**Semestrální projekt MI-PAR 2010/2011:**

Paralelní algoritmus pro řešení problému **Jeskyně pokladů (JSP)**

Daniel Kavan <[kavanda1@fit.cvut.cz](mailto:kavanda1@fit.cvut.cz)>

Michal Včelička <[vcelimic@fit.cvut.cz](mailto:vcelimic@fit.cvut.cz)>

magisterské studium, FIT ČVUT, Kolejní 550/2, 160 00 Praha 6

8. prosinec 2010

# Definice problému a popis sekvenčního algoritmu

## Zadání úlohy Jeskyně pokladů (JSP)

### Úkol

Náhodou se shodou příznivých okolností ocitnete v jeskyni pokladů. Je zde mnoho (přesněji n) předmetů. U každého předmětu znáte jeho cenu a objem. Chcete si z této jeskyně odnést předměty v co největší celkové hodnotě, předměty ale můžete odnést jen ve svém batohu. Máte batoh, kam se vejde neomezený počet předmětů, ale má maximální objem *V*.

Úkolem je nalézt charakteristický vektor *a* množiny odnesených věcí, tak aby max(*C*.*a*), kde *O*.*a* ≤ *V*.

### Vstupní data:

|  |  |  |
| --- | --- | --- |
| **Proměnná** | **Matematická interpretace** | **Popis** |
| *n* | konstanta *n* | počet předmětů |
| *C[1..n]* | cenový vektor *C* | ceny jednotlivých předmětů, 0.9 ≤ *C[i]* ≤ 1.1 |
| *O[1..n]* | vektor objemu *O* | objemy jednotlivých předmětů, 0.9 ≤ *O[i]* ≤ 1.1 |
| *V* | konstanta *V* | maximální objem předmětů, co lze odnést |

Tabulka - Vstupní data úkolu

### Výstup algoritmu:

Výpis množiny předmětů v batohu včetně celkové ceny a celkového objemu.

## Popis sekvenčního algoritmu

Sekvenční algoritmus je typu **BB-DFS**[[1]](#footnote-1), s hloubkou stromu stavů omezenou na *n*. Řešení vždy existuje. Cena, kterou maximalizujeme, je součet cen věcí v batohu. Algoritmus končí, když je cena rovna horní mezi nebo když prohledá celý stavový prostor do hloubky dané *n*.

Těsná horní mez není známa. Odhad horní meze ceny lze vypočítat jako *c\_max*i=*V*(max *C*i/*O*i).

### Nástin algoritmu:

Stav je hodnota charakteristického vektoru *a*. Procházení stromu se tedy rovná postupnému nastavování složek vektoru *a* zleva čísly 0 nebo 1. Při prohledávání větve stromu (ohodnoceno je prvních i položek vektoru *a*) se lze vrátit, pokud

1. cena předmětů v batohu + cena zbývajících dosud neuvažovaných věcí ≤ průběžné maximum
2. cena předmětů v batohu + maximální odhad ceny nejlepšího doplnění do batohu ≤ průběžné maximum

### Odchylky od zadání

V zadání byla specifikována omezení pro hodnoty ve vektorech *O* a *C*:

0.9 ≤ *C[i]* ≤ 1.1

0.9 ≤ *O[i]* ≤ 1.1.

Vstupy s takto omezenými hodnotami bez problémů zpracováváme, nicméně naše implementace si poradí i s hodnotami mimo tento interval. Problémy nám nečiní ani

* předměty těžší než 1.1 (smysl úlohy nezměněn)
* předměty hodnotnější než 1.1 (smysl úlohy nezměněn)
* předměty lehčí než 0.9 (předměty se zápornou hmotností mohou batoh dokonce nadlehčovat)
* předměty lacinější než 0.9 (předměty se zápornou cenou mohou negativně cenově působit na „ostatní obsah batohu“).

