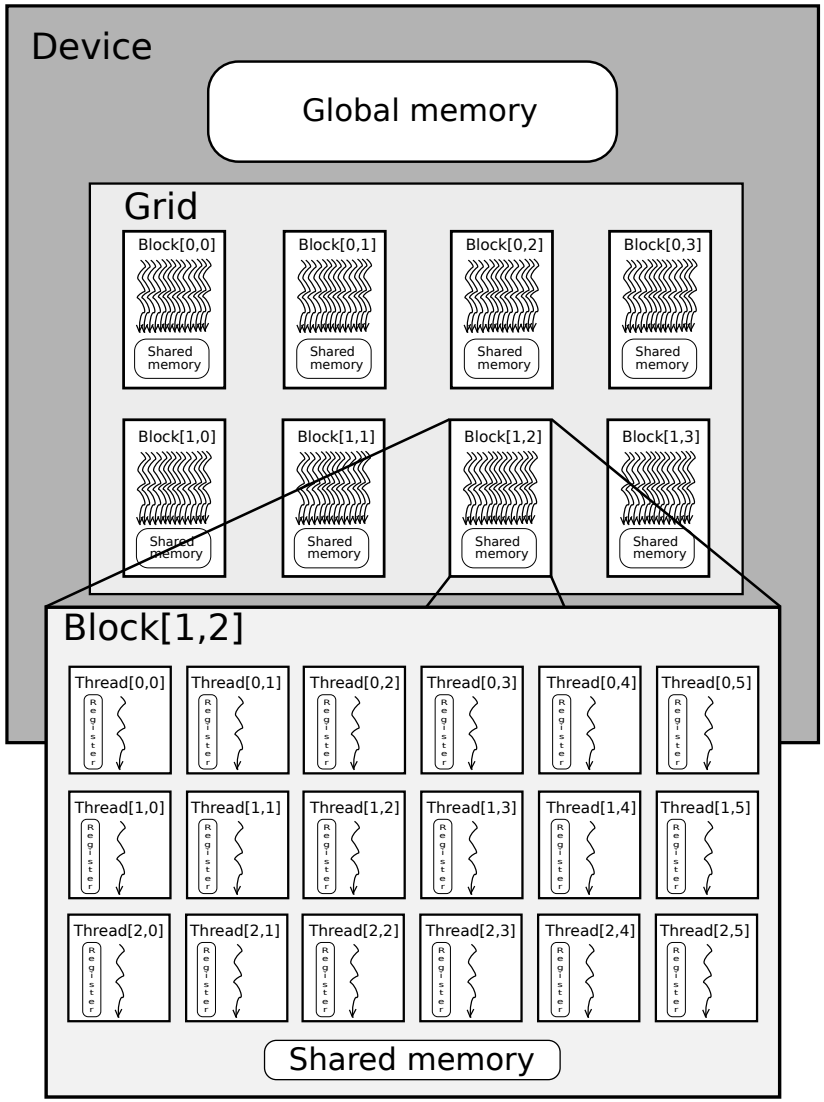

Device
DRAM
Local
Global
Constant
Texture
To Host
GPU
Multiprocessor
Multiprocessor
Multiprocessor
Registers
Shared Memory
Constant and Texture
Caches

