Semestrální projekt MI-PAP 2010/2011:

Paralelní řadící algoritmy

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

April 27, 2011

Contents

List of Tables

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

Vytvořili jsme si základní moduly **Loader** a **Array** (případně ještě pro Shearsort **Array2D**), které sdílíme mezi jednotlivými implementacemi algoritmů.

Modul Array se stará o uchování načtených dat a při načtení se stará o zvětšování alokované paměti v případě potřeby. Modifikace Array2D umožňuje mapování jednorozměrného pole na dvourozměrné.

Loader čte předané parametry programu a nastavuje na jejich základě počet prvků k seřazení a počet spouštěných vláken. Loader dokáže vygenerovat pseudonáhodnou posloupnost prvků nebo načíst data ze zadaného souboru.

Přikompilování těchto modulů nám vytváří stejné rozhraní u všech programů.

Implementací paralelních algoritmů Bitonicsort a Even-Odd mergesort vznikly i prototypy sekvenční. Tyto programy jsme rovněž zařadili do měření.

Měření jsme prováděli plně automatizovaně - v adresáři /utils jsme pro ten účel vytvořili skripty, které nám vytvoří strukturu adresářů /tests, do které podle počtu vlánek a objemu dat vygeneruje skripty pro zadání úlohy do fronty.

Nad touto strukturou adresářů je možné jednoduše provádět zařazení úloh do fronty i vybrání naměřených výsledků.

2 Popis paralelního algoritmu a jeho implementace v OpenMP

2.1 Sudo-lichý mergesort

Algoritmus Sudo-lichý mergesort je navržen na strukturu řadící sítě, kterou si můžeme představit jako systém s distribuovanou pamětí, který obsahuje množinu procesorů uspořádaných do mřížky a vhodně pospojovaných.

Algoritmus funguje tak, že na vstup systému se přivede množina dat, která se během průchodu sítí seřadí.

Řešení paralelizace s knihovou OpenMP pracuje na počítači se sdílenou pamětí, není potřeba množinu dat rozdělovat na dílčí části a distribuovat/rozkopírovat je mezi procesory. Místo toho jsou veškerá data umístěna v paměti v jednom poli a jednotlivá paralelní vlákna přistupují k disjunktním oblastem sdíleného pole.

Snažili jsme se o co nejrovnoměrnější rozdělení dat mezi vlákna, takže data virtuálně rozdělíme na začátku programu na stejné části, jejichž počet je rovný nějaké mocnině dvou a jednotlivá vlákna seřadí tuto podposloupnost dat. Tento přístup je možný díky rekurzivní povaze algoritmu a řadících sítí. Po seřazení podposloupností je potřeba ještě provést jejich sloučení. Sloučení se provádí v logaritmickém čase, protože díky půlení posloupnosti se slučuje vždy $logn$ podposloupností.

2.2 Bitonic sort

Bitonic sort vytváří v posloupnosti dat dvě podposloupnosti, kde jedna je nerostoucí a druhá neklesající. V druhé fázi se dílčí podposloupnosti slévají až jsou nakonec data seřazená.

Rozdělování podposloupností probíhá na disjuktních částech posloupnosti, což umožňuje paralelizaci. Ve smyčce, kterou procházíme pole dat se zvětšuje proměnná, určující rozestup mezi prvky.

2.3 Shearsort

Posledním implementovaným řadícím algoritmem je Shearsort. Algoritmus jsme realizovali s použitím sekvenčního algoritmu mergesort, který voláme na seřazení jednotlivých řádků a sloupců mřízky, do které jsou data virtuálně rozdělena. Virtuálně proto, že používáme pro uložení dat jednorozměrné pole a mapovací funkci, která umožňuje adresovat prvek matice souřadnicemi x a y.

Jednotlivé fáze algoritmu (sloupcové a řádkové) je nutné provést sekvenčně za sebou, ale řazení řádků i sloupců v jedné fázy je možné spouštět najednou paralelně.

3 Naměřené výsledky a vyhodnocení

3.1 Způsob měření

Měření probíhalo za pomoci knihovní funkce **omp_get_wtime()**, která vrací čas běhu programu. Uložením času na začátku výpočtu a jeho odečtením od času na konci výpočtu dostaneme dobu, kterou algoritmus potřebuje na seřazení vstupních dat.