### Naměřené časy pro sekvenční algoritmus různě velká data

|  |  |
| --- | --- |
| Počet předmětu v batohu (*n*) | Čas potřebný k řešení úlohy[[2]](#footnote-2) [s] |
| 4 | 0.000593901 |
| 10 | 0.0027411 |
| 15 | 0.116669 |
| 20 | 3.42538 |
| 22 | 15.9661 |
| 25 | 112.343 |
| 27 | 550.386 |

Tabulka - Naměřené časy pro různé velikosti batohu

## Implementace sekvenčního algoritmu

Implementujeme algoritmus typu DFS pro stromové prohledávání do hloubky s vlastním zásobníkem. Jasně neperspektivní stavy a jejich podstromy (např. aktuální váha batohu s danými předměty by překročila povolené maximum) v rámci implementace kroku **pruning**[[3]](#footnote-3) vyřazujeme.

Průchod stromem v naší implementaci probíhá zjednodušeně následovně:

1. Na zásobník je uložen kořenový prvek stromu.
2. Na zásobníku existuje minimálně jeden stav. Je vyjmut a otestován, zda je možné ho expandovat
   1. Expanze možná → expandováno → test, zda jsou nové stavy perspektivní
      1. Každý perspektivní stav je vrácen na zásobník
      2. Každý neperspektivní stav je zahozen
   2. Expanze není možná (list stromu) → test, zda je list lepším řešením než „zatím nejlepší“
      1. List je lepším řešením → list se stává „zatím nejlepším“
      2. List není zlepšujícím řešením → je zahozen
3. Jakmile je zásobník prázdný → „zatím nejlepší“ je řešením problému.

Ve zdrojovém kódu aplikace je tato implementace sekvenčního algoritmu zpracována ve funkci procedeNode.

### Formát vstupních dat

Vstupní data pro naši implementaci předpokládáme v textovém souboru, který obsahuje:

* počet předmětů
* objem batohu
* dvojice hodnot *value*, *volume* jednotlivých předmětů v počtu právě 2*n*

oddělených znaky typu *whitespace* (mezery, tabulátory, CR, LF, …)

3 2.7 1.05 1 1.04 1 1.03 1

Ukázka kódu - Formát vstupních dat

Ukázka kódu 1 demonstruje formát vstupních dat na 3 předmětech, jejichž hodnoty jsou po řadě 1.05, 1.04, 1.03 a objemy vždy roven 1 a batohu o velikosti 2.7.

### Formát výstupních dat

Výstupní data jsou opět v textové formě, hlavními částmi jsou

* oznámení počtu procesorových jader podílejících se na výpočtu
* zvolený typ zásobníku
* průběžná lokální maxima jednotlivých vláken
* výsledné optimální naplnění batohu (tj. max. cena, hmotnost, binární vektor obsahu batohu)
* spotřebovaný čas pro běh úlohy [s]

P0:There are 2 processes.

StackType: STACK

... případná lokální maxima a dodatečné informace ...

P0:solution with the value of 3538 is:

---Node Content:--- (0)

P0:Vector content: <1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 0 0 0 1 >

P0:Volume: 348

Spotrebovany cas je: 267.985

Ukázka kódu - Formát výstupních dat

# Popis paralelního algoritmu a jeho implementace v MPI

Paralelní algoritmus je typu PBB-DFS-V.