Architektura vláken

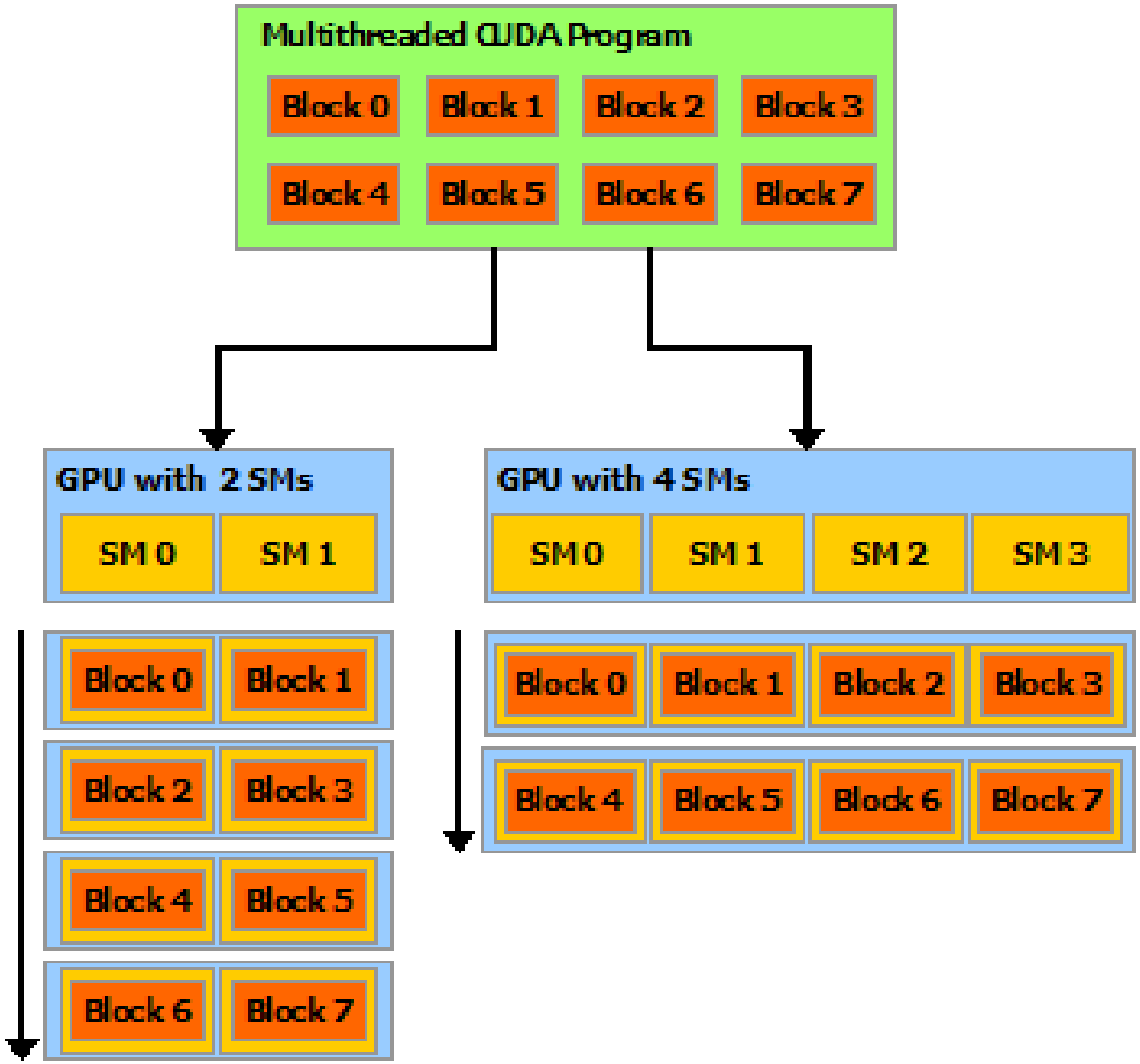
Určení architektury

```
// Urceni dimenzi
dim3 gridD(gX, gY, gZ);
dim3 blockD(bX, bY, bZ);

// Volani kernelu
kernel<<<gridD, blockD>>>(parametry);
```



Přiřazování bloků na SM



Automatické proměnné

Dodefinovaná struktura dim3

- `blockDim.x` - Obsahuje počet vláken uvnitř bloku
- `gridDim.x` - Obsahuje počet bloků uvnitř mřížky
- `blockIdx.x` - Obsahuje index rezidentního bloku
- `threadIdx.x` - Obsahuje index vlákno v bloku

Serializace indexů

```
int idx = blockDim.x * blockIdx.x + threadIdx.x;
```

Jednoduchý příklad C++

```
1   __global__ void fillV(float *v, float c) {
2     int idx = blockDim.x * blockIdx.x + threadIdx.x;
3     if (idx < L) {
4       v[idx] = c;
5     }
6   }
```

```
1   int main(void) {
2     float hV[L];
3     float *dV;
4     for (int i = 0; i < L; i++) hV[i] = 0.0;
5     cudaMalloc((void**) &dV, sizeof(float)*L);
6     cudaMemcpy(dV, &hV, sizeof(float)*L, cudaMemcpyHostToDevice);
7     fillV<<<1,L>>>(dV, 99.1);
8     cudaMemcpy(&hV, dV, sizeof(float)*L, cudaMemcpyDeviceToHost);
9     for (int i = 0; i < L; i++) printf("%f\n", hV[i]);
10    return 0;
11  }
```

