Násobení matic a práce se sdílenou pamětí

PCG – Paralelní výpočty na GPU

Cvičení 2, 2021/2022

Jirka Jaroš

Vysoké učení technické v Brně, Fakulta informačních technologií
Božetěchova 1/2, 612 66 Brno - Královo Pole
jarosjir@fit.vutbr.cz

Interaktivní práce CVT

- **Export cest ke CUDA**

```
export PATH=$PATH:/usr/local/share/cuda-11.4/bin
```

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/share/cuda-
11.4/bin:/usr/local/share/cuda-11.4/targets/x86_64-
linux/lib:/usr/local/share/cuda-11.4/extras/CUPTI/lib64
```

- **Pro export cest použijte script**

```
cd Utilities
chmod +x path_export.sh
eval `./path_export.sh`
```

Příprava na práci – Implementace CVT, profil. Karolína

1. **Přeložte knihovnu libwb – bude se hodit pro další ukázky**

```
cd libwb
make -j
make libwb.a
```

2. **Implementaci a testování funkčnosti provádějte v CVT**
3. **Benchmarkování a profilování na Karolíně pomocí skriptu run.pbs**

Interaktivní práce na Karolíně

- **Připojte se na Karolínu**

```
ssh karolina
```

- **Vaše PC: Připojte si disk z Karolíny a nakopírujte tam obsah 1. cvičení**

```
mkdir /tmp/karolina
sshfs karolina: /tmp/karolina
```

- **Nastartujte job (pozor, máme jen 8 uzlů). Kompilovat lze i na loginu**
  - středa – 19.10.

```
qsub -q R1519445 -A DD-22-68 -I -X -l walltime=1:00:0 -l select=1
```

  - pátek – 21.10.

```
qsub -q R1519446 -A DD-22-68 -I -X -l walltime=1:0:0 -l select=1
```

  - Kdykoliv jindy

```
qsub -q qgpu -A DD-22-68 -I -X -l walltime=1:0:0 -l select=1
```

- **Natáhněte moduly**

```
ml CUDA/11.1.1-GCC-10.2.0 Qt5/5.14.2-GCCcore-10.2.0
```

NAIVNÍ NÁSOBENÍ MATIC

Example 1. Naivní násobení matic – Kernel

- **Otevřete soubor template.cu**
  - Doplňte kernel tak, aby provedl maticové násobení **C** = **A** × **B**
  - Matice jsou uloženy v 1D polích po řádcích
  - CUDA grid i block bude mít 2D organizaci
  - **Pozor, je nutné umět vynásobit i obdélníkové matice**
    - Nápověda je v souboru dataset_generator.cpp

```
__global__ void cudaMatrixMatrixMul(float*       C, // [N, P]
                                    const float* A, // [N, M]
                                    const float* B, // [M, P]
                                    const int    N,
                                    const int    M,
                                    const int    P)
{
  // Insert code to implement vector addition here
  // Use a 2D kernel where each thread calculates one element of the C matrix.

   // C[i][j] = sum_i( sum_j( sum_k(A[i][k] * B[k][j])));

}
```

Překlad a spuštění

CVT (bez profil)

