Semestrální projekt č. 2 z předmětu AD4M36PAP, školní rok 2012-2013

Téma: Tvorba programu pro paralelní výpočetní systém

Hledání nejkratších cest ohodnoceným grafem

ČVUT FEL, obor VT, 1. ročník, e-mail: valiclud@fel.cvut.cz

Jméno: Ludvík Valíček
Datum: 20. prosince 2012

1. Zadání

Semestrální projekt paralelizuje výpočet vyhledávání nejkratší cesty orientovaným grafem mezi všemi vrcholy. Vyhledání nejkratší cesty se řeší Dijkstrovovým a Floyd-Warshallovým algoritmem. Tyto algoritmy jsem již naprogramoval jako součást semestrální práce z předmětu PJC v bakalářském studiu. Projekt řeší a vyhodnocuje paralelizaci pomocí OpenMP, MPI a kombinací OpenMP+MPI, experimenty měření času výpočtu jsou dále vyhodnoceny.

Vstupem programu je vstupní soubor s váženou maticí sousednosti. Matice sousednosti je zapsána ve vstupním textovém souboru po řádcích a sloupcích, jednotlivá čísla matice (ohodnocení grafu) na řádku jsou oddělena mezerou. Program zpracovává pouze čísla ve formátu integer, čísla ve formátu double jsou automaticky konvertována na integer.

Výstup programu je přes řádkové rozhraní, ve formátu počáteční vrchol, koncový vrchol a váha nejkratší cesty. U Dijkstrovova algoritmu jsou navíc výstupem vrcholy, přes které jsme se k cílovému vrcholu dostaly.

Paralelizace je prováděna v prostředí Windows 7, 4-jádrový procesor Intel Core i5-460M 2,53 MHz, vývojové prostředí Netbeans 7.2.

2. Popis problému

2.1. Dijkstrův algoritmus

Dijkstrův algoritmus nalezne nejkratší cestu v orientovaném, ohodnoceném grafu, ohodnocení grafu musí být nezáporné.

Popis vlastního algoritmu:

Je zadán graf *G*, ve kterém *V* je množina všech vrcholů a E množina všech hran. Označme d[v] jako délku nejkratší cesty, kterou je možné se dostat do vrcholu V.

Před vlastním řešením nastavíme hodnotu d[v] pro všechny vrcholy na nekonečno (v našem případě na hodnotu max int), mimo počátečního vrcholu s, ze kterého budeme hledat nejkratší cesty do zbývajících vrcholů. U počátečního vrcholu nastavíme d[v] na hodnotu nula.

Algoritmus si dále pamatuje dvě množiny, pole mark[i] (zde nastavuje příznak „1“ u již navštívených vrcholů) a pole actual[count++] (zde ukládá již navštívený vrcholy a inkrementuje proměnnou count). Algoritmus pracuje v cyklu tak dlouho, dokud proměnná count se nerovná počtu vrcholů. V každém průchodu cyklu se přidá jeden vrchol *vmin* do pole actual[count++] a to takový, který má nejmenší hodnotu d[v] ze všech vrcholů V.

Při každém průchodu cyklu se pro každý vrchol w, do kterého vede hrana z *vmin* (délku hrany označme jako *l(vmin,w)*), provede následující operace: pokud (*d[vmin]* + *l(vmin,w)*) < *d[w]*, pak do *d[w]* se přiřadí hodnota *d[vmin]* + *l(vmin,w)*, jinak se neprovede žádná operace.

Pseudokód:

```
1   function Dijkstra(E, V, s):
2   for each vertex v in V:      // Inicializace
3       d[v] := maxInt           // Nastavení neznámé vzdálenosti z počátku s do vrcholu v
4       last[v] := undefined     // Předchozí vrchol na nejkratší cestě z počátku s k cíli
5       d[s] := 0                // Vzdálenost z s do s
6       count := 0               // Počitadlo navštívených vrcholů
7   while count is lower v(max):              // Samotný algoritmus
```

```
8       count++
9       min := minimal(V)                          // Nalezení vrcholu s nejnižším ohodnocením
10        for each non-visited neighbor v of min:    // Hledání cesty s nejnižším ohodnocením
11          if  d[v] >  d[min] + l(min, v)    then
12            d[v] := d[min] + l(min, v)
13            last[v] := min
```

Asymptotická časová složitost Dijkstrovova algoritmu odpovídá času potřebnému na vykonání funkce extract_min(V) (lineární prohledávání všech vrcholů) a je $O(|V|^2+|E|)$, kde |V| je počet vrcholů a |E| počet hran. Jedná se o nejjednodušší implementaci Dijkstrova algoritmu, který používá pro uložení prioritní fronty pole.

2.2. Floyd-Warshallův algoritmus

Floydův–Warshallův algoritmus nalezne nejkratší cestu tak, že porovnává všechny možné cesty v grafu mezi všemi dvojicemi vrcholů. Pracuje způsobem, že postupně vylepšuje odhad na nejkratší cestu do té doby, než je zřejmé, že odhad je optimální. Ohodnocení grafu musí být také nezáporné.

Popis vlastního algoritmu:

Je také zadán graf G, V je množina vrcholů 1, až N. Označme funkci min(i,j,k), která vrací nejkratší možnou cestu z i do j s použitím pouze vrcholů 1 až k jako mezi-vrcholů.

Při výpočtu nejkratší cesty existují dvě možnosti. Buď je nejkratší cesta v množině vrcholů 1 až k, nebo existuje cesta z vrcholu i do k+1 a z k+1 do j, která je kratší než stávající. Porovnáme tedy hodnotu D[i][j] s hodnotou D[i][k] + D[k][j]. Kratší z nich uložíme do prvku D[i][j]. Po absolvováni všech N fázi je algoritmus u konce a můžeme z matice sousednosti zjistit délky nejkratších cest mezi všemi dvojicemi vrcholů v grafu.

Pseudokód:

```
1 // graph[i][j], obsahuje váhu hrany z i do j, pokud hrana
2 // neexistuje, graph [i][j] = max int, N je počet vrcholů a graph([i][i] = 0
3  function Floyd_Warshall ()
4    for k: = 1 to N
5    begin
6      foreach (i,j) in (1..N)
7      begin
8        graph[i][j] = min(graph[i][j], graph[i][k] + graph[k][j]);
9      end
10   end
11 endproc
```

Algoritmus pracuje se třemi vnořenými cykly, které načítají do počtu vrcholů N, a proto asymptotická časová složitost Floyd-Warshallova algoritmu je $O(N^3)$.

2.3. Implementace

Program byl napsán a odladěn ve vývojovém prostředí Netbeans 6.8, kompilátor Cygwin.

Program je rozdělen do dvou tříd, Dijkstr.cpp a Floyd_Warshall.cpp spolu s příslušnými hlavičkovými soubory. Čtení souboru se provádí metodou inputMatrix (soubor InputMatrix.cpp). Celý program je poté spouštěn metodou main, uložené v samostatném souboru Main.cpp.

Při vývoji programu byly použity tyto cizí knihovny:
- <stdlib.h>
- <iostream>
- <limits>
- <stdio.h>
- <fstream>
- <iomanip>
- <istream>
- <cstdlib>
- <sstream>

Pro testování souboru bylo vytvořeno několik testovacích souborů, např. tři testovací soubory Test1.txt (matice 3x3), Test2.txt (matice 4x4, která obsahuje i záporné ohodnocení) a Test3.txt (matice 5x5), pro další paralelizaci soubory s maticí až do rozsahu 240 x 240. Program je kompletně ošetřen pro všechny možné chyby vstupních souborů.

2.4. Porovnání obou algoritmů

Při spouštění programu za použití testovacích souborů se projevují výhody a nevýhody jednotlivých algoritmů.

Výhodou Dijkstrovova algoritmu oproti algoritmu Floyd-Warshall je, že počítá i s ohodnocením samotného vrcholu, což Floyd-Warshallův algoritmus neumožňuje. Nalezne-li Floyd-Warshallův algoritmus ohodnocení samotného vrcholu, dosadí namísto tohoto ohodnocení nulovou hodnotu, proto se výsledky výpočtu nejkratší cesty u testovacího souboru Test3.txt liší.

Oba dva algoritmy požadují nezáporné ohodnocení matice sousednosti. Pro grafy s hranami se záporným ohodnocením se používá pomalejší Bellmanův-Fordův algoritmus.

