České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra řídicí techniky

Paralelní algoritmus pro rozvrhování činnosti přepravníku materiálu

Bakalářská práce

Michal SMOLA
28.5.2007

Poděkování

Děkuji Ing. Přemyslu Šůchovi za vedení a cenné rady v průběhu práce a rodičům a přítelkyni za podporu při studiu.

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu § 60 Zákona č.121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne _____ _____
podpis

Abstrakt

Cílem této práce je vyzkoušet dosud vytvořené sériové algoritmy pro rozvrhování činnosti přepravníku materiálu, návrh paralelního algoritmu řešícího tento problém a jeho implementace v jazyku C s využitím MPI (message passing interface). Součástí práce je i vyhodnocení zrychlení paralelního algoritmu oproti algoritmu sériovému.

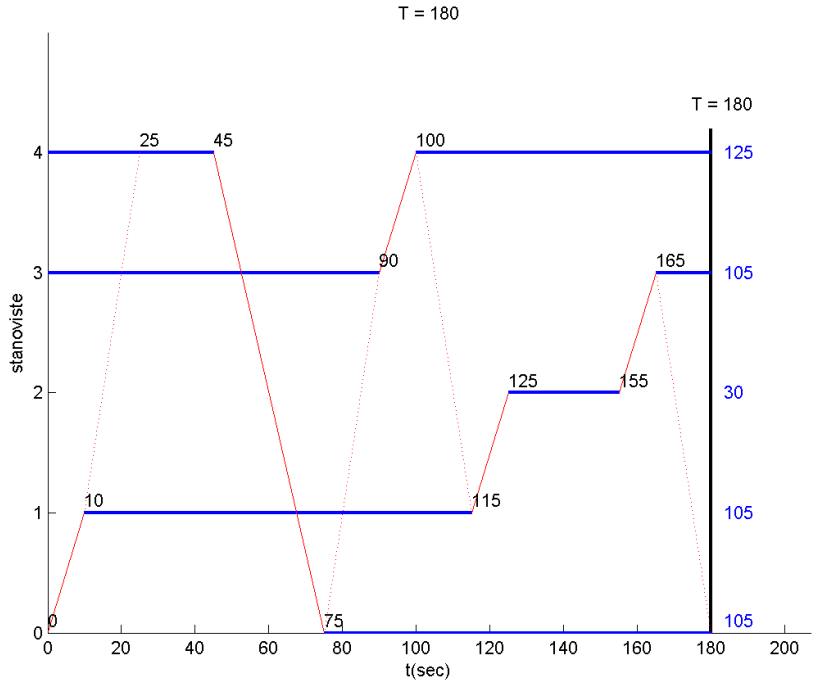
Abstract

The main aim of the work is to evaluate existing serial algorithms solving problem called hoist scheduling, develop parallel algorithm for this problem and implement it in C language using MPI (message passing interface). The study also compare speeds of serial and parallel algorithms.

Obsah

1 Úvod

V automatizovaném výrobním procesu je rozvrhování pohybu přepravníku materiálu klíčovým aspektem průchodnosti celého procesu a jeho efektivity. Může pomoci eliminovat vliv úzkých míst systému (anglický výraz pro tato místa je "bottleneck"). Doba úpravy materiálu v případě rozvrhování sofistikovaným algoritmem může být i několikanásobě kratší, než v případě rozvrhování laickým způsobem. Příklad pohybu přepravníku a materiálu po lince je zobrazen na obrázku 1 - výstup algoritmu MIP.