Vlastní zařazení našich programů do fronty na serveru **star** je popsané v sekci 1.2.

3.2 Naměřené časy

3.2.1 Sekvenční Mergesort

Množství dat (2^n)	Počet vláken	Čas
10	1	0,000236067
10	1	0,000236629
10	1	0,000001956
16	1	0,022770800
16	1	0,022806500
20	1	0,449648000
20	1	0,450177000
24	1	8,545500000
24	1	8,547720000
26	1	37,555900000
26	1	37,559000000

Table 1: Mergesort

3.2.2 Sekvenční Even-Odd Mergesort

Množství dat (2^n)	Počet vláken	Čas
10	1	0,000634586
10	1	0,000634735
10	1	0,000635005
10	1	0,000001676
16	1	0,098675900
16	1	0,099481000
16	1	0,100037000
20	1	3,569340000
24	1	118,748000000
24	1	120,404000000
24	1	130,860000000
26	1	709,920000000
26	1	735,275000000

Table 2: Even-Odd Mergesort

3.2.3 Sekvenční Bitonic sort

Množství dat (2^n)	Počet vláken	Čas
10	1	0,000650510
16	1	0,091356500
20	1	2,216730000
24	1	50,019500000
26	1	233,403000000

Table 3: Bitonic sort

3.2.4 Bitonic sort

Množství dat (2^n)	Počet vláken	Čas
10	2	0,000049379
10	4	0,000057481
10	8	0,000048540
10	16	0,000058248
10	32	0,000048400
16	2	0,009313320
16	4	0,015767700
16	8	0,539144000
16	16	0,006299840
16	32	0,007108750
20	2	0,199643000
20	4	0,122328000
20	8	0,288970000
20	16	0,100444000
20	32	0,096497300
24	2	4,928480000
24	4	3,395880000
24	8	2,753720000
24	16	2,713770000
24	32	2,861780000
26	2	24,432600000
26	4	16,383700000
26	8	13,385400000
26	16	13,238900000
26	32	15,845600000

Table 4: Bitonic sort

3.2.5 Shearsort

Množství dat (2^n)	Počet vláken	Čas
10	2	0,009902510
10	4	0,003633750
10	8	0,052614600
10	16	0,002243120
10	32	0,002845790
16	2	0,376618000
16	4	0,204998000
16	8	0,251678000
16	16	0,118000000
16	32	0,130110000
20	2	9,355110000
20	4	4,719530000
20	8	2,452050000
20	16	2,305150000
20	32	2,391850000
24	2	218,866000000
24	4	109,682000000
24	8	55,209200000
24	16	42,124100000
24	32	39,520600000
26	2	1097,600000000
26	4	550,067000000
26	8	275,809000000
26	16	196,865000000
26	32	189,860000000

Table 5: Shearsort

3.2.6 Even-Odd Mergesort

Množství dat (2^n)	Počet vláken	Čas
10	1	0,000634586
10	2	0,000403688
10	4	0,000482750
10	8	0,006025150
10	16	0,000825326
10	32	0,001204360
16	1	0,098675900
16	2	0,051694300
16	4	0,052649500
16	8	0,096545900
16	16	0,063254200
16	32	0,071388800
20	1	3,569340000
20	2	1,564750000
20	4	1,431630000
20	8	1,651200000
20	16	1,848680000
20	32	2,089870000
24	1	118,748000000
24	2	59,361000000
24	4	56,932900000
24	8	66,153200000
24	16	78,251600000
24	32	86,036700000
26	1	709,920000000
26	2	335,756000000
26	4	312,814000000
26	8	352,921000000
26	16	414,455000000
26	32	480,307000000

Table 6: Even-Odd Mergesort

3.3 Zrychlení

U algoritmů Bitonic sort a Even-Odd Mergesort jsme vztahovali zrychlení k sekvenčnímu použití téhož algoritmu. Shearsort jsme porovnávali se sekvenčním algoritmem Mergesort. Pro výpočet zrychlení jsme uvažovali jen nejvyšší N, tedy množství dat, což je v našem případě 2^{26}.