3. Paralelizace pomocí OMP

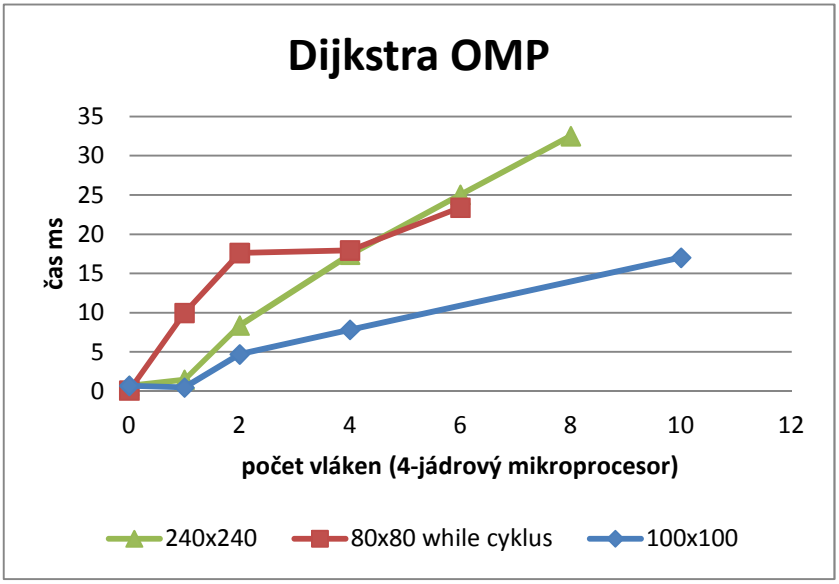
3.1. Dijkstrův algoritmus

Při paralelizaci Dijkstrovova algoritmu jsem se nejdřív rozhodl upravit stávající program paralelizací vnitřní smyčky while. Vnitřní for cyklus jsem rozdělil podle počtu vláken a v kritické sekci byl načítán čítač smyčky while, po přetečení čítače byla paralelizace ukončena omp příkazem flush. Výsledky takto provedené paralelizace byly pomalejší než neparalelizovaný program, viz tabulka 1 (ozn. 80x80 while cyklus). Projekt paralelizace smyčky while má název ParallelGraphs_OMP1.

Pomocí internetu, jsem se snažil najít lepší řešení tohoto problému. Bylo třeba přepsat stávající program z C++ do C a pozměnit algoritmus, aby bylo možné výpočet snáze paralelizovat. V programu jsou použity tři „pragme omp single“ (získání počtu vláken, nastavení hodnot algoritmu na nulu a nekonečno a označení nového, správného vrcholu). A jedna kritická sekce pro nastavení nového minimálního vrcholu, který byl nalezen mezi jednotlivými vlákny. Výsledky této paralelizace byly také pomalejší než neparalelizovaný program, viz tabulka 1 (ozn. 100x100 a 240x240). Tento program je uveden v projektu ParallelDijkstra_OMP, který obsahuje dvě main třídy (paralelizovaný-Dij.c a neparalelizovaný výpočet-DijNonParallel.c), jedna z těchto tříd je vždy zakomentovaná. Hlavní výpočet algoritmu je v metodě dowork().

Výsledky měření uvedené v tabulce a zpracované grafem:

Pos	Počet vláken	Rozměry vážené matice sousednosti	Měření 1	Měření 2	Měření 3	Průměrné naměřené hodnoty		
1	0	80 x 80 while cyklus	0,09806	0,12145	0,10568			0,1084
2	0	100 x 100	0,109	0,11386	0,133314	0,7074		
3	0	240 x 240	0,71358	0,60174	0,80677		0,7074	
4	1	80 x 80 while cyklus	9,20801	11,5555	9,17516			9,9796
5	1	100 x 100	0,4514	0,53366	0,46801	0,4844		
6	1	240 x 240	1,6678	1,35664	1,4316		1,4853	
7	2	80 x 80 while cyklus	17,8698	17,0626	17,9723			17,6349
8	2	100 x 100	4,3953	5,12105	4,6238	4,7134		
9	2	240 x 240	8,30398	8,629702	8,25698		8,3969	
10	4	80 x 80 while cyklus	16,637	19,3569	17,8856			17,9598
11	4	100 x 100	7,94861	8,08233	7,48748	7,8395		
12	4	240 x 240	16,87946	17,40015	17,98325		17,421	
13	6	80 x 80 while cyklus	24,6648	24,024	21,4839			23,3909
14	6	100 x 100	NO	NO	NO			
15	6	240 x 240	24,94154	25,48127	24,69881		25,0405	
16	10	100 x 100	17,60722	16,35714	17,11813	17,0275		
17	8	240 x 240	32,15632	34,01826	31,38479		32,5198	

