Semestrální projekt MI-PAP 2010/2011:

Paralelní řadící algoritmy

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

4. května 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.

Dále bylo potřeba odebrat kontrolu stability řazení, protože originální verzi programu nebylo možné měřit pro větší množinu dat. Z tohoto důvodu jsme museli odebrat alokaci paměti pro klíče hodnot a tím pádem i odstranit příslušné parametry volaných kernelů.

3 Naměřené výsledky

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
4	0.000200800
5	0.000282080
6	0.000556640
7	0.001100000
8	0.002022474
9	0.007240000
10	0.014284000
11	0.016643200
12	0.019943290
13	0.027145500
14	0.043032042
15	0.093929000
16	0.215102000
17	0.702910000
18	2.339670000
19	8.812000000
20	35.23100000
21	166.1040000
22	652.323000

Tabulka 1: Shearsort

3.2.2 Even-Odd Mergesort

Množství dat (2^n)	Čas
6	0.000026000
7	0.000006000
8	0.000005000
9	0.000005000
10	0.000004000
11	0.000011000
12	0.000021000
13	0.000033000
14	0.000050000
15	0.000068000
16	0.000091000
17	0.000116000
18	0.000145000
19	0.000176000
20	0.000212000
21	0.000250000
22	0.000291000
24	0.000334000
25	0.000383000
26	0.000433000
27	0.000487000
28	0.000543000

Tabulka 2: Even-odd mergesort

3.2.3 Bitonic sort

Množství dat (2^n)	Čas
6	0.000027000
7	0.000006000
8	0.000005000
9	0.000005000
10	0.000005000
11	0.000040000
12	0.000080000
13	0.000122000
14	0.000167000
15	0.000231000
16	0.000281000
17	0.000323000
18	0.000381000
19	0.000442000
20	0.000510000
21	0.000578000
22	0.000648000
24	0.000721000
25	0.000801000
26	0.000905000
27	0.001000000
28	0.001057000

Tabulka 3: Bitonic sort

4 Závěr

Po seznámení se se základním principem práce s technologií CUDA jsme začali pracovat na paralelizaci řadících algoritmů touto druhou variantou. Oproti použití technologie OpenMP vyžadovala implementace mnohem větší úsilí, takže jsme stihli realizaci algoritmu shearsort. Naštěstí jsme zbylé dva algoritmy našli v příkladech od společnosti NVIDIA, takže pro měření jsme použili mírně modifikované (co do spouštění) tyto implementace.

Technologie CUDA respektive použitá grafická karta nám umožnila využít mnohem více spuštěných vláken oproti např. OpenMP bez nezanedbatelného zvýšení režie, která s použitím vláken v OS souvisí.