```
make
make run{0-5}
```

Karolína (profil)

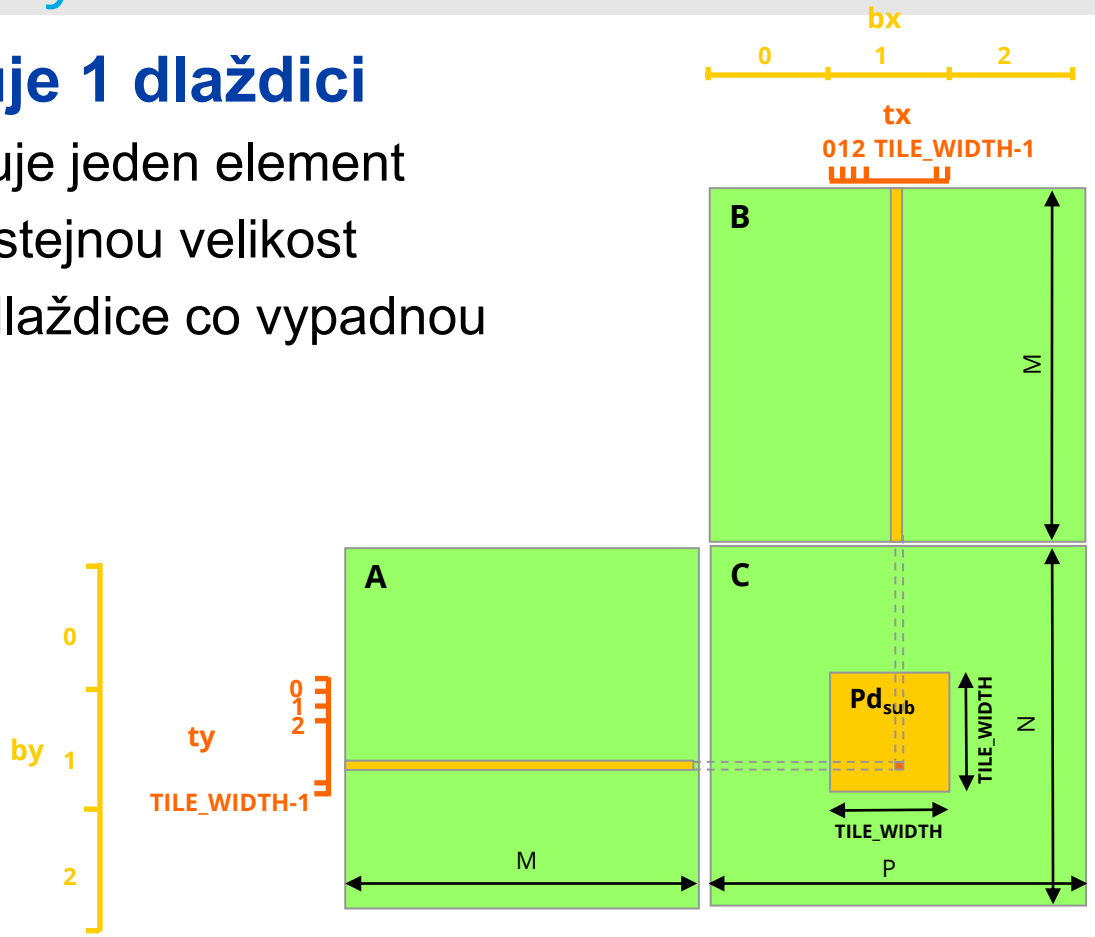
```
qsub run.pbs
```

Upravte soubor:

```
make
make prof{0-5}
```

Example 1. Princip využití více bloků

- **Každý blok zpracuje 1 dlaždici**
  - Každé vlákno zpracuje jeden element
  - Všechny bloky mají stejnou velikost
  - Pozor na vlákna a dlaždice co vypadnou z rozsahu matice!



Example 1. Naivní násobení matic – Pomocné rutiny

- **Otevřete soubor template.cu**
  - Doplňte kód pro alokaci data na GPU
  - Doplňte kód pro kopii data na GPU
  - Nastavte velikost kernelu a spusťte kernel
  - Stáhněte data zpět na CPU

  - Vygenerujte vstupní data
- **Překlad, spuštění a profilování pro různé velikosti problému**

