FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Posudek oponenta závěrečné práce

Student:	Pavel Erazím
Oponent práce:	Ing. Daniel Langr, Ph.D.
Název práce:	Předzpracování dat pro řazení algoritmem Sample sort
Obor:	Teoretická informatika

Datum vytvoření: 24. 5. 2019

Hodnotící kritérium:	*Způsob hodnocení – následující škálou 1 až 4:*
1. Splnění zadání	*1=zadání splněno,* *2=zadání splněno s menšími výhradami,* ***3=zadání splněno s většími výhradami,*** *4=zadání nesplněno*

Popis kritéria:
Posuďte, zda předložená ZP dostatečně a v souladu se zadáním obsahově vymezuje cíle, správně je formuluje a v dostatečné kvalitě naplňuje. V komentáři uveďte body zadání, které nebyly splněny, posuďte závažnost, dopady a případně i příčiny jednotlivých nedostatků. Pokud zadání svou náročností vybočuje ze standardů pro daný typ práce nebo student případně vypracoval ZP nad rámec zadání, popište, jak se to projevilo na požadované kvalitě splnění zadání a jakým způsobem toto ovlivnilo výsledné hodnocení.

Komentář:

Práce se zabývá netriviálním tématem paralelního řazení, konkrétně paralelizace řadícího algoritmu Samplesort. Cíle práce byly většinou splněny dle zadání, velkou výhradu mám ale k bodu 5), kde byla provedena měření v sekvenční verzi algoritmu na neoptimalizovaných testovacích programech (více viz dále). Takováto měření zcela postrádají smysl.

Hodnotící kritérium:	*Způsob hodnocení – bodové hodnocení 0 až 100 bodů (známka A až F):*
2. Písemná část práce	*55 (E)*

Popis kritéria:
Zhodnoťte přiměřenost rozsahu předložené ZP vzhledem k obsahu, tj. zda všechny části ZP jsou informačně bohaté a ZP neobsahuje zbytečné části. Dále posuďte, zda předložená ZP je po věcné stránce v pořádku, případně vyskytují-li se v práci věcné chyby nebo nepřesnosti. Zhodnoťte dále logickou strukturu ZP, návaznosti jednotlivých kapitol a pochopitelnost textu pro čtenáře. Posuďte správnost používání formálních zápisů obsažených v práci. Posuďte typografickou a jazykovou stránku ZP, viz Směrnice děkana č. 26/2017, článek 3. Posuďte, zda student využil a správně citoval relevantní zdroje. Ověřte, zda jsou všechny převzaté prvky řádně odlišeny od vlastních výsledků, zda nedošlo k porušení citační etiky a zda jsou bibliografické citace úplné a v souladu s citačními zvyklostmi a normami. Zhodnoťte, zda převzatý software a jiná autorská díla, byly v ZP použity v souladu s licenčními podmínkami.

Komentář:
Práce je dobře členěná a čitelná, jednotlivé kapitoly a sekce na sebe logicky navazují, ani po formální stránce nemá práce žádné závažné nedostatky. Tak tomu ale není s obsahem, kde se jeden velmi závažný nedostatek vyskytuje. Konkrétně se jedná měření a porovnání výkonu sekvenčních řadících algoritmů při vypnutých optimalizacích překladu (sekce 5.3). Takové měření je zcela nesmyslné a nemá žádnou vypovídací hodnotu. Velmi dobře je to vidět na porovnání s výsledky sekce 5.5, kde optimalizace použity byly. Např. s optimalizacemi řadí std::sort uniformní data při n=50M za 8,24 [s]. Bez optimalizací to je podle obr. 5.6 přibližně 20 [s], tedy cca. 2,5x déle. V důsledku toho vůbec netušíme, jak je navržený algoritmu Samplsort rychlý v porovnání s algoritmy std::sort a Flashsort. Optimalizované verze se totiž mohou chovat úplně jinak než ty neoptimalizované. Tomu napovídá i následující úvaha - při zapnutých optimalizacích Samplesort s jedním vláknem seřadil uniformní data pro n=80M za 17,16 [s]. Rozdíl mezi n=50M a n=80M je podle obr. 5.6 cca. 1.66 násobný. Pokud tedy extrapolujeme čas 8,24 algoritmu std::sort pro n=50M na n=80M, dostaneme čas 8,24x1,66=13,67 [s]. Jedná se tedy o rychlejší sekvenční řazení než poskytuje Samplesort. Sekce 5.3 přitom napovídá o přesném opaku, tj. že Samplesort je rychlejší než std::sort.

