Semestrální projekt MI-PAR 2010/2011:

Paralelní algoritmus pro řešení problému: Minimálně lomená souvislá čára

Tomáš Čejka
Martin Lukáš
magisterské studijum, FIT ČVUT, Kolejní 550/2, 160 00 Praha 6

9. prosince 2010

1 Definice problému a popis sekvenčního algoritmu

Práce se zabývá algoritmem pro konstrukci souvislé lomené čáry skládající se z posloupnosti navazujících úseček, která spojí všech k bodů a bude obsahovat minimum zlomových bodů. Sekvenční algoritmus je typu BB-DFS s hloubkou prohledávaného prostoru omezenou na k-2. Řešení vždy existuje.

Vstupní a výstupní jsou téměř stejného formátu. Body jsou v souboru reprezentovány dvěmi celými čísly na řádce. U vstupního souboru je navíc na prvním řádku uveden celkový počet čtených bodů, na pořadí bodů nezáleží. U výstupního souboru na pořadí bodů záleží, jejich pořadí udává posloupnost bodů z nichž je konstruován graf lomené čáry.

Pro reprezentaci bodu načteného ze souboru byla vytvořena třída Point. Objekty této třídy jsou uloženy v globálně přístupném dynamicky vytvořeném poli, jehož velikost je rovněž zachycena v globální proměnné.

Při rekurzivním procházení stavového prostoru byl využit implicitní zásobník pro ukládání zpracovávaných stavů. Tento zásobník je tvořen datovou strukturou std::list ze standartní C++ knihovny. Stavy jsou reprezentovány třídou State, jež uchovává pole celočíselných indexů používaných k adresaci v poli načtených bodů. Tato třída také obsahuje operace např. pro expandování stavu na zásobník.

Expanze je metoda stavu, která vytváří následníky stavu přidáním doposud nepoužitého bodu. Tímto postupem lze rekurzivně generovat kompletní stavový prostor, všechny permutace pořádí bodů.

Expanduje se vždy stav z vrcholu zásobníku. Při expanzi stavu se počítá cena, nebo-li počet zlomů vytvořených spojením posloupnosti bodů na než

ukazují indexy stavu. Tato cena je taktéž součástí stavu, tedy není nutné opakovaně počítat počet zlomů celé čáry, ale pouze vždy při expanzi rozhodnout o inkrementaci ceny.

Pokud stav nelze již expandovat, je porovnána globální minimální cena s nově vypočtenou hodnotou a případně uchováno řešení s aktualizací globálního minima. Během odebírání stavů ze zásobníku je porovnána cena odebraného stavu s globálním minimem, pak je stav dále expandován nebo je smazán.

Pořadí procházení stavů není nijak měněno, tj. stavy nejsou na zásobníku seřazeny dle ceny, i když by to bylo zřejmně vhodné.

Po vyprázdnění implicitního zásobníku je proveden výpis řešení a program je ukončen.

2 Popis paralelního algoritmu a jeho implementace v MPI

Po implementaci sekvenčního algoritmu jsme přistoupili k jeho paralelizaci. Paralelizace v našem případě spočívá v sestavení konečného automatu pro distribuci stavů a mezivýpočtů mezi zásobníky výpočetních uzlů. Je tedy použit totožný sekvenční algoritmus, který je spouštěn paralelně.

Zprávy které jsou mezi uzly zasílány mají nastaven příznak dle něhož je určen stav přijímacího automatu. Jsou to především tyto příznaky:

- MSG_DIST_DATA - příznak šíření počátečních dat, broadcast
- MSG_WORK_REQUEST - příznak žádosti o práci, stavy
- MSG_WORK_TRANSFER - příznak příchozí práce, stavů
- MSG_NO_WORK - příznak odpovědi žádné dostupné práce
- MSG_BEST_GLOBAL_SOLUTION - příznak zprávy obsahující nejlepší globální řešení

Paralelní algoritmus pracuje takto. Výpočetní uzly se nejprve inicializují funkcí MPI_Init z knihovny MPI. Každému uzlu je přiřazeno unikátní pořadové číslo. Uzel s pořadovým číslem 0 načte data ze souboru a vyšle je hromadným vysíláním(MPI_Broadcast) ostatním uzlům. Po odeslání těchto uzlů začne svůj lokální sekvenční výpočet, tj. umístí na svůj zásobník první

stavy. Ostatní uzly se cyklicky v pořadí sekvenčních čísel uzlů dotazují na práci. Pokud dotazovaný uzel nemá žádný stav na zásobníku, který by mohl zaslat, dotazuje se v pořadí dalšího uzlu. V případě, že uzel může nabídnout stavy, vybere je a zašle je žadateli o práci.

Výběr stavů ze zásobníku, které budou odeslány pokud je o ně zažádáno, je realizován algoritmem D-ADZ (půlení u dna). Žadatel obdrží stavy ze zásobníku s jednou polovinou stavu co nejblíže dnu. Tento algortimus bylo snadné realizovat, jelikož virtuální zásobník je řešen strukturou spojového seznamu.

Aby nedocházelo ke zbytečnému výpočtu cenově horších stavů než některé z nalezených na jiném uzlu, je implementováno tzv. šíření nejlepších výsledků. Pokud uzel nalezne lepší řešení než znají všechny ostatní uzly, rozešle těmto uzlům celočíselnou hodnotu pomocí neblokujícího MPI_ISend ve zprávě s příznakem MSG_BEST_GLOBAL_SOLUTION. Uzly si tuto globálně nejlepší cenu uloží a cenově horší stavy již neexpandují. Paralelní algoritmus je typu G-PBB-DFS-D.

Pokud uzel jež žádá o práci ji nedostane od žádného z uzlů, pak postupuje dle algoritmu pro distribuované ukončení paralelního výpočtu (ADUV). Reakce na příchod WHITE/BLACK tokenu je součástí stavového automatu pro komunikaci. Pokud je výpočet ukončen, tj. všechny uzly obdrží zprávu s příznakem MSG_FINISHING, je nutné přenést nejlepší známé řešení na uzel 0. Toto je řešeno použitím funkce MPI_Barrier, jež slouží pro synchronizaci všech uzlů. Po této synchronizaci je rozesláno hromadným vysíláním cena nejlepšího řešení. Uzel jež toto řešení vypočítal jej zašle uzlu 0, který ho zapíše do souboru a výpíše na obrazovku.

3 Naměřené výsledky a vyhodnocení

3.1 Způsob měření

Měření probíhalo za použitím knihovních funkcí OpenMPI. Konkrétně se jednalo o funkce MPI_Barrier(), která nám zaručila synchronizaci všech procesorů, a funkci MPI_Wtime(), která vrací čas v sekundách. Čas výpočtu jsme změřili jako rozdíl časů t_2 a t_1, kde t_1 je čas před započetím výpočtu tesně za inicializací knihovny OpenMPI a t_2 je čas na konci výpočtu před ukončením práce s knihovou.

Část kódu po inicializaci:

```
MPI_Barrier(MPI_COMM_WORLD);
t1 = MPI_Wtime();
```

Část kódu před ukončením:

```
MPI_Barrier(MPI_COMM_WORLD);
t2 = MPI_Wtime();
if (cpu_id == CPU_MASTER) {
     std::cout << "Spotrebovany cas: " << t2 - t1 << std::endl;
}
```

Měřili jsme na třech různých instancích problému se 16-ti body v rovině. Tyto tři instance jsme pojmenovali *Test A*, *Test B* a *Test C*. Naměřili jsme dobu sekvenčního výpočtu a následně paralelního výpočtu se stejným algoritmem výpočtu. Pro všechny paralelní testy jsme měřili úlohu na různém počtu procesorů, konkrétně pro 2, 4, 8, 16, 24 a 32 procesorů. Všechna měření jsme zopakovali při spuštění na propojovací síti InfiniBand a Ethernet.

Naměřené hodnoty jsme zanesli do tabulek rozdělených podle typu propojovací sítě. Nakonec jsme spočítali zrychlení oproti sekvenčnímu řešení a výsledky opět zanesli do tabulek.

Pomocí OpenOffice.org jsme nechali vykreslit grafy znázorňující závislost času výpočtu na počtu procesorů a závislost zrychlení na počtu procesorů. Všechny grafy jsou v příloze.

