Semestrální projekt MI-PAR 2011/2012:

Paralelní algoritmus pro řešení problému

Tomáš Čerevka
Adam Činčura
magisterské studium, FIT ČVUT, Kolejní 550/2, 160 00 Praha 6

November 28, 2011

1 Definice problému

Úloha ZHV: Zobecněné Hanojské Věže

1.1 Vstupní data

n = přirozené číslo představující celkový počet žetonů,$n \geq 16$.
Žeton $i, i = 1, ..., n$, má průměr i.
s = přirozené číslo představující počet tyček,$n \div 4 \geq s > 3$
r = číslo cílové tyčky,$1 \leq r \leq s$
$V[1, .., s]$ = množina neúplných hanojských věží.

1.2 Definice

Hanojská věž o výšce k je věž k různých žetonů, které jsou uspořádány od nejmenšího k největšímu a rozdíly ve velikostech sousedních žetonů jsou vždy 1. Neúplná hanojská věž o výšce k je věž k různých žetonů, které jsou uspořádány od nejmenších k největším a rozdíly ve velikostech alespoň 1 dvojice sousedních žetonů je alespoň 2.

Například pro k=5 je to neúplná věž 2,3,7,8,10.

1.3 Generování počátečního stavu

V pořadí průměrů žetonů $n, n-1, .., 1$ se žetony náhodně rozhazují na s tyček, takže vznikne s obecně neúplných hanojských věží.

1.4 Pravidla

Jeden tah je přesun žetonu z vrcholu jedné věže na jedné tyčce na jinou prázdnou tyčku nebo na vrchol věže začínající žetonem s větším průměrem.

1.5 Úkol

Na zadané tyčce r postavte úplnou hanojskou věž o výšce n pomoci minimálního počtu tahů.

1.6 Výstup algoritmu

Výpis počtu tahů a jejich posloupností v následujícím formátu: žeton, původní tyčka $\rightarrow$ cílová tyčka.

1.7 Sekvenční algoritmus

Řešení musí existovat. Sekvenční algoritmus je typu BB-DFS s neomezenou hloubkou prohledávání (obecně se mohou pro $s > 3$ při prohledávání generovat cykly). Hloubku prohledávání musíme omezit horní mez (viz dále). Ve stavu, kdy nelze přesunout žádný žeton, se provede návrat. Cena, kterou minimalizujeme, je počet tahů. Algoritmus končí, když je počet tahů roven dolní mezi, jinak prohledává celý stavový prostor do hloubky dané horní mezí.

Těsnou dolní mez počtu tahů lze určit takto: je-li žeton na cílové tyčce, ale není na správné pozici, pak musí provést aspoň 2 tahy. Je-li žeton na jiné než cílové tyčce, pak musí provést aspoň 1 tah. Dolní mez je součet těchto minimálních počtů tahů pro všechny žetony. Tato dolní mez je dosažitelná pro dostatečně velká s.

2 Popis sekvenčního algoritmu

Program pro sekvenční algoritmus je napsán objektovým přístupem. Hrací deska, věž, tah tokenem a objekt ukládaný na zásobník jsou reprezentovány každý svou třídou. Veškerý výkonový kód algoritmu je obsažen ve třídě Solver. Implementaci vlastního zásobníku jsem neprováděli, použili jsme standardní implementaci z knihovny STL.

Algoritmus začíná načtením vstupních dat, připravením výchozího stavu a jeho uložením na zásobník. Poté algoritmus prohledává stavový prostor až do hloubky horní meze specifikované v zadání. Pokud algoritmus nalezne řešení, zmenší hloubku prohledávaného prostoru na hloubku nalezeného řešení - ve větší hloubce není možné nalézt lepší řešení. V případě, kdy algoritmus nalezne další řešení, lepší (v menší hloubce) než původní řešení, zmenší opět hloubku prohledávání na hloubku lepšího řešení.