Další poznámky k obsahu a formální stránce:

- Paralelní verze poskytuje na 8 jádrovém procesoru zrychlení kolem hodnoty 3. To po pravdě řečeno není moc dobrá hodnota. Efektivní paralelní implementace napr. quicksort a mergesortu dosahují zrychlení, speciálně na náhodných datech, která se blíží počtu jader (dle zkušeností si troufám tvrdit, že by to zde bylo kolem hodnoty 7). Je to dáno tím, že sice je paralelizován rekurzivní sestup Samplesortu pomocí OpenMP úloh (tasks), ale již není paralelizovaná samostatná činnost těchto úloh. Na nejvyšší úrovni rekurze tak např. veškeré úkony provádí pouze 1 vlákno a 7 jader je nevyužitých. (Toto je např. důvod proč u quicksortu efektivní implementace paralelizují nejen rekurzi, ale i partitioning.) Je ale pravda, že paralelizovat ostatní kroky (tj. defakto partitioning podle přihrádek) je úloha, která svoji složitostí přesahuje možnosti této bakalářské práce.

- GNU implementace Standardní knihovny C++ (libstdc++) dodávaná s překladači GCC obsahuje paralelní verze řadících algoritmů založené na OpenMP (tzv. libstdc++ Parallel mode). V případě std::sort se jedná o paralelní (hybridní) quicksort s paralelním rozdělováním. Pro vyhodnocení navržené paralelní verze algoritmu Samplesort by bylo vhodné ho porovnat i s paralelní verzí std::sort.

- Poněkud zavádějící je označní implementované verze algoritmu Samplesort jako in-place. In-place řadící algoritmy se označují takové, které pro svůj běh potřebují méně extra paměti než O(n), kde n je velikost řazených dat. Např. to platí o QS, který potřeuje při efektivní implementaci pouze O(log n). Uvedený Samplesort ale má extra paměťové nároky O(n), což vyplývá z textu i zdrojových kódů. Jedná se sice o n bytů, nikoliv n čísel typu int, ale i to je O(n).

- str. vii - "...Samplesort.Cílem..." - chybí mezera.
- str. vii - Abstrakt je poměrně krátký, měl by více popisovat téma i dosažené výsledky.
- str. vii - "...more effective..." -> "more efficient" - V AJ "effective" znamená, že něco má daný efekt. Effective jsou tudíž všechny řadící algoritmy, protože vedou k seřazené posloupnosti dat (požadovaný efekt). Pokud se týká vyšší effektivity ve smyslu výpočetního výkonu či lepšího využití výpočetních prostředku, používá se "efficient".
- str. 1 - "implementacia paralelizaci" - chybí mezera.
- str. 3 - Název kapitoly mi nepřijde vhodný. Cíle práce totiž popisuje jen její první sekce, druhá sekce s cíly práce vůbec nesouvisí.
- str. 4 - "...včetně ." - mezera navíc.
- str. 5 - "K pochopení Samplesortu..." - Ve formálním českém textu takovéto formulace nevypadají příliš dobře. Lepší např. "K pochopení algoritmu Samplesort" apod.
- str. 5 - "Quicksort je jednoduchý na implementaci..." - Toto platí pouze o základní (učebnicové) verzi. Praktické efektivní implementace jsou velmi složité a kombinují Quicksort s dalšími algoritmy, např. využívají insertionsort pro řazení malých částí pole a heapsort pro zamezení nejhorších případů O(n^2). Dále se používají různé optimalizace, např. tail call elimination apod. Viz např. zdrojové kódy std::sort.
- str. 9 - V algoritmu 2 chybí rekurzivní volání, defakto tento algoritmus provádí rozdělování (partitioning), nikoliv řazení.
- str. 15 a dále - "from - levý interval" - Spíše levá hranice intervalu? Podobně u "to - pravý interval".
- str. 34 - Příliš nechápu význam generování náhodných dat pomocí programu Mathematica a načítání ze souborů. Data lze snadno (a pravděpdobně i mnohem rychleji) generovat přímo v C++. Od C++11 jsou dostupná i různá rozdělení, včetně těch využitých autorem. Pokud je potřeba opakovaně pracovat se stejnými daty, lze to snadno vyřešit použitím stejné hodnoty seed pro náhodný generátor.
- sekce 5.3 - Oceňuji podrobnou analýzu hledání optimálních parametrů algoritmu Samplesort. Bohužel ale v kontextu neoptimalizovaného překladu výsledky zcela postrádají význam.
- obr 5.6 - Popisek zde se pravděpodobně váže k uniformnímu rozdělení, nikoli normálnímu.
- str. 47 - Závěr by měl více obsahovat shrnutí dosažených výsledků a jejich zobecnění než popisu obsahu jednotlivých kapitol.