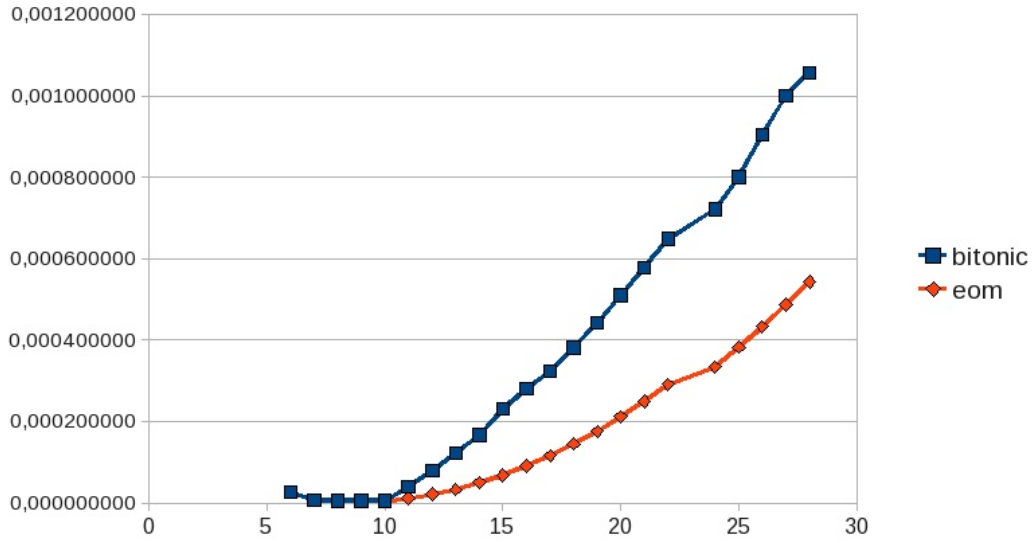
5 Literatura

1. Stránky předmětu MI-PAP
2. GPU synblocks
3. Seriál o technologii CUDA na serveru Root.cz
4. Stránky společnosti NVIDIA