Při provádění expanze zásobníku algoritmus kontroluje zda by provedením tahu nevznikl cyklus délky 1 (tah tokenem tam a zpátky). Pokud cyklus vznikl, pak příslušný stav není na zásobník uložen. Dále je při expanzi zásobníku kontrolováno zda nový stav může vést k řešení pomocí těsné dolní meze. Pokud nemůže vést k řešení, také není na zásobník uložen. Běh algoritmu je předčasně ukončen pokud je nalezeno řešení v hloubce rovné dolní mezi.

Algoritmus načítá zadání ze souboru. Název souboru je programu specifikován jako parametr příkazové řádky -f. Příklad spuštění programu, kdy je zadání specifikováno v souboru input1.txt, je následující:

compiledFile -f/path/to/input1.txt

Formát vstupního souboru je následující:

4 0 22
8 2
6 5
4 3
7 1

Význam souboru je následující: 4 věže, cílová věž je 0. v pořadí, maximální hloubka prohledávání je 22, následují řádky pro jednotlivé věže, každá věž je specifikována hodnotami svých tokenů v sestupném pořadí dle velikosti.

Výstupem algoritmu je posloupnost tahů, které postaví na požadované pozici kompletní hanojskou věž ve formátu:

$[2, 0 \rightarrow 2]$
$[1, 3 \rightarrow 2]$
$[7, 3 \rightarrow 0]$
$[5, 1 \rightarrow 3]$
$[1, 2 \rightarrow 3]$
$[6, 1 \rightarrow 0]$

$[2, 2 \rightarrow 1]$
$[1, 3 \rightarrow 1]$
$[5, 3 \rightarrow 0]$
$[3, 2 \rightarrow 3]$
$[4, 2 \rightarrow 0]$
$[3, 3 \rightarrow 0]$
$[1, 1 \rightarrow 3]$
$[2, 1 \rightarrow 0]$
$[1, 3 \rightarrow 0]$
Solution depth: 15

První číslo udává hodnotu tokenu, kterým je tah prováděn. Zbylá dvě čísla udávají index věže, z které byl token odebrán(před šipkou) a index věže, na kterou je token umístěn(za šipkou). Zde je naše jediná odchylka od zadání - číslujeme věže od 0 namísto od jedničky. Stejné číslování používáme i při načítání dat.

2.1 Doba běhu sekvenčního algoritmu

Prohledávaná hloubka	Doba běhu[s]
18	48
19	70
20	146
21	429
22	649

Table 1: Doba běhu při úplném prohledání stavového prostoru

3 Popis paralelního algoritmu a jeho implementace v MPI

Paralelní algoritmus je typu L-PBB-DFS-D, vznikl paralelizací sekvenčního algoritmu, proto jsou metody pro expanzi zásobníku a vyhodnocení stavu na vrcholu zásobníku stejné. Podstatnou změnou oproti sekvenčnímu řešení je

nahrazení zásobníku vektorem - také standardní implementace z STL. Toto nahrazení jsme provedli kvůli zvolenému dělení zásobníku u dna.

Po spuštění algoritmu proces MASTER(id=0) načte zadání, provede několik expanzí zásobníku a poté pošle všem ostatním procesům jejich díl práce. Po přijetí přidělené práce všechny procesy provádějí sekvenční prohledávání stavového prostoru(každý své části).

Každý proces může být ve stavu aktivní(neprázdný zásobník), nebo neaktivní(prázdný zásobník).

Aktivní proces provádí výpočet a každou 150. expanzi zkontroluje příchozí zprávy. Jak často provádět kontrolu zpráv je nastaveno konstantou a je možné libovolně měnit. Pokud má proces ve frontě nějaké příchozí zprávy, provádí jejich zpracování dokud není fronta prázdná. Pokud proces během výpočtu vyprázdní zásobník, přejde do stavu neaktivní.

Neaktivní proces vybere dárce a tomu pošle žádost o práci, pokud má u sebe peška, pošle ho následujícímu procesu ve směru virtuální hamiltonovské kružnice. Poté pouze obsluhuje případné příchozí zprávy. Při příchodu kladné zprávy od dárce uloží poslanou práci na zásobník a přechází do stavu aktivní, při záporné odpovědi dárce vygeneruje nového dárce a žádost o práci opakuje.

Pro spuštění algoritmu lze použít stejný příkaz jako pro spuštění sekvenční verze.

3.1 Algoritmus hledání dárce

Pro hledání dárce jsme zvolili algoritmus Náhodné výzvy (NV-AHD). Tento algoritmus pokaždé, když se proces stane nečinným vygeneruje náhodně index dárce z množiny $\{0, ..., (p-1)\} - \{i\}$, kde p je počet procesů a i je index žádajícího procesu.

3.2 Algoritmus dělení zásobníku

Zvolili jsme dělení zásobníku u dna. U naší úlohy se dá předpokládat, že stavy u dna zásobníku pod sebou skrývají více práce než stavy výše v zásobníku.

3.3 Algoritmus pro distribuované ukončení výpočtu

Použili jsme modifikovaný Dijkstrův peškový algoritmus. Tento algoritmus je vhodný při používání dynamického vyvažování zátěže.

4 Teoretická efektivita algoritmu

V nejhorším případě prohledáme sekvenčně celý stavový prostor. Velikost prostoru závisí na počtu věží s a na počtu tokenů n. Proto pro složitost sekvenčního řešení platí:

$$SU(s,n) = O(s^n)$$

Pro paralelní výpočet jsme se pokoušeli rozdělit stavový prostor rovnoměrně mezi všechny procesory. Výsledný paralelní čas je dán součtem doby výpočtu na jednom procesoruT_v a komunikačních nákladů T_k. Paralelní čas lze tady vyjádřit následovně:

$$T(s,n,p) = T_v + T_k = O(\frac{s^n}{p}) + T_k(s,n,p)$$

Pro cenu, zrychlení a efektivnost platí následující vztahy:

$$C(s,n,p) = p \times T(s,n,p) = p \times (\frac{s^n}{p}) + T_k(s,n,p) = O(s^n) + p \times T_k(s,n,p)$$

$$S(s,n,p) = \frac{SU(s,n)}{T(s,n,p)} = \frac{O(s^n)}{O(\frac{s^n}{p}) + T_k(s,n,p)} = O(p)$$

U výsledné $S(s,n,p)$je zanedbána komunikační režie.

$$E(s,n,p) = \frac{SU(s,n)}{C(s,n,p)} = \frac{O(s^n)}{O(s^n) + p \times T_k(s,n,p)}$$

Všechny uvedené vztahy předpokládají rovnoměrné rozdělení práce.

Pro náš algoritmus jsme stanovili hranici, za kterou už se nevyplatí výpočet paralelizovat na 8. Pokud procesor má na dně zásobníku stavy, kterým zbývá méně než 8 kroků k dosažení maximální prohledávané hloubky, už se o tyto stavy s nikým nedělí a výpočet dokončí sám.

5 Naměřené výsledky

Měření jsme provedli třikrát pro každý vstup a daný počet procesorů. Dále prezentované výsledky jsou průměrem těchto třech měření.

Měřili jsme dobu běhu paralelního algoritmu, počet odeslaných žádostí o práci a počet odmítnutých žádostí o práci.

Použité vstupy:

- Input1.txt - Obsahuje 4 věže, prohledávání probíhá do hloubky 20. Nejlepší řešení existuje v hloubce 18.
- Input2.txt - Obsahuje 4 věže, prohledávání probíhá do hloubky 23. Nejlepší řešení existuje v hloubce 17.
- Input3.txt - Obsahuje 4 věže, prohledávání probíhá do hloubky 22. Nejlepší řešení existuje v hloubce 15.

CPU	InfiniBand								
	Input1			Input2			Input3		
	Time	Requests	No work	Time	Requests	No work	Time	Requests	No work
1	1,991.04	0.00	0.00	371.93	0.00	0.00	1,497.74	0.00	0.00
2	405.58	22.00	2.00	267.42	15.33	2.33	308.52	34.00	2.00
4	244.07	75.00	13.33	166.89	77.00	15.00	213.25	109.00	10.33
8	122.71	340.33	141.00	126.63	324.00	116.67	181.24	426.67	206.00
16	95.46	903.00	254.33	51.41	1,140.00	607.33	35.69	919.00	439.00
24	49.72	1,884.00	921.67	29.25	1,922.67	862.67	22.90	2,373.67	1,322.00
32	53.03	4,013.67	2,053.00	19.73	3,051.00	1,559.33	24.42	2,727.67	1,565.00

Figure 1: Tabulka naměřených hodnot pro síť InfiniBand

CPU	Ethernet								
	Input1			Input2			Input3		
	Time	Requests	No work	Time	Requests	No work	Time	Requests	No work
1	1,991.04	0.00	0.00	371.93	0.00	0.00	749.37	0.00	0.00
2	403.85	22.67	2.00	265.17	14.67	3.33	227.48	145.95	8.22
4	239.75	81.00	21.67	167.06	76.00	13.33	160.81	121.14	31.56
8	122.77	255.67	63.67	127.35	355.00	153.67	118.77	208.48	158.33
16	90.23	811.00	145.00	48.65	1,438.67	877.00	59.68	454.74	467.89
24	53.17	2,254.33	1,220.67	29.19	1,737.33	644.00	31.85	1,288.50	1,484.44
32	41.97	3,549.67	1,813.67	20.15	4,263.00	2,480.33	28.54	1,502.32	1,919.56

Figure 2: Tabulka naměřených hodnot pro síť Ethernet

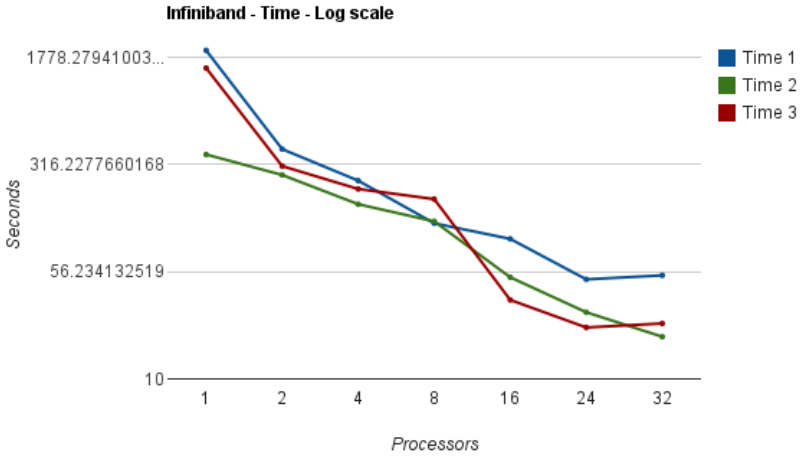

Figure 3: Doba běhu algoritmu na síti InfiniBand - časová osa má logaritmické měřítko

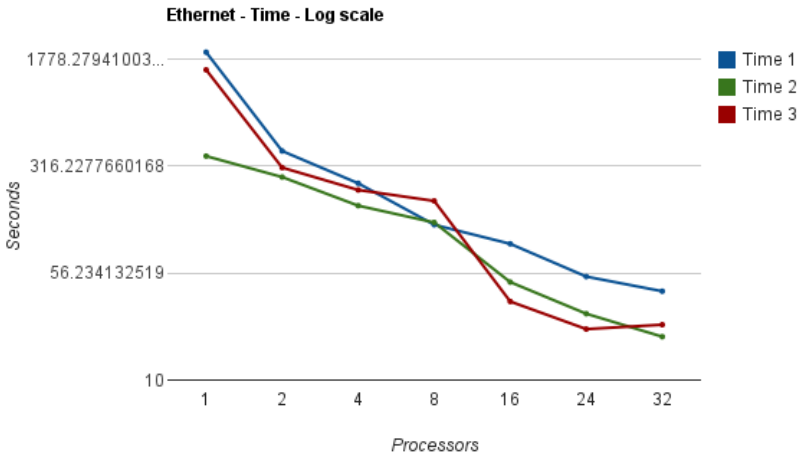

Figure 4: Doba běhu algoritmu na síti Ethernet - časová osa má logaritmické měřítko

Jak je patrné z předchozích grafů, náš algoritmus běží stejně rychle na síti InfiniBand jako na síti Ethernet. Stejnou dobu běhu si vysvětlujeme tím, že naše aplikace komunikuje krátkými zprávami($\leq 1kb$). Dalším důvodem je zvolené dělení zásobníku u dna, kdy posílané stavy pod sebou skrývají velké množství práce díky čemuž neprobíhá komunikace nijak intenzivně.

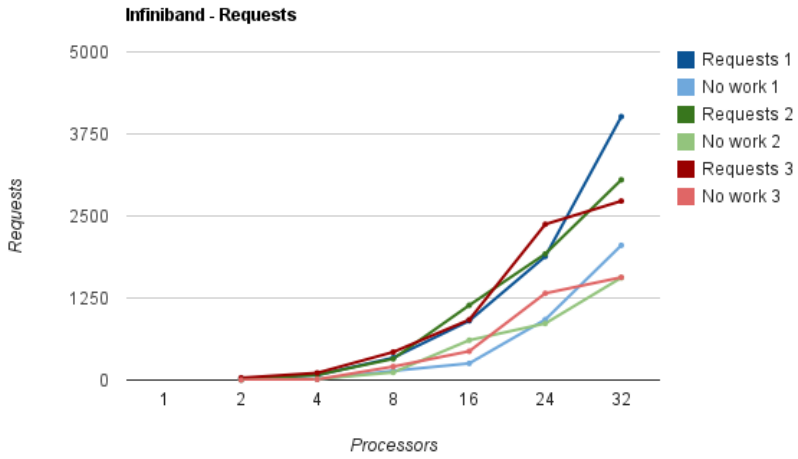

Figure 5: Počty žádostí o práci a zamítnutých žádostí o práci na síti InfiniBand

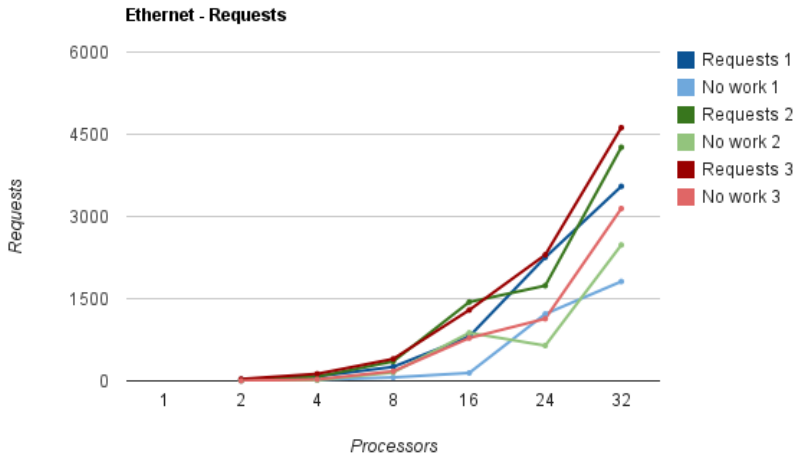

Figure 6: Počty žádostí o práci a zamítnutých žádostí o práci na síti Ethernet

Výše uvedené grafy zobrazující počty žádostí o práci ukazují, že námi

navržené komunikační schéma je velmi účinné pro pro malé počty procesorů. Při použití nejvýše 8 procesorů přesahuje úspěšnost žádostí o práci 90% . Při zvyšujícím se počtu procesorů úspěšnost žádostí klesá až k 50% . Do tohoto čísla se negativně promítá doba těsně před koncem výpočtu, kdy dochází práce a procesory se pokoušejí nějakou získat. Dá se proto předpokládat, že během výpočtu mají žádosti o práci vyšší úspěšnost.

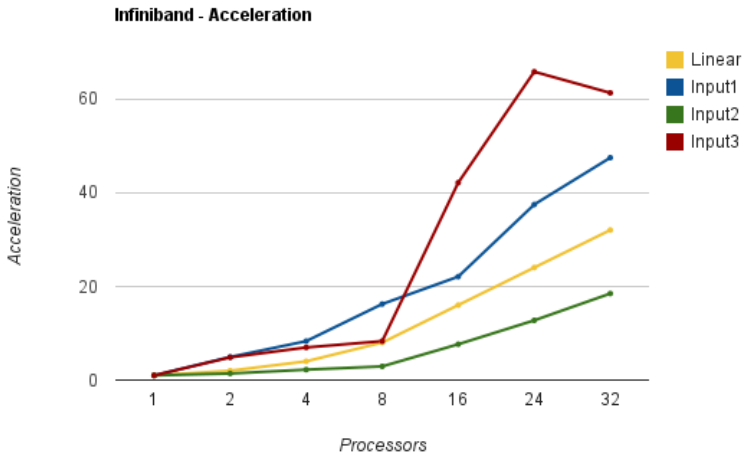

Figure 7: Graf paralelního zrychlení pro jednotlivé vstupy.

Z grafu zrychlení vyplývá, že pro input1 a input3 dosáhl náš algoritmus superlineárního zrychlení. K superlineárnímu zrychlení došlo pravděpodobně protože bylo výrazně dříve nalezeno řešení v menší hloubce, než byla hloubka prohledávaná. Tím došlo k omezení prohledávané hloubky a tedy i redukci počtu prohledávaných stavů. Tomu odpovídá i velký rozdíl v prohledávané hloubce a hloubce nalezeného řešení.

6 Závěr

Během vypracovávání semestrální práce jsme se setkali s problémy, které se při běžném sekvenčním programování nevyskytují. To nás nutilo zamyslet se nad nimi z jiného úhlu pohledu, než na který jsme byli doposud zvyklí.

Jednalo se především o techniky dělby práce mezi procesory, jejich vzájemná komunikace a bitová serializace dat pro posílání po síti.

Kromě toho jsme byli nuceni osvěžit si správu paměti v C++, především používání referencí. Odstraněním nadbytečných kopírujících konstruktorů jsme dosáhli až šesti násobného zrychlení sekvenčního řešení.

Při samotném měření na svazku Star jsme se potýkali s nemalým množstvím problémů, které se při testování na lokálním počítači nevyskytovaly. Prvním problémem bylo cachování zpráv a jejich dávkové zpracovávání, kvůli čemuž algoritmus selhával při ukončování výpočtu. Na další problém jsme narazili při měření na síti Ethernet, která nestíhala odesílat asynchronní požadavky, které jsme již považovali za odeslané, dostatečně rychle, takže se nám přepisoval buffer zpráv. Tato chyba se odhalovala obzvláště špatně, jelikož se tvářila jako přetečení bufferu.

Za hlavní přínos práce považujeme praktickou zkušenost s paralelním programováním.

7 Literatura

- Prof. Ing. Pavel Tvrdík, CSc., Paralelní systémy a algoritmy, ČVUT, Praha 2006
- Doc. Jiroušek Radim, DrSc., Metody reprezentace a zpracování znalostí v umělé inteligenci, VŠE, Praha 1995