Obrázek 1: Ukázka řešení úlohy rozvrhování pomocí MIP algoritmu

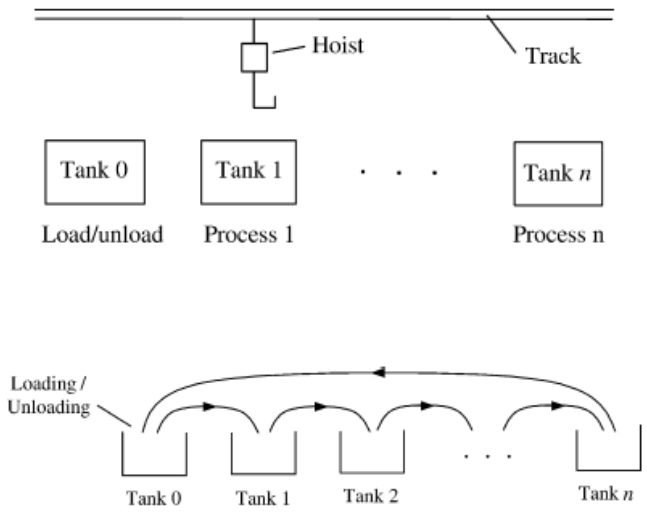
S rostoucím počtem stanovišť výrobního procesu rychle stoupají nároky na čas běhu algoritmů řešících rozvrhování. Jedná se totiž o NP úplnou úlohu, která není řešitelná v polynomiálním čase [4]. Jednou z možností, jak tento čas snížit jsou paralelní algoritmy.
V rámci práce jsem implementoval dva již existující algoritmy řešící rozvrhovaní pohybu přepravníku. Prvním z nich je algoritmus používající MIP (Mixed Integer Programming - smíšené celočíselné programování), který jako první definovali Philips a Unger [1] a později vylepšili o tzv. "slack times" a rozšířili Jiyin Liu s kolektivem [2]. Druhým algoritmem byl pak B&B algo-

ritmus (Branch and Bound - algoritmus větví a mezí), který byl publikován v [3]. Algoritmus B&B jsem upravil do paralelní podoby a taktéž implementoval. Od paralelního algoritmu jsem si sliboval zrychlení výpočtu s téměř lineární závislostí na počtu procesorů [6]. Rozsah práce bohužel neumožňoval blížší prozkoumání a případné využití některých nástrojů pro řešení optimalizačních problémů vyvinutých [7].

2 Popis problému

2.1 Základní úloha

Úkolem rozvrhování činnosti přepravníku materiálu je zlepšení průchodnosti (zefektivnění) výrobní linky. Ta se skládá z nakládacího stanoviště, kde materiál vstupuje do celého procesu, vykládacího stanoviště, kde materiál linku opouští, a z několika stanovišť, kde každé z nich provádí na materiálu nějakou úpravu. Někdy je stanoviště označováno jako nádrž, oba termíny budu používat zaměnitelně. V každém stanovišti se v jednom okamžiku může vyskytovat pouze jeden materiál. Příklad výrobní linky je zobrazen na obrázku 2



Obrázek 2: Ukázkové schema výrobní linky (obrázek převzat z [2])

Materiál musí projít v určeném pořadí všemi stanovišti a v každém z nich setrvat předem určenou dobu (doba je definována dolním a horním limitem pro trvání úprav materiálu). V praxi je časté, že nakládací a vykládací stanoviště je fyzicky stejné místo. V převzatých algoritmech i v mém paralelním algoritmu je toho předpokládáno. Pokud se s časem nemění parametry výrobního procesu (limity dob úprav materiálu v jednotlivých stanovištích), je pohyb přepravníku prováděn v cyklech. Zaručí se tak stejná kvalita všech produktů (stejné doby úprav materiálu). Úloha neztratí na obecnosti, když je začátek cyklu definován jako čas, kdy materiál opustí nakládací stanoviště.

2.2 Rozšíření

Pokud se mezi stanovišťi vyskytuje jedno nebo více časově náročných stanovišť, dochází zde ke zdržení a plýtvání časem. Proto se někdy tato časově náročná stanoviště duplikují a tím se zrychlí průchod materiálů linkou [2]. Druhé rozšíření základní úlohy se týká případu, kdy materiál navštíví jedno stanoviště vícekrát a může tak blokovat nádrž jinému materiálu [2]. Tato rozšíření jsem neuvažoval, uvádím je zde pouze pro úplnost.

2.3 Primitivní řešení úlohy

Takzvané *primitivní řešení* úlohy, kdy přepravník vždy při úpravách materiálu čeká nad stanovištěm na dokončení těchto úprav a přesun materiálu do další nádrže, může být za jistých okolností ideální. Pokud ale uvažujeme základní úlohu a nastane případ s jedním nebo více časově náročnými stanovišťi (relativně k ostatním), přepravník čekající na dokončení úprav v těchto nádržích je zbytečně nevyužitý. Primitivní řešení úlohy je použito v algoritmu branch and bound (větví a mezí) pro výpočet horní meze cyklu T.