A Grafy závislosti času na velikosti dat

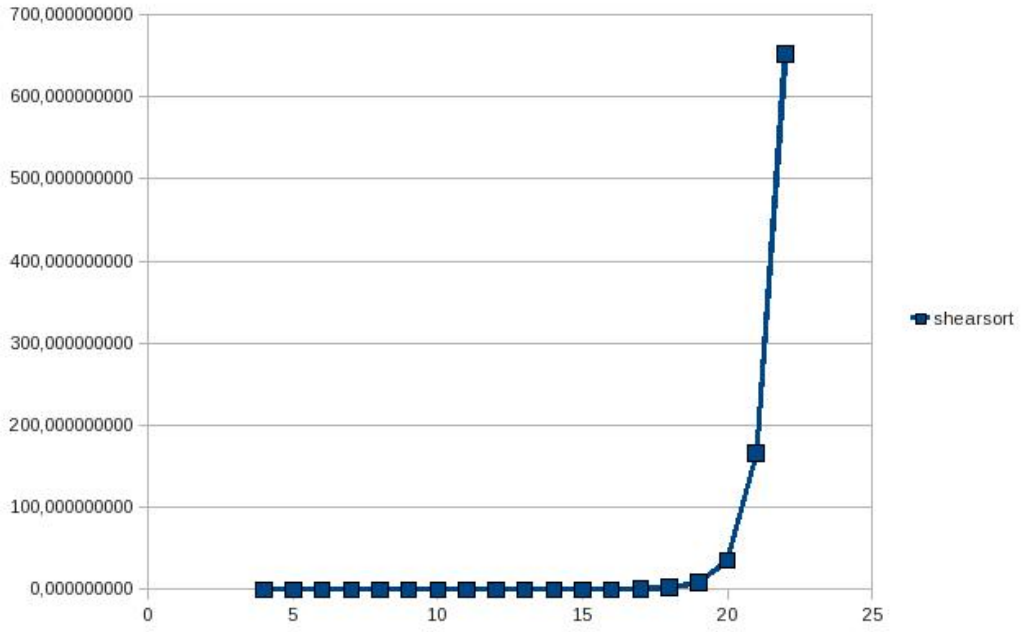
A.1 Porovnání algoritmů

A.1.1 EOMergesort, Bitonic sort



Obrázek 1: Graf závislosti času na velikosti dat EOMergesort, Bitonic sort

A.1.2 Shearsort

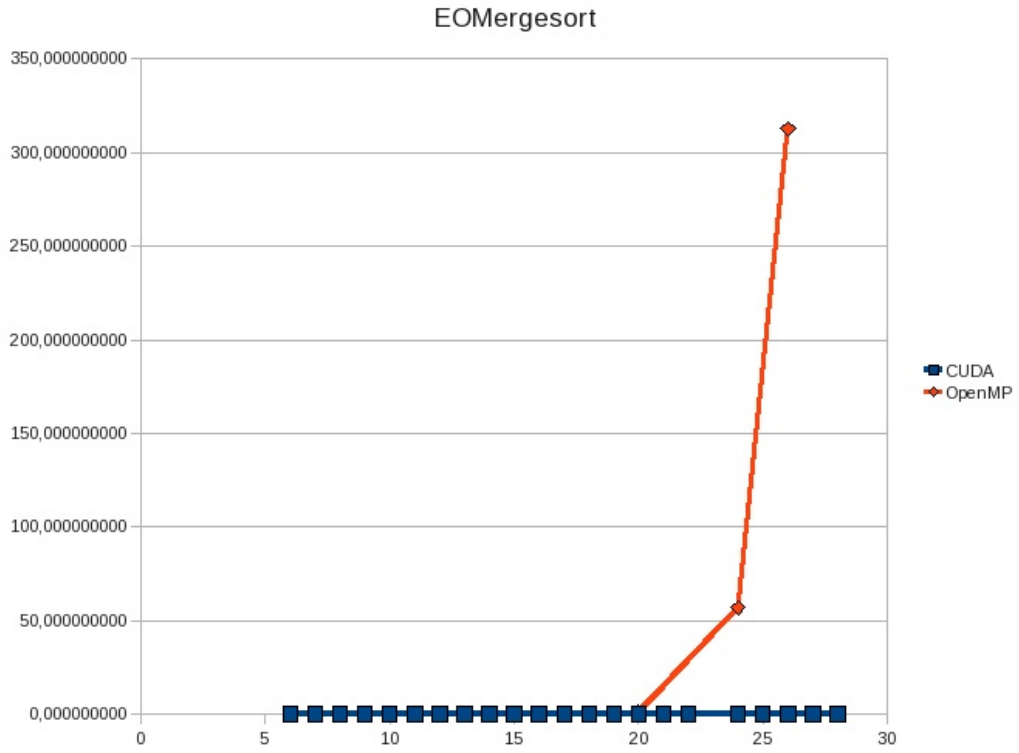


Obrázek 2: Graf závislosti času na velikosti dat Shearsort

B Porovnání technologií CUDA a OpenMP

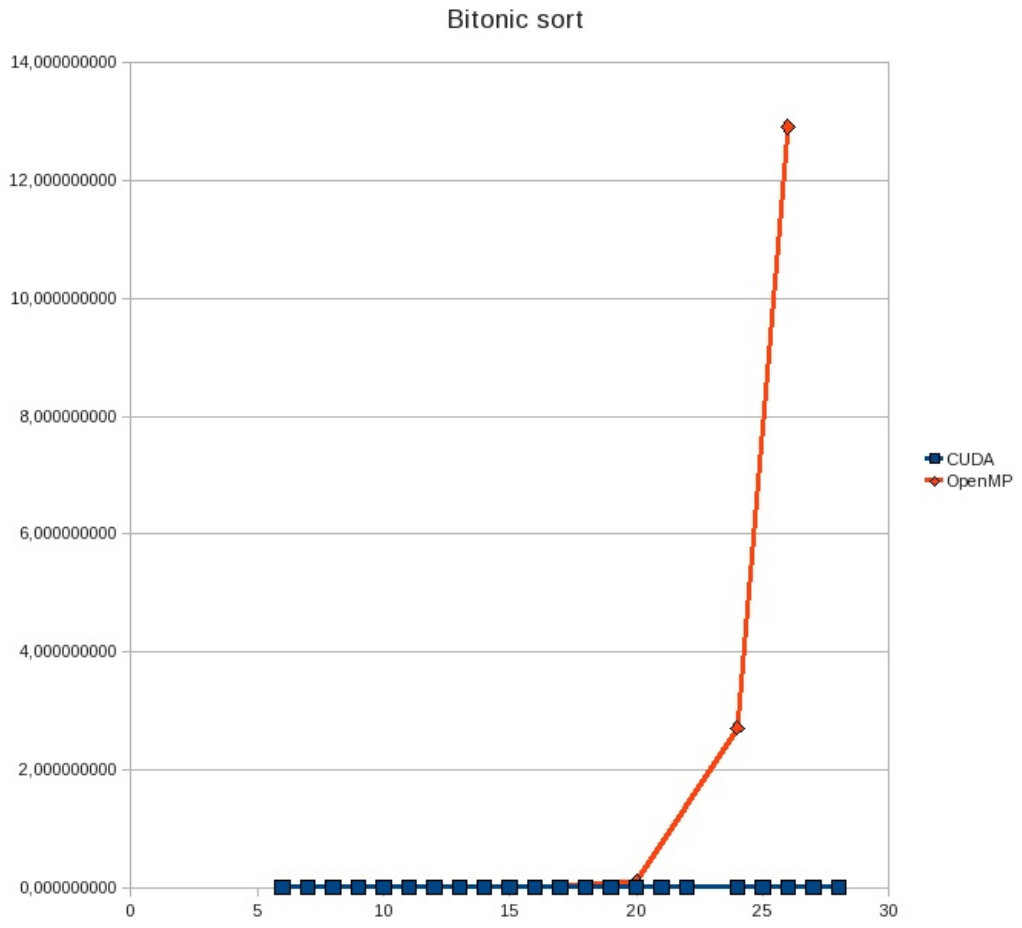
Grafy vznikly použitím hodnot z měření na grafické kartě a vybráním nejlepších hodnot při měření verze pracující s OpenMP. Vycházeli jsme z předpokladu, že se snažíme porovnat nejlepší dosažené výsledky implementací algoritmů obou technologií. Proto jsme u OpenMP vzali počty vláken podle nejlepšího naměřeného času.