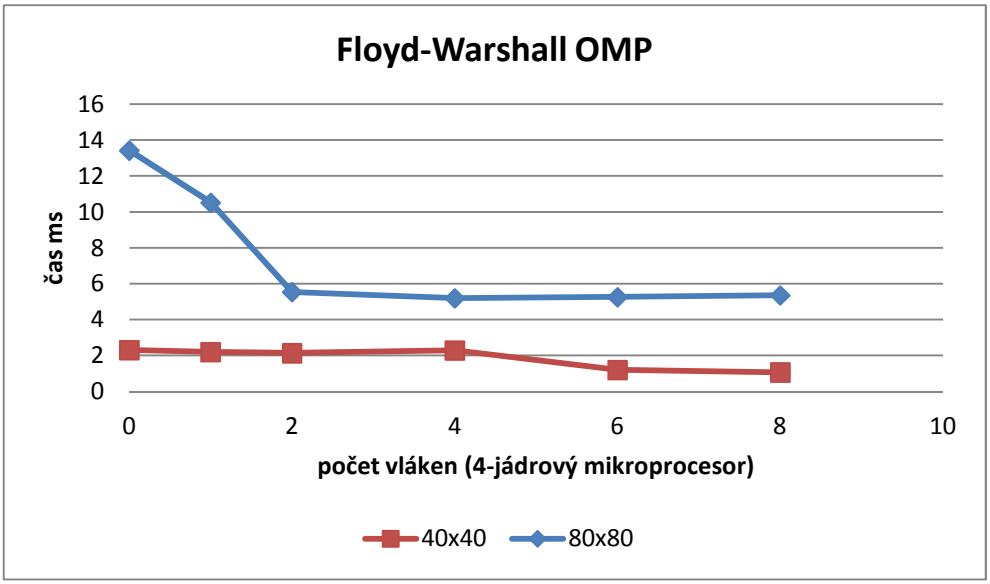


3.2. Floyd-Warshallův algoritmus

Paralelizace Floyd Warshallova algoritmu byla snazší, algoritmus obsahuje tři vnořené (nested) smyčky, které jsem paralelizoval pomocí direktivy „#pragma omp for collapse (3)“. Projekt paralelizace Floyd-Warshallova algoritmu má název ParallelGraphs_OMP1.

Výsledky měření uvedené v tabulce a zpracované grafem:

Pos	Počet vláken	Rozměry vážené matice sousednosti	Měření 1	Měření 2	Měření 3	Průměrné naměřené hodnoty	
1	0	80 x 80	14,6228	13,3229	12,3131	13,4196	
2	0	40 x 40	2,3555	2,7445	1,83682		2,3123
3	1	80 x 80	11,0481	10,5797	9,92888	10,5189	
4	1	40 x 40	2,15329	2,01957	2,42762		2,2002
5	2	80 x 80	5,21993	5,89703	5,51451	5,5438	
6	2	40 x 40	1,95676	2,19705	2,27364		2,1425
7	4	80 x 80	4,93506	5,03312	5,62027	5,1962	
8	4	40 x 40	2,31254	2,28053	2,28174		2,2916
9	6	80 x 80	5,74994	5,26045	4,76933	5,2599	
10	6	40 x 40	1,3493	0,956298	1,29586		1,2005
11	8	80 x 80	5,10242	5,23532	5,73616	5,358	
12	8	40 x 40	1,00776	0,95872	1,2367		1,0677



4. Paralelizace pomocí MPI, Floyd-Warshallův algoritmus

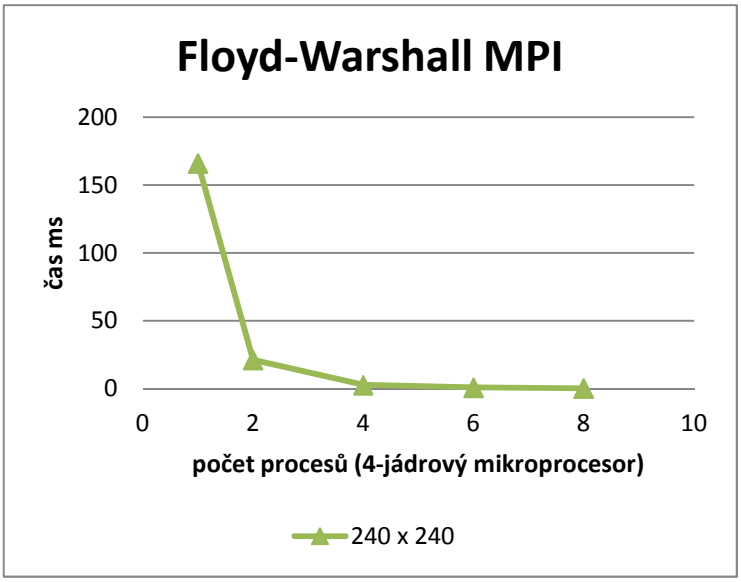
Paralelizace je provedena v samostatném projektu Parallel_MPI. Byly použity tyto MPI funkce:

- MPI_Init(&argc,&argv): inicializace MPI procesu,
- MPI_Finalize(): ukončení MPI porcesu,
- MPI_Comm_size(MPI_COMM_WORLD,&numtasks): získání počtu procesů v teamu, tj. paralelizovaná smyčka může být rozdělena mezi „členy“ teamu,
- MPI_Comm_rank(MPI_COMM_WORLD,&rank): získání ID čísla procesu v teamu,
- MPI_Bcast(&stopping, 1, MPI_INT, 0, MPI_COMM_WORLD): vysílá hodnotu z jednoho procesu ostatním procesům teamu,
- MPI_Reduce(&graph, &graphFinal, 1, MPI_INT, MPI_MIN,0, MPI_COMM_WORLD): „redukuje“ výsledky paralelních výpočtů pomocí MPI operace MPI_MIN a ukládá výslednou zprávu (message) do proměnné graphFinal. Výsledná zpráva je velikosti 1, datového typu integer a je vložena do proměnné graphFinal pouze u procesů, jejichž „rank“ je znám. Tuto funkci volají všechny procesy uvedené v MPI procesním teamu V poslední části programu je použita tato funkce k nalezení minima přes všechny „členy“ teamu.

Měření byla provedena na jednoprocesorovém systému, kdy počet tasků byl zadáván ručně. Jedná se tak pouze o příklad, jak by přibližně mohlo měření a implementace vypadat v praxi.

Výsledky měření uvedené v tabulce a zpracované grafem(matice souslednosti 240 x 240):

Pos	Počet procesů	Rozměry vážené matice sousednosti	Měření 1	Měření 2	Měření 3	Průměrné naměřené hodnoty
1	1	240 x 240	165,853	166,094	165,913	165,9533
2	2	240 x 240	21,3117	21,1464	21,4143	21,2908
3	4	240 x 240	2,83932	2,67683	2,94306	2,8197
4	6	240 x 240	0,8967	0,82055	1,02397	0,9137
5	8	240 x 240	0,359829	0,359826	0,359829	0,3598



5. Paralelizace pomocí OMP a MPI, Floyd-Warshallův algoritmus

Do výše uvedeného programu (projekt Parallel_MPI) byly přidány OMP direktivy # pragma omp parallel sharp a # pragma omp for collapse (3). Příslušné direktivy jsou okomentovány v kódu. Výsledky měření byly přibližně stejné jako u výše uvedené kapitoly 4. Měření byla provedena na jednoprocesorovém systému, kdy počet tasků byl zadáván ručně. Jedná se tak pouze o příklad, jak by přibližně mohlo měření a implementace vypadat v praxi.

6. Závěr

V této semestrální práci jsem se seznámil se základy paralelizace programů. Při práci jsem navazoval na semestrální práci z bakalářského studia, kdy jsem se rozhodl paralelizovat grafové algoritmy. Z grafů vyplývá, že paralelizace Dijkstrovova algoritmu je složitější a náročnější na čas se vzrůstajícím počtem vláken. Toto je způsobené poměrně malou, vstupní maticí vážené sousednosti. Dle různých studií z internetu vyplývá, že paralelizovat Dikstrovův algoritmus není snadné a vyplatí se až od velkého počtu uzlů (tisíce až desetitisíce uzlů). Když jsem se pokusil vygenerovat vstupní matici vážené sousednosti o rozměrech 1000 x 1000, program předčasně ukončil svojí činnost z důvodů nedostatku paměti na heapu. Paralelizace Floyd-Warshallova algoritmu vychází podle předpokladů.

Vypracování úlohy pro mě bylo přínosem a pomohlo mi lépe pochopit základy paralelního programování.