# Naměřené výsledky a vyhodnocení

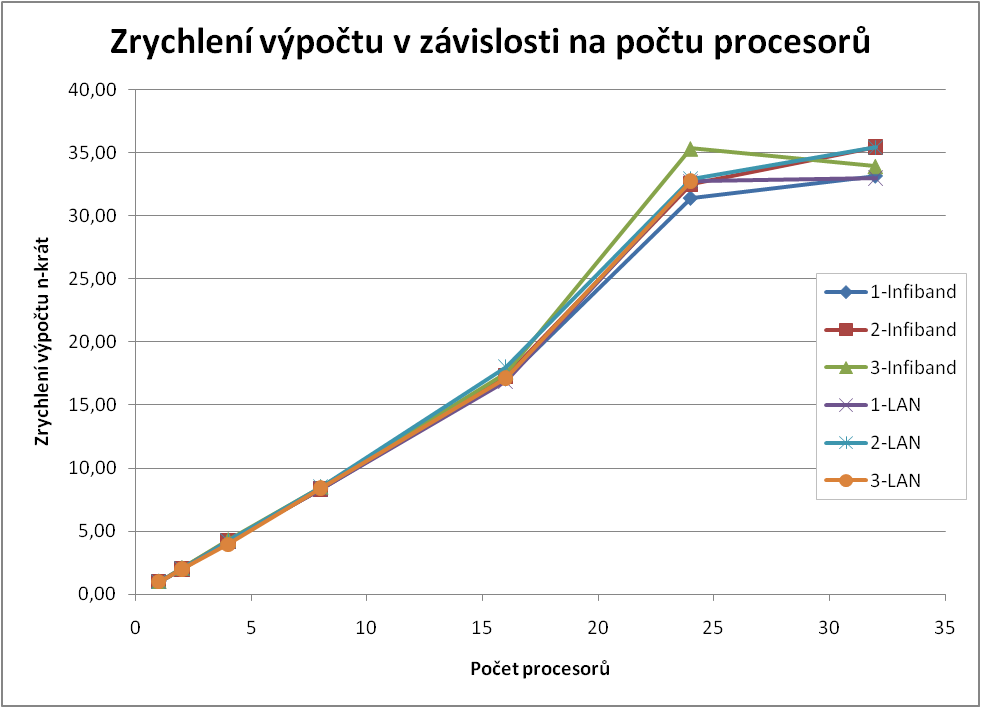
Naměřené časy pro jednotlivé počty procesorů:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| # CPU's | 1 - Infiband | 2 - Infiband | 3 - Infiband | 1 - LAN | 2 - LAN | 3 - LAN |
| 1 | 550,39 | 562,66 | 562,99 | 550,97 | 573,27 | 540,66 |
| 2 | 267,99 | 280,16 | 272,84 | 271,59 | 280,61 | 271,35 |
| 4 | 131,54 | 133,15 | 131,16 | 130,91 | 134,77 | 136,78 |
| 8 | 66,05 | 67,62 | 66,31 | 65,86 | 67,18 | 64,30 |
| 16 | 32,26 | 32,57 | 32,04 | 32,67 | 31,82 | 31,59 |
| 24 | 17,53 | 17,31 | 15,94 | 16,84 | 17,43 | 16,50 |
| 32 | 16,60 | 15,89 | 16,60 | 16,70 | 16,20 |  |

Hodnoty zrychlení oproti řešení na jednom procesoru:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| # CPU's | 1-Infiband | 2-Infiband | 3-Infiband | 1-LAN | 2-LAN | 3-LAN |
| 1 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 | 1,00 |
| 2 | 2,05 | 2,01 | 2,06 | 2,03 | 2,04 | 1,99 |
| 4 | 4,18 | 4,23 | 4,29 | 4,21 | 4,25 | 3,95 |
| 8 | 8,33 | 8,32 | 8,49 | 8,37 | 8,53 | 8,41 |
| 16 | 17,06 | 17,27 | 17,57 | 16,86 | 18,02 | 17,12 |
| 24 | 31,40 | 32,50 | 35,33 | 32,71 | 32,89 | 32,76 |
| 32 | 33,16 | 35,42 | 33,92 | 33,00 | 35,40 |  |

Graf zavisloti zrychlení na počtu procesorů. (Graf jednotlivých časů jsme vynechali, protože pro názornost řešení se nám tento zdál více vypovídající):



# Závěr

# Literatura

1. C++ Reference
   1. STL vector <<http://www.cplusplus.com/reference/stl/vector/>>
   2. STL deque (double ended queue) <<http://www.cplusplus.com/reference/stl/deque/>>
2. Šoch, M.: Programovaní pod MPI (pro MI-PAR) <<https://users.fit.cvut.cz/~soch/mi-par/>>
3. Šimeček, I.: Poznámky k implementaci (pro MI-PAR)  
   <<https://edux.fit.cvut.cz/courses/MI-PAR/labs/poznamky_k_implementaci>>
4. Wikipedia.org: Branch and Bound <<http://en.wikipedia.org/wiki/Branch_and_bound>>

1. Branch-and-Bound Depth-First Search [↑](#footnote-ref-1)
2. Čas vybrané úlohy pro zvolený počet předmětů *n*, přesný čas vždy závisí na datech, zde klademe důraz na řád výsledků. [↑](#footnote-ref-2)
3. Krok BB algoritmu, kdy je neperspektivní podstrom vyřazen z prohledávání. Viz zdroj [4]. [↑](#footnote-ref-3)