B.1 Even-Odd Mergesort



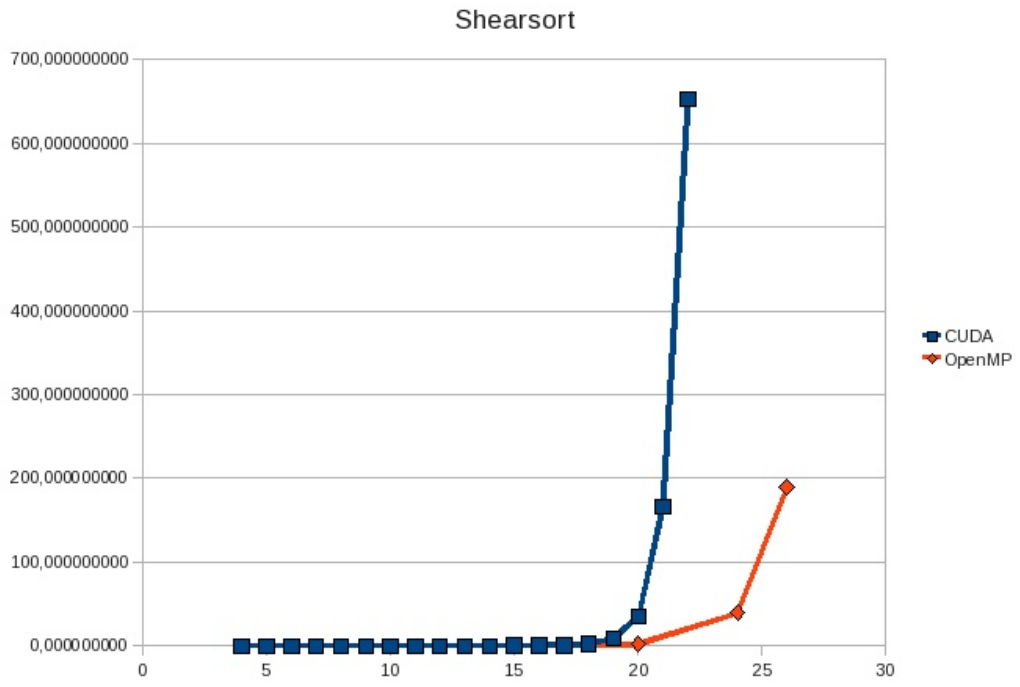
Obrázek 3: EOMergesort - CUDA vs. OpenMP

B.2 Bitonic sort



Obrázek 4: Bitonic sort - CUDA vs. OpenMP

B.3 Shearsort



Obrázek 5: Shearsort - CUDA vs. OpenMP