Semestrální projekt PAR 2006/2007:

MBK: Minimálně barevná kostra grafu

Pavel Dědourek
Vojtěch Hora
5. ročník, obor počítače, K336 FEL ČVUT, Karlovo nám. 13, 121 35 Praha 2

10. ledna 2007

1 Definice problému a popis sekvenčního algoritmu

1.1 Definice problému

1.1.1 Zadání

Nalezněte minimálně barevnou kostru K grafu G, tj. kostru s minimálním počtem barev hran.

1.1.2 Definice

V grafu G je každá jeho hrana ohodnocena přirozeným číslem z množiny B = {1, ..., b} představující její barvu. Kostra grafu G je strom (tj. souvislý acyklický podgraf) obsahující všechny uzly grafu G.

1.1.3 Vstupní data

n = přirozené číslo představující počet uzlů grafu, n $\geq$ 5
G = (V, E) = jednoduchý neorientovaný souvislý graf o n uzlech s ohodnocenými hranami
b = přirozené číslo, b $\geq$ 3

Soubor obsahující vstupní data je následujícího formátu:

Na prvním řádku je počet uzlů, následující řádky představují matici sousednosti (tj. počet řádků i sloupců je stejný s počtem uzlů, pokud mezi uzly existuje hrana, tak je v daném řádku a uzlu jednička).

Program se spustí takto: *mbk <počet barev> <vstupní soubor>*

2 Popis sekvenčního algoritmu

Rozhodli jsme se použít algoritmus **Generování všech koster grafu** z 5. kapitoly (Minimální kostry a stromy) skripta Teoretická informatika [2]. Pouze bylo potřeba daný algoritmus převést z rekurzivní verze na řešení inkrementální se zásobníkem s omezenou hloubkou na $n - 1$.

Algoritmus tedy funguje následujícím způsobem. Na zásobník se v hlavním cyklu vloží vždy dvě částečná řešení. První obsahující cestu s danou hranou a druhé řešení, které naopak danou hranu neobsahuje. U obou řešení se poznačí, že hrana byla z grafu vybrána. Při výběru hran se kontroluje pomocí DFS algoritmu, zda-li nevznikla kružnice (detekce zpětných hran). Pokud by vznikla, tak by se řešení nevkládalo na zásobník. Při této operaci se počítá obarvení dané

kostry. Při vkládání na zásobník se kontroluje, zda již nebyla nalezena kostra s nižší barevností než právě vkládané řešení. Algoritmus končí, pokud je zásobník prázdný.

3 Popis paralelního algoritmu a jeho implementace v MPI

Paralelní algorimtus začíná tak, že si každý procesor načte vstupní datové struktury, které jsou pro všechny procesory stejné. Pouze první procesor vloží na zásobník první, prázdnou cestu, ostatní nemají práci. Pokud procesor nemá práci, žádá o ni postupně všechny ostatní procesory. Pokud tázaný procesor má práci, předá ji žadateli (odebere nejhlubší prvek ze svého zásobníku a žadatel si ho vloží na svůj zásobník), jinak ohlásí že nemá práci. Když žadatel o práci dostane zprávu „nemám práci“, zeptá se dalšího procesoru. Pokud tuto zprávu dostane od všech procesorů, znamená to, že žádný procesor nemá práci, takže rozběhne proces ukončování.

Procesor, který začíná proces ukončování, odešle sousednímu procesoru zprávu „budem končit“, a ten ji pošle dalšímu. Po obdržení této zprávy procesor nesmí žádat o práci, a na požadavky o ni odpovídá, že ji nemá. Když se zpráva dostane zpět k prvnímu procesoru, znamená to, že již všechny procesory vědí o ukončování, a že již žádný procesor nerozesílá požadavky o práci (a nečeká na odpovědi). V tuto chvíli procesor odešle zprávu „končíme“. Každý procesor, který ji obdrží, ji pošle dál a skončí.

Když během výpočtu procesor najde lepší řešení (kostru o menším počtu barev), než bylo zatím nalezené, tak o tom pošle informaci ostatním procesorům, aby nehledali řešení, které je stejné nebo horší než toto řešení.

4 Naměřené výsledky a vyhodnocení

4.1 Měření na sekvenčním algoritmu

Měření bylo prováděno pro grafy s maximálním obarvením 5 barev.

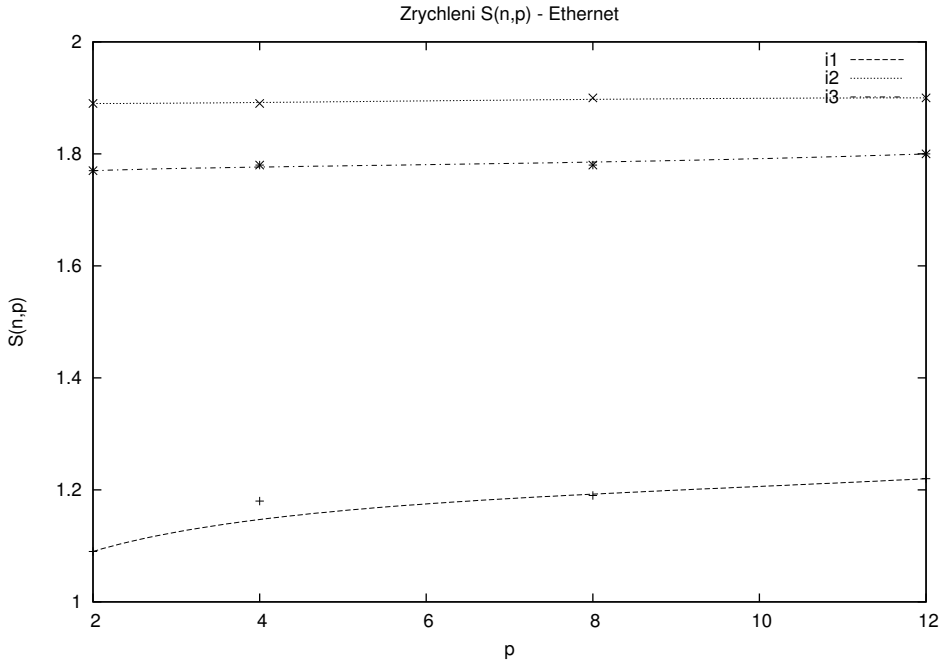
instance	i1	i2	i3
počet uzlů	20	23	22
počet hran	28	26	33
čas výpočtu [sec]	364	688	1298

4.2 Měření na paralelním algoritmu

4.2.1 Ethernet

T (n,p) [sec]	Počet procesorů (p)			
	2	4	8	12
i1	333,88	308,60	304,68	297,75
i2	364,38	363,00	362,25	361,31
i3	733,42	729,71	729,05	722,05

S (n,p)	Počet procesorů (p)			
	2	4	8	12
i1	1,09	1,18	1,19	1,22
i2	1,89	1,89	1,90	1,90
i3	1,77	1,78	1,78	1,80