Práce se zdroji je v pořádku, pouze překlep "SIMEČEK" -> "ŠIMEČEK".

Hodnotící kritérium:	*Způsob hodnocení – bodové hodnocení 0 až 100 bodů (známka A až F):*

3. Nepísemná část, přílohy

70 (C)

Popis kritéria:
Dle charakteru práce se případně vyjádřete k nepísemné části ZP. Například: SW dílo – kvalita vytvořeného programu a vhodnost a přiměřenost technologií, které byly využité od vývoje až po nasazení. HW – funkční vzorek – použité technologie a nástroje, Výzkumná a experimentální práce
– opakovatelnost experimentů

Komentář:

Zdrojové kódy jsou psané v C++ formou dvou zdrojových souborů. Trochu mi zde chybí jakékoliv logické členění. Např. bych čekal, že algoritmy budou v samostatných souborech s rozhraním (API) v hlavičkových souborech a implementací ve zdrojových (pokud se nejdná o šablony). Testovací programy by pak měly být implementované ve zdrojových souborech oddělených od implementace algoritmů. Jinak se zdá být kód v pořádku a obsahuje i určité množství komentářů. Ty by ale měly více popisovat API funkcí (funkcionalitu, význam parametrů, návratové hodnoty, výjimky, atd.). Jedna přípomínka - "#include <bits/stdc++.h>" - opradu ne, nejdná se o hlavičkový soubor jazyka C++ a díky tomu je kód nepřenositelný.

Hodnotící kritérium: | *Způsob hodnocení – bodové hodnocení 0 až 100 bodů (známka A až F):*

4. Hodnocení výsledků, jejich využitelnost

55 (E)

Popis kritéria:
Dle charakteru práce zhodnoťte možnosti nasazení výsledků práce v praxi nebo uveďte, zda výsledky ZP rozšiřují již publikované známé výsledky nebo přinášející zcela nové poznatky.

Komentář:

Přínosem práce jsou určitě navržené varianty algoritmu Samplesort, především sekvenční verze. Paralelní verze mi zatím v praxi nepřijde příliž užitečná z důvodů nízké škálovatelnosti zapříčiněné neparalelizací vlastního rozdělování dat do přihrádek. Tomu odpovídá i nízké dosažené paralelní zrychlení (cca. 3 na 8 jádrech). Paralelizace rozdělování dat v rámci algoritmu Samplesort tak tvoří zajímavou úlohu pro budoucí řešení jako návaznost na tuto práci. Ohledně měření - jak již bylo řečno, zde jsou dosažené výsledky úplně nepoutitelné. Je to velká škoda, protože přidáním jednoho přepínače k překladu programů by bylo vše jinak.

Hodnotící kritérium: | *Způsob hodnocení – nehodnotí se*

5. Otázky k obhajobě

Popis kritéria:
Uveďte případné dotazy, které by měl student zodpovědět při obhajobě ZP před komisí (body oddělte odrážkami).

Otázky:

1) Mnohé řadící algoritmy, jako je např. quicksort, lze implementovat pouze pomocí dvou operací s daty, a to konkrétně porovnání dvou prvků a jejcih prohození (compare & swap). Jak je to u navrženého algoritmu Samplesort? Které operace s daty jsou potřeba? Lze algoritmus/implementaci zobecnit na libovolné typy dat (v práci jsou fixně řazena jen celá čísla).

2) Jak složitá by byla paralelizace rozdělování dat do přihrádek? Řeší tuto úlohu nějaké existující algoritmy a implementace? (Např. u quicksortu je paralelní partitioninig nejsložitější částí paralelizace celého řazení.)

3) Z jakého důvodu si autor vybral Flashsort pro porovnávání?

4) Proč byla testovací data generována pomocí programu Mathematica a ne přimo v C++?

Hodnotící kritérium: | *Způsob hodnocení – bodové hodnocení 0 až 100 bodů (známka A až F):*

6. Celkové hodnocení

60 (D)

Popis kritéria:
Shrňte stránky ZP, které nejvíce ovlivnily Vaše celkové hodnocení. Celkové hodnocení nemusí být aritmetickým průměrem či jinou hodnotou vypočtenou z hodnocení v předchozích jednotlivých kritériích. Obecně platí, že bezvadně splněné zadání je hodnoceno klasifikačním stupněm A.

Text hodnocení:

Celkově měla tato práce i její vypracování autorem poměrně velký potenciál, který byl však degradován jednou naprosto klíčovou chybou. Navržené algoritmy a jejich implementace se však zdají být použitelné pro další výzkum v oblasti paralelních řadících algoritmů.

Podpis oponenta práce: