Semestralní projekt MI-PAR 2011/2012:

Paralelní algoritmus pro řešení problému

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

November 26, 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 hanoiských 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. Neuplná 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 nasledujícím formátu: žeton, původni 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 dolni 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.

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 kružnici. 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 Naměřené výsledky a vyhodnocení

1. Zvolte tři instance problému s takovou velikostí vstupních dat, pro které má sekvenční algoritmus časovou složitost kolem 5, 10 a 15 minut. Pro meření čas potřebný na čtení dat z disku a uložení na disk neuvažujte a zakomentujte ladící tisky, logy, zprávy a výstupy.

2. Měřte paralelní čas při použití $i = 2, \cdot, 32$ procesorů na sítích Ethernet a InfiniBand.

3. Z naměřených dat sestavte grafy zrychlení $S(n, p)$. Zjistěte, zda a za jakych podmínek došlo k superlineárnímu zrychlení a pokuste se je zdůvodnit.

4. Vyhodnoďte komunikační složitost dynamického vyvažování zátěže a posuďte vhodnost vámi implementovaného algoritmu pro hledání dárce a dělení zásobníku pri řešení vašeho problému. Posuďte efektivnost a škálovatelnost algoritmu. Popište nedostatky vaší implementace a navrhněte zlepšení.

5. Empiricky stanovte granularitu vaší implementace, tj., stupeň paralelismu pro danou velikost řešeného problému. Stanovte kritéria pro stanovení mezí, za kterými již není učinné rozkládat výpočet na menší procesy, protože by komunikační náklady převážily urychlení paralelním výpočtem.

5 Závěr

Celkové zhodnocení semestrální práce a zkušenosti získaných během semestru.

6 Literatura

A Návod pro vkládání grafů a obrázků do Latexu

Nejjednodušší způsob vytvoření obrázku je použít vektorový grafický editor (např. xfig nebo jfig), ze kterého lze exportovat buď

- postscript formáty (ps nebo eps formát) nebo
- latex formáty (v pořadí prostý latex, latex s macry epic, eepic, eepicemu). Uvedené pořadí odpovídá růstu komplikovanosti obrázků který formát podporuje (prostá latex macra umožnují pouze jednoduché, epic makra něco mezi, je třeba vyzkoušet).

Následující příklady platí pro všechny případy.

Obrázek v postscriptu, vycentrovaný a na celou šířku stránky, s popisem a číslem. Všimnete si, jak řídit velikost obrazku.

Latexovské obrázky maji přípony *.latex, *.epic, *.eepic, a *.eepicemu, respective.

Vypuštením závorek `figure` dostanete opět pouze rámeček v textu bez čísla a popisu.

Takhle jednoduše můžete poskládat obrázky vedle sebe.

Řídit velikost latexovskych obrázků lze příkazem

```
\setlength{\unitlength}{0.1mm}
```

které mění měřítko rastru obrázku, Tyto příkazy je ale současně nutné vyhodit ze souboru, který xfig vygeneroval.

Pro vytváření grafu lze použít program gnuplot, který umí generovat postscriptovy soubor, ktery vložíte do Latexu výše uvedeným způsobem.