3.2 Naměřené časy

Počet CPU	A	B	C
1	321,8	307,3	338,6
2	39,46390	36,33120	40,27212
4	13,18690	12,84250	21,87080
8	7,81430	10,90410	15,13350
16	7,51740	8,99971	11,43290
24	37,14640	7,44230	9,58400
32	6,92410	7,53790	9,39125

Tabulka 1: Propojovací síť Ethernet

Počet CPU	A	B	C
1	321,8	307,3	338,6
2	38,95370	35,46150	40,27100
4	12,20080	12,45520	21,30230
8	7,14178	10,43290	15,05280
16	6,55517	8,99971	11,43290
24	6,21340	7,14230	8,85400
32	6,24150	7,13790	8,42100

Tabulka 2: Propojovací síť InfiniBand

3.3 Spočítané zrychlení

Počet CPU	A	B	C
1	1	1	1
2	8,15	8,78	8,99
4	24,40	24,84	16,55
8	41,18	29,26	23,92
16	42,81	35,45	31,66
24	45,03	42,86	37,77
32	46,48	42,32	38,55

Tabulka 3: Zrychlení Propojovací síť Ethernet

Počet CPU	A	B	C
1	1	1	1
2	8,26	9,00	8,99
4	26,38	25,61	16,99
8	45,06	30,58	24,05
16	49,09	35,45	31,66
24	51,79	44,66	40,89
32	51,56	44,69	42,99

Tabulka 4: Zrychlení Propojovací síť InfiniBand

Zrychlení nám vyšlo superlineární pro všechna měření paralelní úlohy, protože platí

$$S > p$$

, kde S je zrychlení a p je počet procesorů. Důvodem takového zrychlení je vhodné rozesílání nejlepších dosažených výsledků a tím i četné ořezávání stavového prostoru.

3.4 Vyhodnocení

Každý procesor má vlastní čítač, který inkrementuje pokaždé, když žádá nějaký jiný procesor o práci. Tím je zajištěné, že procesory budou žádostmi zatěžovány rovnoměrně. Při rozdělování zásobníku a následném odeslání práce se dbá na to, aby odesílané stavy nebyly příliš rozpracované, protože z hlediska zpoždění při komunikaci je rychlejší, když procesor stav dopočítá sám, než aby tento stav poslal pro expandování jinému procesoru. Kdyby se totiž posílaly stavy blízké listům stavového prostoru, procesory, které práci dostávají, by žádali o práci mnohem častěji.

Během implementace a testování jsme empiricky zjistili, že je výhodné, když se zásobník nedělí přesně na polovinu, ale odesílá se část, která je o něco menší než polovina. Tím jsme docílili toho, že nultému procesoru by měla dojít práce jako poslednímu, což je důležité z hlediska ukončovacího algoritmu, který je startován vždy od nultého procesoru. Stav, kdy nultému procesoru dojde práce dříve než ostatním je nežádoucí, protože nultý procesor začne posílat pešky a čekat na ukončení práce ostatních procesorů. Už nežádá další práci.

Protože jsme neměli z časových důvodů (čekání ve frontě na clusteru) a kvůli omezeným prostředkům možnost změřit dobu zpracování úlohy pro 16 bodů na více než 32 procesorech, nejsme schopni přesně určit, pro kolik procesorů převáží komunikace natolik, že již nebude výhodné počet procesorů zvyšovat. Z grafu zrychlení však můžeme pozorovat snižující se tendenci růstu a z toho usuzujeme, že pro 16 bodů mez, za kterou již není účinné problém rozkládat, leží kolem 40 procesorů.

4 Závěr

Měřením jsme ověřili existenci zrychlení při použití paralelizace sekvenčního řešení problému, které závisí na počtu použitých procesorů. Při implemen-

taci jsme se seznámili se základními funkcemi a vlastnostmi knihovny pro synchronizaci paralelních algoritmů pracující na bázi tzv Message Passing. Konkrétně se jednalo se o velmi rozšířenou knihovnu OpenMPI.

Vzhledem ke vzrůstajícímu významu paralelizace a distribuovaných výpočtů v oblasti informačních technologií pro nás bylo seznámení se s paralelním programováním velmi důležité a práce s OpenMPI přínosná.