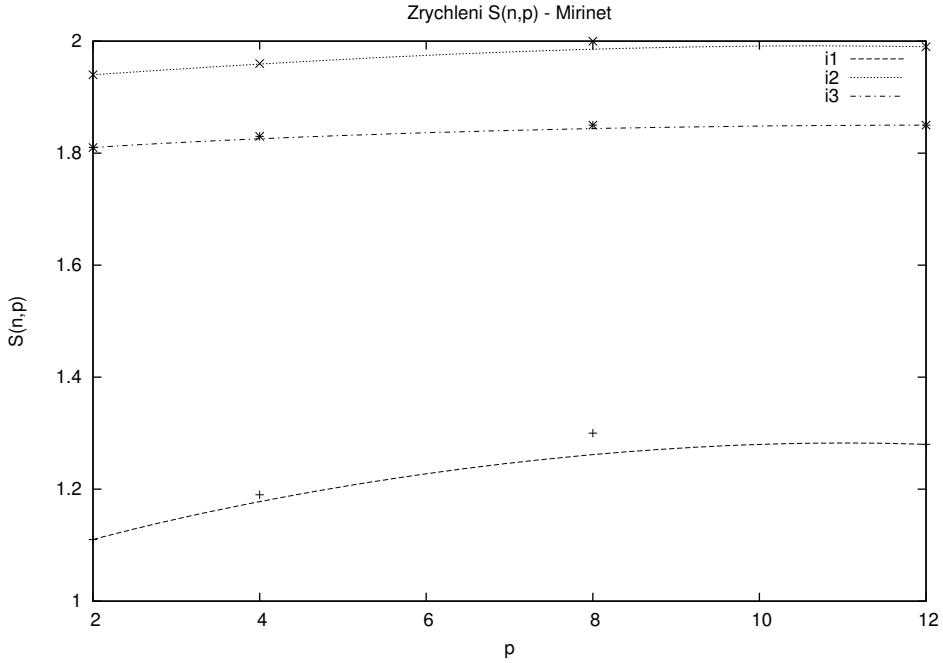


Obrázek 1: Zrychlení na síti Ethernet

4.2.2 Myrinet

T (n,p) [sec]	Počet procesorů (p)			
	2	4	8	12
i1	328,35	304,57	279,69	283,54
i2	355,14	350,04	343,78	345,57
i3	717,7	707,93	700,20	699,96

S (n,p)	Počet procesorů (p)			
	2	4	8	12
i1	1,11	1,19	1,30	1,28
i2	1,94	1,96	2,00	1,99
i3	1,81	1,83	1,85	1,85



Obrázek 2: Zrychlení na síti Myrinet

4.3 Superlineární zrychlení

Na obrázcích (1, 2) je vidět, že ani v jednom případě nedošlo k superlineárnímu zrychlení. Je to nejspíše způsobeno špatným rozdělením práce mezi procesory. Na zásobníku totiž vždy vzniknou jen dva nové stavy (bez hrany a s hranou). Proto procesory čekající na práci mohou být obslouženy postupně za sebou. Tedy až poté co nový procesor co dostal práci vytvoří další stavový prostor.

4.4 Komunikační složitost

4.4.1 Ethernet

Počet zpráv	Počet procesorů (p)			
	2	4	8	12
i1	5,5	18212	77947	85964
i2	5,5	26308	45291	50419
i3	6	57321	100977	116989

4.4.2 Myrinet

Počet zpráv	Počet procesorů (p)			
	2	4	8	12
i1	5,5	20998	79748	87794
i2	5,5	29036	48211	51684
i3	6	62262	108344	123005

Uvedená tabulka poskytuje údaje o množství poslaných zpráv mezi procesory. Je z ní patrné, že počet zpráv extrémně rychle roste v závislosti na počtu procesorů, přestože algoritmus se ptá na došlé zprávy v každé 100. iteraci. Z toho plyne, že algoritmus pro přidělování práce se zdá býti značně neefektivním. Bylo by nutné ho pozměnit, aby nedocházelo k posílání takového počtu zpráv.

Náš algoritmus není vůbec efektivní, neb na dvou procesorech trvá výpočet stejně dlouho jako na více procesorech. Algoritmus ze stejných důvodů ani neprojevuje známky dobré škálovatelnosti. Tj. při změně velikosti vstupního souboru nám stejně nepomůže zvýšit množství procesorů, protože nesejde na tom, zda daný problém počítají dva procesory, či více.

4.5 Granularita

Procesory se mezi sebou dotazují při každé 100. iteraci v algoritmu. Při zvýšení této hodnoty docházelo k jevu, že jen část procesorů se zabývala problémem a ostatní procesory neměly co na práci. V tomto případě byla granularita hrubá, procesory nebyly rovnovážně vytížené a program trval neúměrně dlouho. Zato v druhém případě, kdy se procesory stále dotazovaly na přijaté zprávy (případ snížení počtu iterací), docházelo k mnohočetnému posílání zpráv a komunikační náklady přerostly výpočetní složitost. Tudíž výpočet opět trval neúměrně dlouho.

5 Závěr

Daný problém, nalézt nejméně obarvenou kostru grafu, se zdál z prvního pohledu celkem jednoduchý díky algoritmu [2], ale při zpracování parelelní části se objevily potíže se správným rozdělováním zátěže, jelikož danou datovou strukturou byl téměř úplný binární strom. My jsme se rozhodli na začátku pro dynamické rozdělování práce, a tím nejspíš nastal ten problém, že algoritmus vůbec nezrychloval. Při zvyšování množství procesorů se navyšoval extrémně komunikační provoz. Algoritmus je tedy značně neefektivní a špatně škálovatelný. Nejspíše by pro zlepšení pomohlo jiné dělení zásobníku.

Literatura

[1] Josef Kolář: *Teoretická informatika*

[2] Pavel Tvrdík: *Paralelní systémy a algoritmy*