Zrychlení je počítáno jako

$$S = \frac{sekvencni\_cas}{paralelni\_cas}$$

3.3.1 Bitonic sort

Vlákna	Zrychlení
1	1
2	9,55
4	14,25
8	17,44
16	17,63
32	14,73

Table 7: Zrychlení algoritmu Bitonic sort

3.3.2 Even-Odd Mergesort

Vlákna	Zrychlení
1	1
2	2,89
4	3,1
8	2,75
16	2,34
32	2,02

Table 8: Zrychlení algoritmu Bitonic sort

3.3.3 Shearsort

Vlákna	Zrychlení[1]
1	1
2	0,03
4	0,07
8	0,14
16	0,19
32	0,2

Table 9: Zrychlení algoritmu Shearsort

3.4 Vyhodnocení

Nejlepšího zrychlení jsme dosáhli u algoritmu Bitonic sort, který jsme porovnávali s jeho sekvenční variantou. Pokud bychom porovnávali sekvenční řešení problému, jednoznačně nejlepší implementací řadícího algoritmu je sekvenční Mergesort s časem[2] 37,559s na 2^{26} čísel. Následuje daleko za ním algoritmus Even-Odd Mergesort s časem 709,92s a na posledním míste je Bitonic sort s časem 940,95s.

Z těchto časů plyne i vypočtené zrychlení, které sice pro Shearsort vychází velmi špatně, ale na druhou stranu Shearsort stále zlepšuje svůj čas se vzrůstajícím počtem vláken.

4 Závěr

Seznámili jsme se s vývojem paralelních aplikací na víceprocesorovém stroji se sdílenou pamětí s využitím implementace technologie OpenMP, konkrétně pro GCC knihovna GOMP.

Zjistili jsme, že vývoj za pomoci OpenMP je paralelizace algoritmu mnohem jednodušší než při použití systémových volání pro vytváření a synchronizaci vláken. OpenMP nám umožnilo zparalelizovat smyčky minimální modifikací funkčního zdrojového kódu, což značně urychlilo implementaci řešení.

[1] Jako sekvenční algoritmus byl zvolen Mergesort

[2] Uvádíme vždy nejlepší naměřené časy

5 Literatura

1. Stránky předmětu MI-PAP

A Grafy závislosti času na počtu vláken

A.1 Měření na posloupnosti čísel o velikosti 2^{26}

A.1.1 Čas

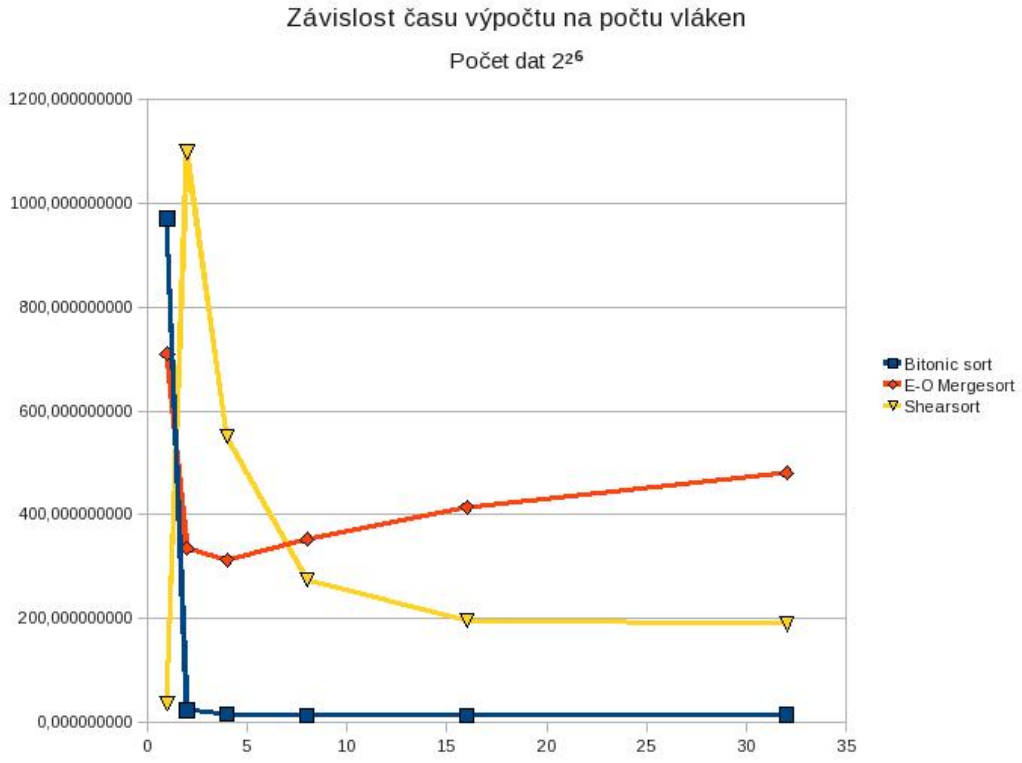

Figure 1: Graf závislosti času na počtu vláken.

A.1.2 Zrychlení

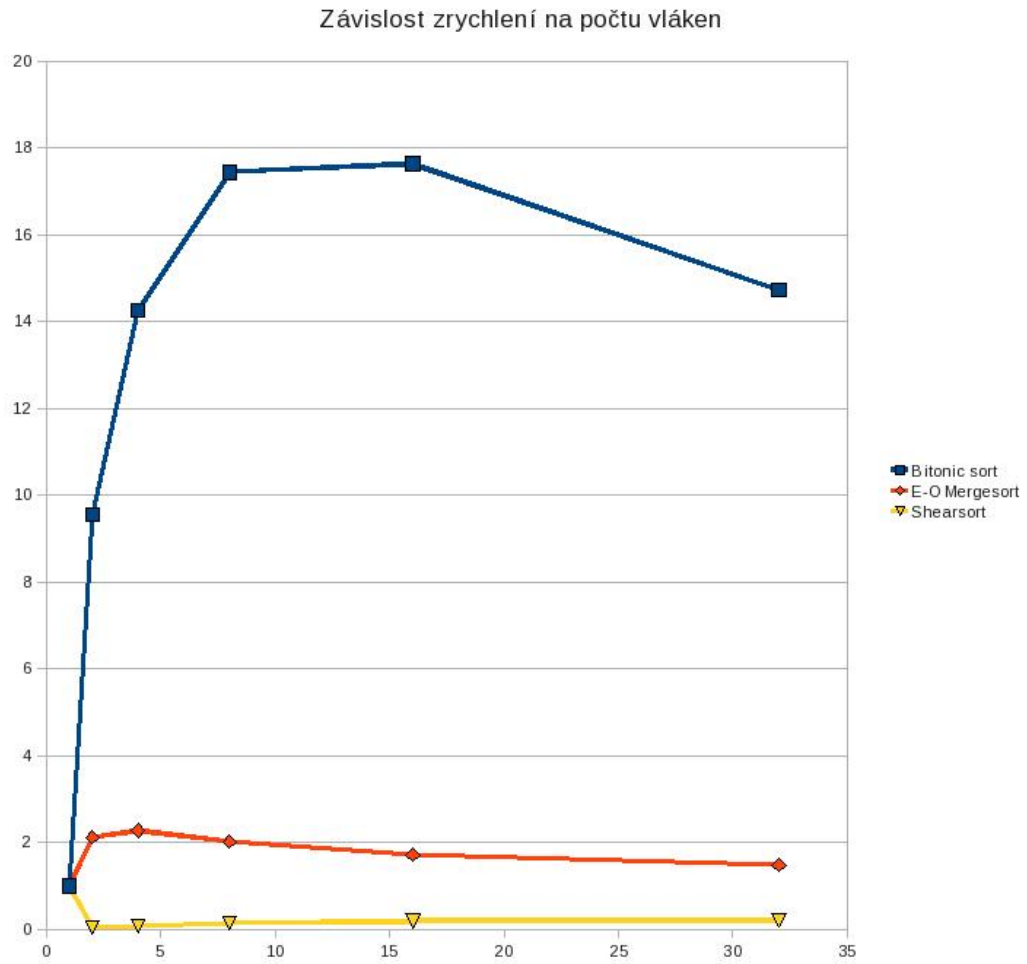

Figure 2: Zrychlení výpočtu v závisosti na počtu vláken.