2.4 Související práce

Jako první se vytvořením algoritmu řešícím problém rozvrhování činnosti přepravníku zabývali Phillips a Unger [1]. Jejich algoritmus využívá celočíselného programování a předpokládá, že doby přesunů materiálu z jednoho stanoviště na další jsou neměnné. V praxi je ale přesun materiálu ohraničen pouze minimální dobou, kterou přepravník pro přesun potřebuje a prodloužením této doby lze dosáhnout lepšího výsledku [2]. V roce 1988 Shapiro a Nuttle publikovali první algoritmus větví a mezí řešící rozvrhování přepravníku materiálu [11]. V roce 2002 publikovali Jiyin Liu, Yun Jiang a Zhili Zhou algoritmus využívající MIP, který neobsahoval předpoklad neměnných dob přesunů materiálů. Tento algoritmus pro základní problém vypočte optimální řešení.
Nepodařilo se mi najít žádnou práci řešící problém rozvrhování činnosti přepravníku paralelním algoritmem.
Paralelními algoritmy řešícími NP optimalizační problémy (do kterých úloha rozvrhování činnosti přepravníku patří) se zabývá práce Filipa Bláhy [7].

2.5 Přinos práce

Přínos práce spočívá v:

- implementaci algoritmů B&B a MIP
- porovnání rychlostí běhů a kvality výsledků těchto algoritmů
- návrhu a implementaci paralelního algoritmu
- ověření funkčnosti paralelního algoritmu a snižování doby jeho běhu s rostoucím počtem procesorů

V sekci 3 je popsána funkce algoritmu MIP, v sekci 3.3 je popsána funkce algoritmu B&B a v sekci 5 je popsán mnou navržený paralelní algoritmus. Experimentální výsledky z běhu všech tří algoritmů jsou uvedeny v sekci 6 a závěrenčné vyhodnocení experimentálních výsledků je uvedeno v sekci 6.3

3 Popis funkce algoritmu MIP

Úloha ILP (Integer Linear Programming) se od lineárního programování liší v tom, že proměnné mohou nabývat pouze celočíselných hodnot. Pokud některé proměnné mohou nabývat i neceločíselných hodnot, jedná se o úlohu MIP (Mixed Integer Programming). Nevýhodou ILP je složitost algoritmu řešícího tuto úlohu. Jedná se o NP úplnou úlohu, která není řešitelná v polynomiálním čase [4].
Algoritmus, který jsem implementoval, definovali Philips a Unger [1] a později vylepšili Jiyin Liu s kolektivem [2]. Jedná se o úlohu MIP. Algoritmus jsem implementoval v Matlabu a k vyřešení matice a vektoru pravých stran sestavených z rovnic uvedených v sekci 3.3 jsem použil nástroj GLPK [5], který je včleněn do Scheduling Toolboxu [10].

3.1 Parametry problému

K popisu problému použijeme následující značení:

- n = počet stanovišť mimo nakládacího a vykládacího stanoviště
- s_i = stanoviště číslo i (s_0 a s_{n+1} je nakládací/vykládací stanoviště), $i = 0, 1, 2, ..., n + 1$;
- a_i = minimální doba procesu ve stanovišti i, $i = 0, 1, 2, ..., n$;
- b_i = maximální doba procesu ve stanovišti i, $i = 0, 1, 2, ..., n$;
- d_i = minimální čas potřebný k přesunu nosiče z nádrže s_i do nádrže s_{i+1}, $i = 0, 1, 2, ..., n$;
- c_{ij} = doba nutná k přesunu nezatíženého přepravníku z nádrže i do nádrže j, $i, j = 0, 1, 2, ..., n + 1$;
- M = velká kladná konstanta

3.2 Rozhodovací proměnné

- T = doba trvání jednoho cyklu;
- t_i = čas počátku pohybu i (od začátku cyklu), $i = 1, 2, ..., n$.
  Pohyb i je definován jako pohyb ze stanoviště i do stanoviště $i+1$. Čas počátku pohybu 0 z nakládacího stanoviště je vždycky nula (odpovídá začátku cyklu);

- w_i = čas čekání před započetím pohybu $i = 0, 1, 2, ..., n$ (rozdíl mezi minimálním potřebným časem pro pohyb i a časem pro tento pohyb opravdu použitým);
- y_{ij} = binární proměnná, $y_{ij} = 1$ pokud pohyb i předchází pohybu j ($t_i < t_j$), jinak $y_{ij} = 0$. $i = 1, 2, ..., n-1, \quad j = 2, 3, ..., n; \quad i < j$