Optimalizace

Předcházet rozdělení (Divergenci) warpu

- Vzniká např. kvůli podmínkám typu:

```
if (threadIdx.x < 11) {
  branch1();
} else {
  branch2();
}
```

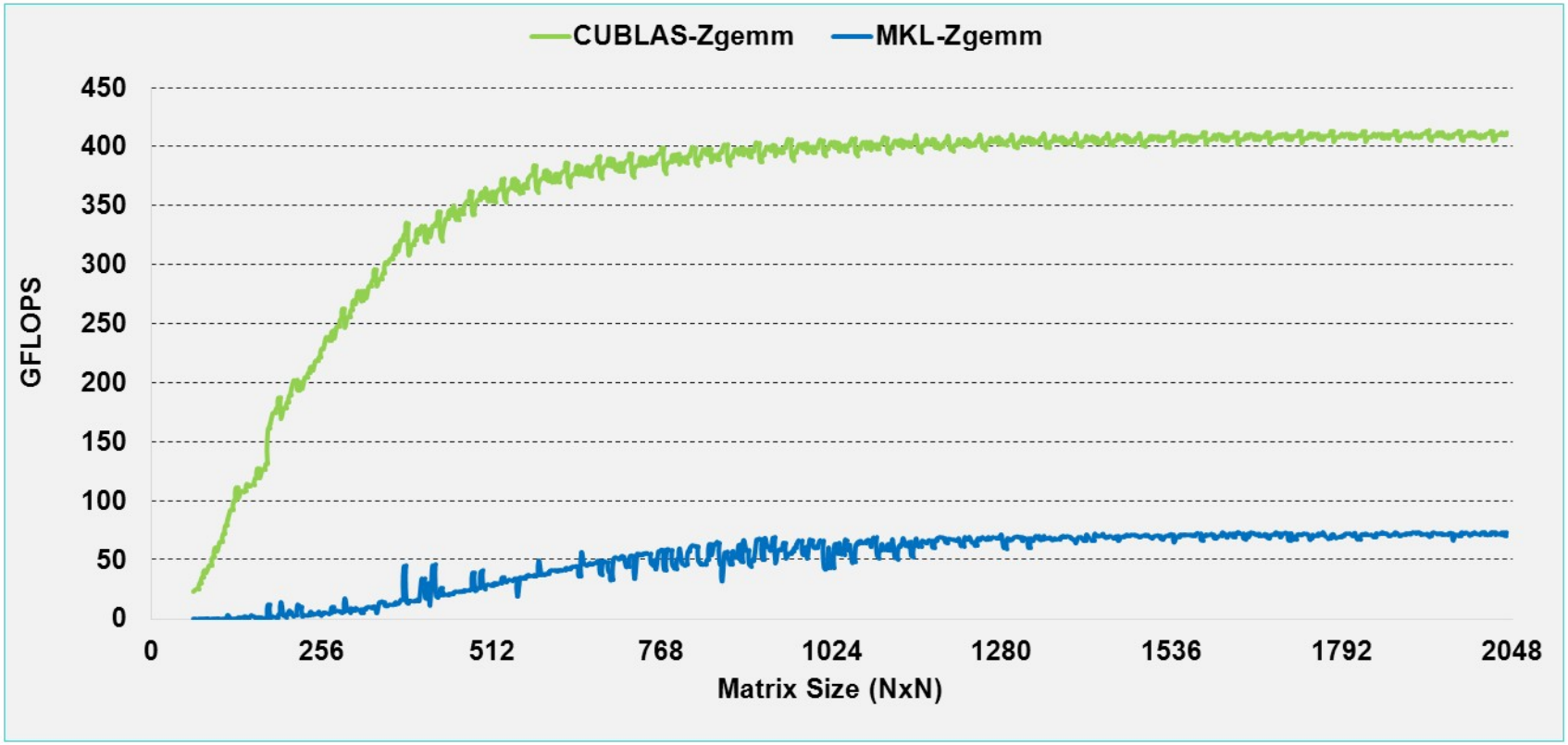
- Dochází k serializaci
- Potřeba provést obě větve
- Část vláken warpu nepracuje, spustí se v další iteraci

Knihovny

Užitečné knihovny

- CUBLAS - CUDA implementace knihovny BLAS
- CUSPARSE - CUDA implementace operací s řídkými maticemi
- MAGMA - CUDA implementace knihovny LAPACK
- Thrust - Paralelní algoritmy pro GPU, efektivnější programování

CUBLAS - porovnání výkonu



- cuBLAS 4.1 on Tesla M2090, ECC on
- MKL 10.2.3, TYAN FT72-B7015 Xeon x5680 Six-Core @ 3.33 GHz

- Performance may vary based on OS ver. and motherboard config.

Maticové násobení na CPU (GLS BLAS)

```
1  int main(void) {
2    float *h_A, *h_B, *h_C;
3    int n2 = N * N;
4    float alfa = 1.0, beta = 0.0;
5    //*
6
7    // Initialize host matrices
8    h_A = new float[n2];
9    h_B = new float[n2];
10   h_C = new float[n2];
11
12   // Compute cblas sgemm
13   cblas_sgemm(CblasColMajor, CblasNoTrans, CblasNoTrans, N, N, N, alfa, h_A,
         N, h_B, N, beta, h_C, N);
14
15   //**
16
17   return 0;
18 }
```

Maticové násobení na GPU (CUBLAS)

```
1   //*
2   float *d_A = 0, *d_B = 0, *d_C = 0;
3   cublasStatus_t status; cublasHandle_t handle;
4
5   //**
6   // Initialize device matrices
7   CUDA_CHECK_RETURN(cudaMalloc((void**) &d_A, n2*sizeof(float)));
8   CUDA_CHECK_RETURN(cudaMalloc((void**) &d_B, n2*sizeof(float)));
9   CUDA_CHECK_RETURN(cudaMalloc((void**) &d_C, n2*sizeof(float)));
10
11  // Transfer data into global memory and fill zeros into h_C
12  cublasSetMatrix(N, N, sizeof(float), h_A, N, d_A, N);
13  cublasSetMatrix(N, N, sizeof(float), h_B, N, d_B, N);
14  CUDA_CHECK_RETURN(cudaMemset(d_C, 0, n2*sizeof(float)));
15
16  /* Initialize CUBLAS */
17  status = cublasCreate(&handle);
18
19  /* Performs operation using cublas */
20  cublasSgemm(handle, CUBLAS_OP_N, CUBLAS_OP_N, N, N, N, &alfa, d_A, N, d_B,
        N, &beta, d_C, N);
21
22  // Clear memory
23  CUDA_CHECK_RETURN(cudaFree(d_A));
24  CUDA_CHECK_RETURN(cudaFree(d_B));
25  CUDA_CHECK_RETURN(cudaFree(d_C));
26  CUDA_CHECK_RETURN(cudaDeviceReset());
27  delete [] h_A; delete [] h_B; delete [] h_C;
```