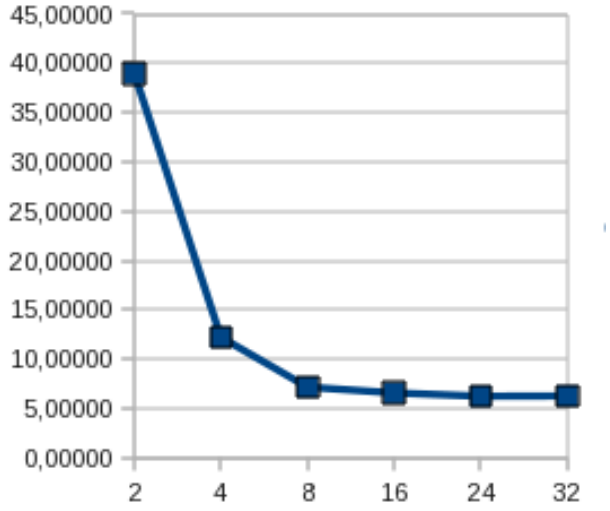
5 Literatura

1. Stránky cvičení - prohledávání do hloubky
2. Stránky cvičení - poznámky k implementaci

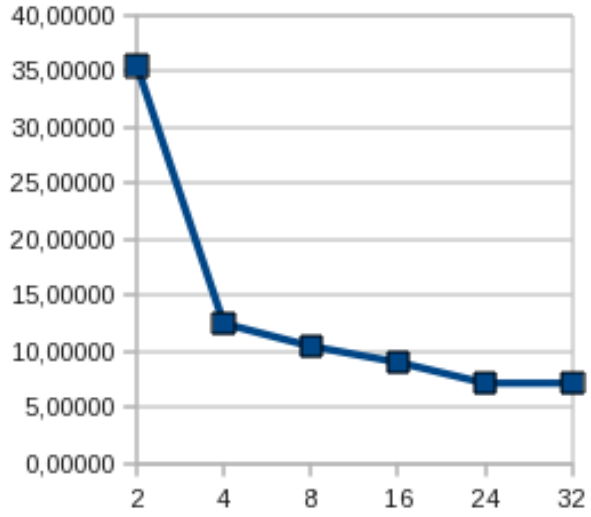
A Grafy závislosti času na počtu procesorů

A.1 Propojovací síť InfiniBand

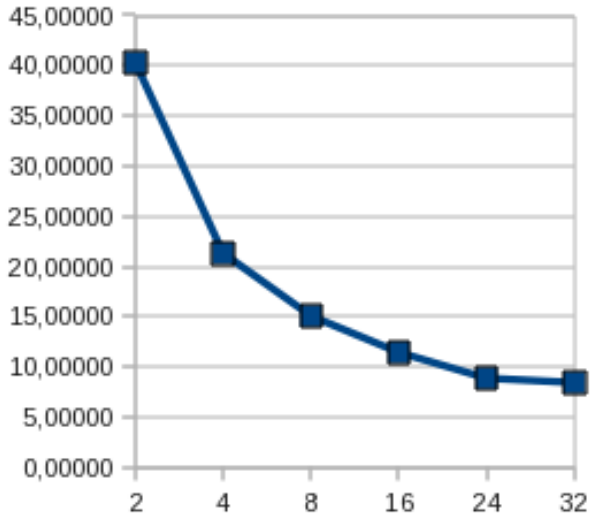
Grafy závislosti času na počtu procesorů při použití propojovací sítě InfiniBand.