3.3 Popis problému

Se zavedeným značením je problém formulován takto:

$$Minimalizovat\ T \tag{1}$$

$$a_0 \leq T - t_n - d_n - w_n \leq b_0,\ pokud\ s_{n+1} = s_0 \tag{2}$$

$$d_i + a_{i+1} - M(1 - y_{i,i+1}) \leq t_{i+1} - t_i - w_i \leq d_i + b_{i+1} + M(1 - y_{i,i+1}),\ i = 0, 1, 2, ..., n-1 \tag{3}$$

$$d_i + a_{i+1} - My_{i,i+1} \leq t_{i+1} + T - t_i - w_i \leq d_i + b_{i+1} + My_{i,i+1},\ i = 1, 2, ..., n-1 \tag{4}$$

$$t_i \geq d_o + w_0 + c_{s_1,s_i},\ i = 1, 2, ..., n \tag{5}$$

$$t_j - t_i \geq d_i + w_i + c_{s_{i+1},s_j} - (1 - y_{ij})M,\ i = 1, 2, ..., n-1, j = 2, 3, ..., n;\ i < j \tag{6}$$

$$t_i - t_j \geq d_j + w_j + c_{s_{j+1},s_i} - y_{ij}M,\ i = 1, 2, ..., n-1;\ j = 2, 3, ..., n;\ i < j \tag{7}$$

$$T - t_i \geq d_i + w_i + c_{s_{i+1},s_0},\ i = 1, 2, ..., n \tag{8}$$

$$T \geq 0 \tag{9}$$

$$t_i \geq 0,\ w_i \geq 0,\ i = 1, 2, ..., n \tag{10}$$

$$y_{ij} \in \{0,1\},\ i,j = 1,2,...,n;\ i < j \tag{11}$$

Skupina omezení (9)-(11) zaručuje kladné hodnoty proměnných T, t_i a w_i a binární hodnoty u proměnné y_{ij}. Skupina omezení (2)-(4) zaručuje splnění časových limitů procesu v každé z nádrží. Omezení (2) zaručuje limity pro nakládací/vykládací stanoviště (s_0). Počáteční čas pohybu $i+1$ (a zároveň čas, kdy nosič opustí nádrž s_{i+1}) je t_{i+1}. Čas vstupu nosiče do nádrže s_{i+1} je $t_i + d_i + w_i$.
Pokud $t_i < t_{i+1}$, nosič vstoupí a opustí nádrž s_{i+1} ve stejném cyklu. Čas procesu ve stanovišti s_{i+1} je pak roven $t_{i+1} - (t_i + w_i + d_i)$. Omezení (3) v tomto případě zaručuje dodržení limitů délky procesu a omezeni (4) je splněno automaticky (díky $My_{i,i+1}$). V opačném případě, kdy $t_i > t_{i+1}$, nosič vstoupí do nádrže v jednom cyklu a až v následujícím ji opustí. Čas procesu v nádrži je pak $t_{i+1} + T - (t_i + w_i + d_i)$. Limity délky procesu v nádrži s_{i+1} pak zaručuje omezení (4) a automaticky je splněno omezení (3) (díky $M(1 - y_{i,i+1})$).
Další skupina omezení (5)-(8) zaručuje dostatek času pro přejezd prázdného přepravníku mezi jednotlivými přesuny nosičů. Pro dva pohyby i a j, pokud $t_i < t_j$ přepravník potřebuje čas nejméně $d_i + w_i + c_{s_{i+1},s_j}$ mezi začátky pohybů. V tomto případě $y_{ij} = 1$. Omezení (6) zaručuje splnění těchto požadavků a omezení (7) je splněno automaticky (díky $y_{ij} * M$). Jinak, pokud $t_i > t_j$, omezení (7) zaručuje dostatek času pro přejezd prázdného přepravníku a omezení (6) je splněno automaticky. Omezení (5) a (8) zaručují dostatek času mezi pohybem 0 a ostatními pohyby. Při pohybu i přepravník přejede z s_i do s_{i+1}, proto přejezd nezatíženého přepravníku mezi pohyby i a j je z nádrže s_{i+1} do nádrže s_j a čas potřebný pro přejezd je c_{s_{i+1},s_j}.