```
# Ladění v CVT
make
make gen
make run{0-5}
```

```
# Profiling Karolína
qsub run.pbs
```

```
make run5
./ex1 -e MatrixMultiplication/Dataset/5/output.raw -i
MatrixMultiplication/Dataset/5/input0.raw,MatrixMulti
plication/Dataset/5/input1.raw  -t matrix
-Example 1: CUDA naive matrix-matrix multiplication -
Matrix A [3072, 3072]
Matrix B [3072, 3072]
Matrix C [3072, 3072]
Time to calculate the answer:  756.4 ms
==$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
{
"timer":[],
"logger":[],
"solution_exists": true,
"solution":{
"correctq": true,
"message": "Solution is correct."
}
}
```

Example 1. Naivní násobení matic – Otázky

- **Funguje výpočet správně pro libovolné velikost matic?**

- **Profilování (pro velké matice)**
  - Na kolik procent pracuje GPU?
  - Jaká byla dosažena propustnost paměti? Kolik procent z teoretického maxima to je?
  - Kolik GFLOPS jste dosáhli? Kolik % z teoretické maxima to je?
  - Čím je brzděn výpočet?
  - Jaká je dosažená aritmetická intenzita?
  - Pomohlo by změnit velikost bloku?
  - Co dalšího byste pro zvýšení výkonnosti mohli provést?

DLAŽDICOVÉ NÁSOBENÍ MATIC SE SDÍLENOU PAMĚTÍ

Example 2. SM násobení matic – Kernel

- **Otevřete soubor template.cu**
  - Doplňte kernel tak, aby provedl maticové násobení **C** = **A** × **B**
  - Matice jsou uloženy v 1D polích po řádcích
  - CUDA grid i block bude mít 2D organizaci
  - Blok v SM paměti bude mít 2D organizaci
  - **Pozor, je nutné umět vynásobit i obdélníkové matice**

```
__global__ void cudaMatrixMatrixMul(float*       C, // [N, P]
                                    const float* A, // [N, M]
                                    const float* B, // [M, P]
                                    const int    N,
                                    const int    M,
                                    const int    P)
{
  // Insert code to implement vector addition here
  // Use a 2D kernel where each thread calculates one element of the C matrix.

   // C[i][j] = sum_i( sum_j( sum_k(A[i][k] * B[k][j])));

}
```

Překlad a spuštění

CVT (bez profil)

```
make
make run{0-5}
```

Karolína (profil)

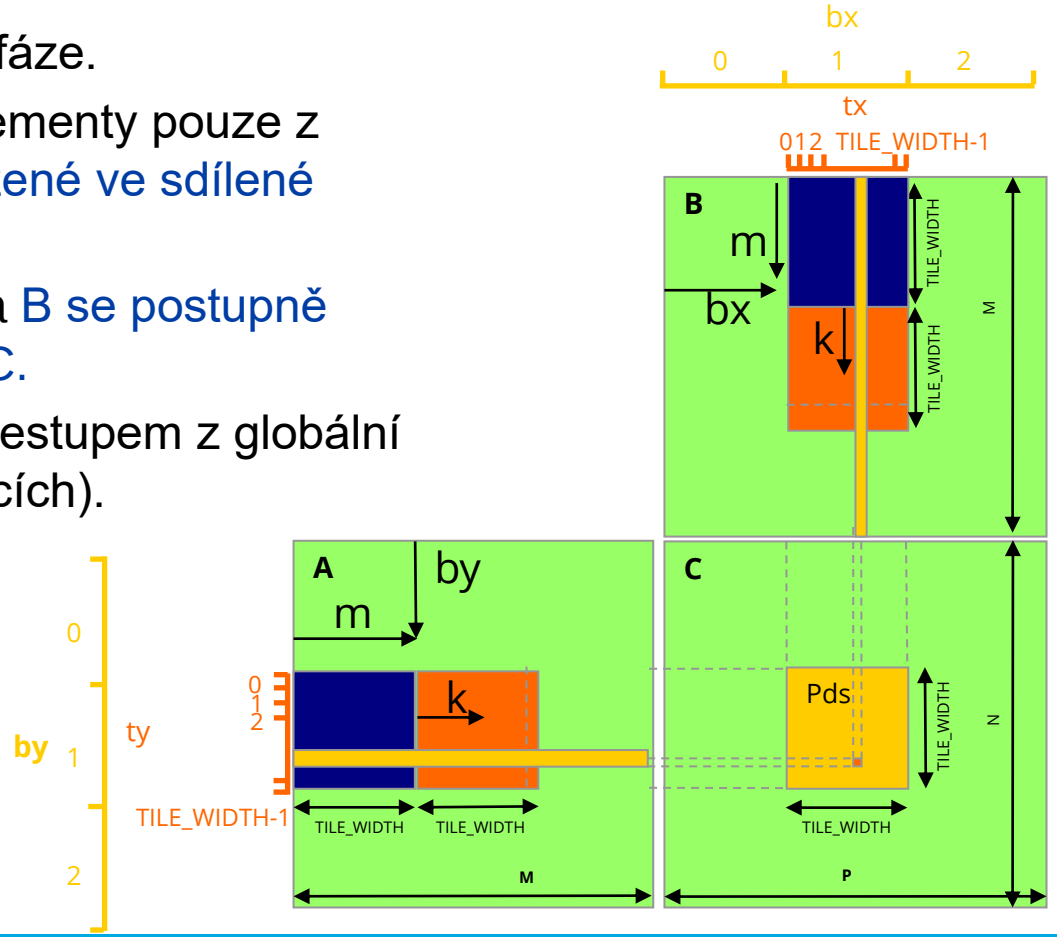
```
qsub run.pbs
```

Upravte soubor:

```
make
make prof{0-5}
```

Example 2. Princip dlaždicového násobení v SM

- Výpočet kernelu je rozdělen na fáze.
- V každé fázi se zpracovávají elementy pouze z jedné dlaždice matic A a B, uložené ve sdílené paměti.
- Částečné výsledky z dlaždic A a B se postupně akumulují ve výsledné dlaždici C.
- Řeší rovněž problém čtení s rozestupem z globální paměti (po řádcích vs. po sloupcích).
- Lze výborně použít i u CPU! (dlaždice uložené v L1 cache).



Example 2. SM násobení matic – Pomocné rutiny

- **Otevřete soubor template.cu**
  - Doplňte kód pro alokaci data na GPU
  - Doplňte kód pro kopii data na GPU
  - Nastavte velikost kernelu a spusťte kernel
  - Stáhněte data zpět na CPU

  - Vygenerujte vstupní data
- **Překlad, spuštění a profilování pro různé velikosti problému**

```
# Ladění v CVT
make
make gen
make run{0-5}
```

```
# Profiling Karolína
qsub run.pbs
```

```
make run5
./ex1 -e MatrixMultiplication/Dataset/5/output.raw -i
MatrixMultiplication/Dataset/5/input0.raw,MatrixMulti
plication/Dataset/5/input1.raw  -t matrix
-Example 1: CUDA naive matrix-matrix multiplication -
Matrix A [3072, 3072]
Matrix B [3072, 3072]
Matrix C [3072, 3072]
Time to calculate the answer:  224.4 ms
==$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
{
"timer":[],
"logger":[],
"solution_exists": true,
"solution":{
"correctq": true,
"message": "Solution is correct."
}
}
```

Example 2. SM násobení matic – Otázky

- **Funguje výpočet správně pro libovolné velikost matic?**

- **Profilování (pro velké matice)**
  - Na kolik procent pracuje GPU?
  - Jaká byla dosažena propustnost paměti? Kolik procent z teoretického maxima to je?
  - Kolik GFLOPS jste dosáhli? Kolik % z teoretické maxima to je?
  - Čím je brzděn výpočet?
  - Jaká je dosažená aritmetická intenzita?
  - Pomohlo by změnit velikost bloku?
  - Co dalšího byste pro zvýšení výkonnosti mohli provést?

- **Kolikrát je tato varianta lepší než ta předchozí?**

Vzorové řešení zveřejníme v pátek