Obrázek 1: Test A



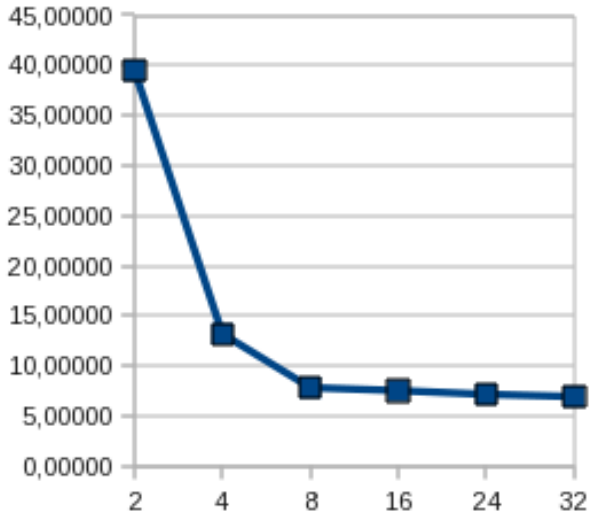
Obrázek 2: Test B



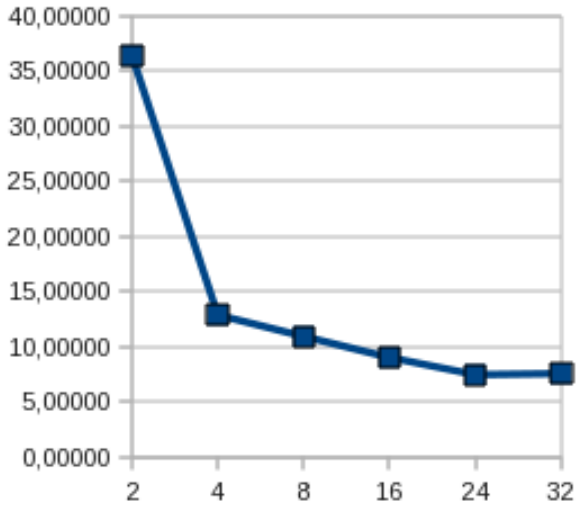
Obrázek 3: Test C

A.2 Propojovací síť Ethernet

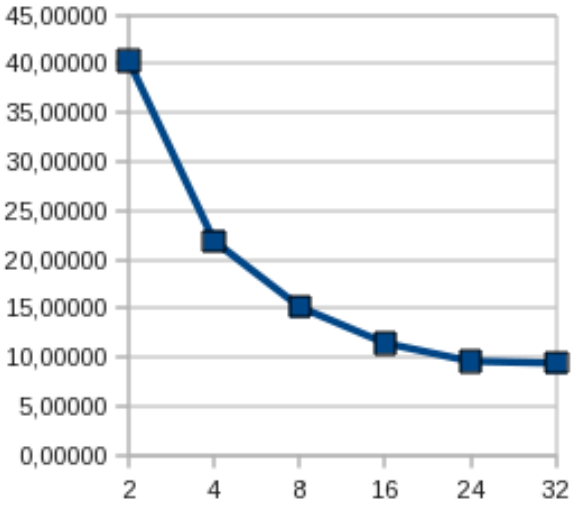
Grafy závislosti času na počtu procesorů při použití propojovací sítě Ethernet.



Obrázek 4: Test A



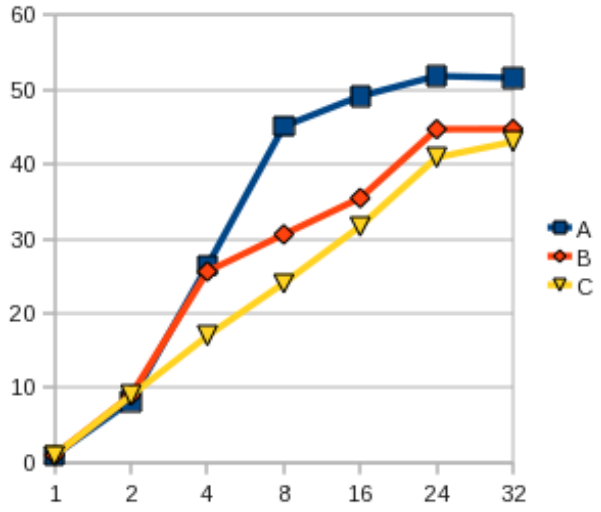
Obrázek 5: Test B



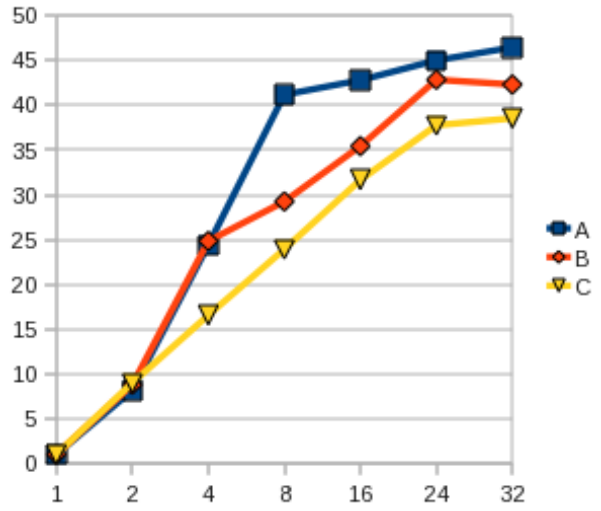
Obrázek 6: Test C

B Grafy zrychlení

Grafy zobrazující závislost zrychlení výpočtu na počtu procesorů.



Obrázek 7: InfiniBand



Obrázek 8: Ethernet