**Semestrální projekt MI-PAP 2010/2011:**

# Paralelní řadící algoritmy

**Pavel Benáček**
**Tomáš Čejka**
**magisterské studijum, FIT ČVUT, Kolejní 550/2, 160 00 Praha 6**

**27. dubna 2011**

# Obsah



# Seznam tabulek

# 1 Definice problému a popis sekvenčního algoritmu

## 1.1 Zadání

Podle zadání semestrální práce je úkolem implementace aspoň tří z následujících řadících algoritmů: Shearsort, 3D sort, Sudo-lichý Mergesort a Bitonic sort.

Pro řešení jsme si vybrali algoritmy Shearsort, Sudo-lichý Mergesort a Bitonic sort.

Pro účely měření času běhu výsledných programů a porovnání paralelních algoritmů se sekvenčním řazením jsme si implementovali sekvenční algoritmus Mergesort a ten vzali jako referenční sekvenční řešení.

Semestrální práce spočívala v implementaci vybraných řadících algoritmů jako sekvenční řešení, paralelní řešení na počítači se sdílenou pamětí s použitím knihovny OpenMP a následně s využitím technologie CUDA společnosti NVIDIA®.

## 1.2 Řešení

Algoritmy byly ve své podstatě navrženy tak, aby co nejlépe seděli možnostem a architektuře CUDA. Bylo dbáno na několik základních vlastností. A to:

- co největší počet vláken v bloku
- co nejvíce nezávislých bloků
- minimalizace komunikace do globální paměti

Tyto části jsou společné pro všechny algoritmy pro CUDA. Zaručují jistou rychlost a škálovatelnost problému.

# 2 Popis paralelního algoritmu a jeho implementace v CUDA

## 2.1 Shearsort

Vzhledem k možnosti HW v grafickém akcelerátoru jsme zvolili implementaci pomoci even odd transpozition sortu. Toto využívání je možné zejména díky schopnosti vytvořit velký počet vláken v zařízení (a to,že každý prvek v poli má své vlastní vlákno). Při samotné implementaci bylo dbáno na několik základních pravidel. A to pravidla v úvodní části 1.1.

Po nastartování kernelu se provede nakopírování dat do sdílené paměti. Samotné řazení probíhá ve sdílené paměti a mimo bloková komunikace (výměna) probíhá přez globální paměť (kde komunikaci provádí POUZE prvky na přechodu mezi bloky).

Samotný shearsort byl implementován jako kernel, tak veškerá část algoritmu běží na GPU. Pro tuto příležitost byl implementován algoritmus synchronizace mezi bloky, který byl popsán v 2. Tento algoritmus nám umožnil synchronizaci mezi bloky a tak se nemuselo z kernelu vystupovat a provádět synchronizaci bloků v rámci programu hosta.

Program si sám spočítá rozložení matice tak, aby se do globální paměti vlezl celý sloupec 2D matice.

## 2.2 Sudo-lichý mergesort, Bitonic sort

Řešení algoritmů sudo-lichý mergesort a bitonic sort jsme nalezli v ukázkových příkladech, dodávaných společně s SDK, stažitelných ze stránek společnosti NVIDIA. Jedná se o příklad pojmenovaný **sortingNetworks**. Spustitelný program, který se v příkladu generoval, prováděl řazení pouze algoritmem Bitonic sort. Zdrojový kód v **main.cpp** jsme proto upravili tak, aby se dal spustit i sudo-lichý mergesort. Program nyní očekává při spuštění jeden parametr a to buď *bitonic* nebo *eom* a podle toho se použije řadící algoritmus. SortingNetworks jsme přesunuli z adresářové struktury příkladů tak, aby se dal zkompilovat samostatně a nezávisle na příkladech a k tomu jsme upravili soubor Makefile.

# 3 Naměřené výsledky a vyhodnocení

## 3.1 Způsob měření

Měření bylo prováděno nad daty, které se generovalo pro každý každý problém znova a které samozřejmě do doby běhu algoritmu nebylo započítáno. Měření času bylo prováděno prostředky z CUDA SDK. Pro vyhodnocení jsme zvolili konstatní počet vláken na blok.

## 3.2 Naměřené časy

### 3.2.1 Shearsort

| Množství dat ($2^n$) | Čas |
|---:|---|
| 6 | 0,00191242 |
| 7 | 0,00198106 |
| 8 | 0,00355130 |
| 9 | 0,00360128 |
| 10 | 0,00558307 |
| 11 | 0,01193860 |
| 12 | 0,04350320 |
| 13 | 0,24930300 |
| 14 | 1,28781000 |

Tabulka 1: Shearsort

### 3.2.2 Even-Odd Mergesort

| Množství dat ($2^n$) | Čas |
|---:|---:|
| 6 | 0.001378000 |
| 7 | 0.001691000 |
| 8 | 0.002085000 |
| 9 | 0.002512000 |
| 10 | 0.002978000 |
| 11 | 0.003739000 |
| 12 | 0.004578000 |
| 13 | 0.005573000 |
| 14 | 0.006700000 |

Tabulka 2: Even-Odd Mergesort

### 3.2.3 Bitonic sort

| Množství dat ($2^n$) | Čas |
|---:|---:|
| 6 | 0.001399000 |
| 7 | 0.001699000 |
| 8 | 0.002082000 |
| 9 | 0.002499000 |
| 10 | 0.002951000 |
| 11 | 0.005144000 |
| 12 | 0.007467000 |
| 13 | 0.009914000 |
| 14 | 0.012502000 |

Tabulka 3: Bitonic sort

## 3.3 Vyhodnocení

# 4 Literatura

1. Stránky předmětu MI-PAP

2. GPU synblocks