Neuronová síť s přepínacími jednotkami

Účel

- Separační úlohy
- Aproximace funkcí

Stavební kameny NNSU

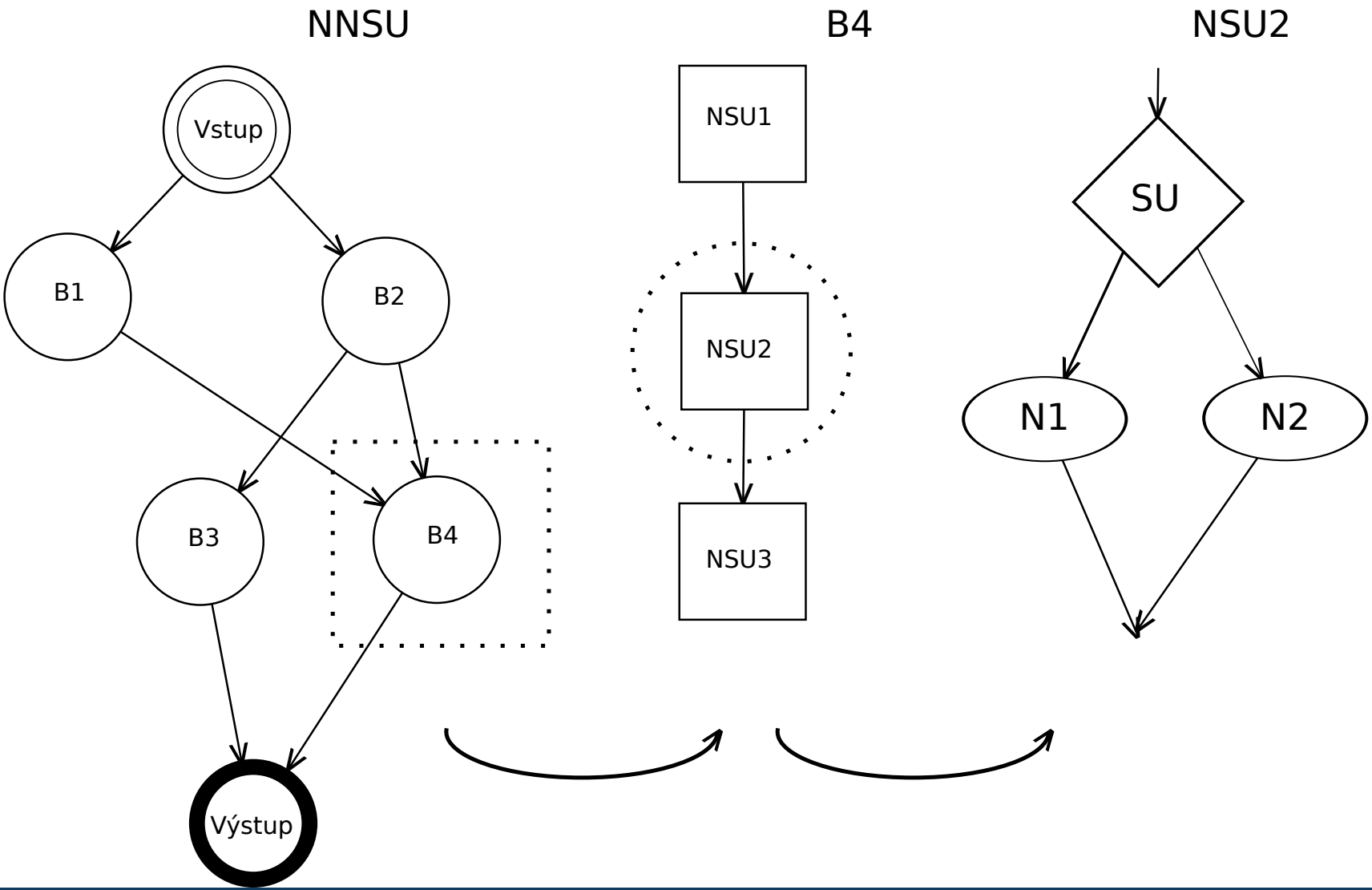
Neuron s přepínací jednotkou (NSU)

- Přepínací jednotka
- Perceptrony
- Vstupy rozdělovány na pereptrony pomocí NSU

Blok

- Zřetězení NSU
- Bloky uspořádány do acyklického grafu

Architektura NNSU - znázornění



Serializace sítě

IP kód

- Reprezentace sítě - Program Symbol Tree's
- Serializace - Readův kód
- Kombinace PST a Readových kódů - IPCode

Učení sítě

První úroveň

- Architektura stromové struktury bloků
- Genetická optimalizace
- Paralelní implementace

Druhá úroveň

- Učení perceptronů každého NSU
- Řešení soustav lineárních rovnic
- Sériová implementace

Integrace CUDA do projektu

Úprava Make souborů

- Předpis pro překlad .cu souborů
- Nastavení cest ke knihovnám
- Vytvoření obalového kódu pro CUDA

Další kroky

Určit práh velikosti úlohy

- Malé úlohy spouštět na CPU
- Rozsáhlejší úlohy spouštět na GPU

Konec

Prostor pro dotazy