4 Popis funkce algoritmu větví a mezi (branch and bound algorithm)

Algoritmus větví a mezí odpovídá prohledávání stavového prostoru do hloubky. Pro větší efektivitu algoritmu se před začátkem vlastního větvení vypočítají meze pomáhající rozpoznat stavy, které nevedou k požadovanému výsledku. V průběhu větvení se tyto meze upravují v závislosti na průběžných výsledcích. Narozdíl od řešení přes MIP, tento algoritmus někdy nespočítá optimální řešení, protože neobsahuje takzvané "slack time" proměnné, tedy čekání přepravníku před započetím přesunu materiálu. Toto čekání v některých případech umožní jinou kombinaci přesunů a tím kratší dobu cylku.
Algoritmus, který jsem implementoval v rámci této práce a později upravil do paralelní podoby byl publikován v [3]. Mnou implementovaný algoritmus se mírně liší od publikovaného. Vzájemné rozdíly algoritmů:

- Publikovaný algoritmus při větvení upřednostňuje přesun materiálu, který má nejvyšší součet dob zbývajících úprav a přesunů. Můj algoritmus při větvení přesunuje materiály postupně podle jejich identifikačních čísel.
- Popsaný algoritmus vypočítává dolní mez cyklu třemi různými způsoby. Můj algoritmus způsobem jedním.
- Popsaný algoritmus po každém větvení zkoumá, zda výpočet může vést k lepšímu řešení, než je nejlepší dosud nalezené. Můj algoritmus pouze zkoumá, zda aktuální délka cyklu není delší, než nejkratší dosud vypočtená.

4.1 Vstupní parametry problému

K popisu zadání problému je použito následující značení:

- $n =$ počet stanovišť mimo nakládacího a vykládacího stanoviště
- $a_i =$ minimální doba úprav materiálu ve stanovišti i, $i = 0, 1, 2, ..., n$;
- $b_i =$ maximální doba úprav materiálu ve stanovišti i, $i = 0, 1, 2, ..., n$;
- $d_i =$ minimální čas potřebný k přesunu materiálu z nádrže s_i do nádrže s_{i+1}, $i = 0, 1, 2, ..., n$;
- $c_{ij} =$ čas trvání přejezdu nezatíženého přepravníku z nádrže i do nádrže j, $i, j = 0, 1, 2, ..., n + 1$;

4.2 Meze (bounds)

- $mMax$ = nejvyšši počet materiálů přesunutých přepravníkem v jednom cyklu
- $UBofT$ = horní mez doby trvání jednoho cyklu
- $LBofT$ = dolní mez doby trvání jednoho cyklu

4.3 Stavový prostor

Proměnné představující jeden stav (uzel větvení):

- m = počet materiálů přesunutých v jednom cyklu
- V_j = čísla nádrží, ve kterých se nachází materiál přes hranici cyklu, $j = 1, 2, ..., m$;
- $stav_i$ = pokud je nádrž i prázdná, pak $stav_i = 0$, jinak $stav_i$ odpovídá číslu materiálu, který v ní je, $i = 0, 1, 2, ..., n$;
- $pozHaku$ = číslo nádrže nad kterou se nachází přepravník
- t_i = čas počátku přesunu materiálu z nádrže i, $i = 0, 1, 2, ..., n$;
- $Tnow$ = čas uplynulý od začátku cyklu
- e_i = čas vstupu materiálu do nádrže i (konec pohybu $i - 1$) , $i = 1, 2, ..., (n + 1)$;

V kódu algoritmu se nacházejí ještě pomocné proměnné p a $pStart$, které vyjadřují skutečnou délku úpravy materiálu v příslušné nádrži.

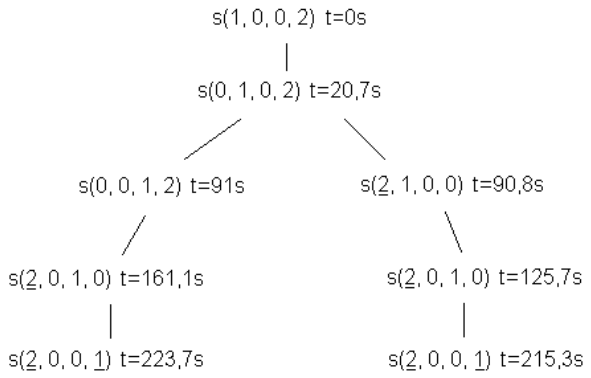
4.4 Algoritmus řešení

Algoritmus začíná výpočtem mezí. Dolní a horní mez pro délku cyklu a maximální počet meteriálu přesunutelný v jednom cyklu. Horní mez délky cyklu představuje primitivní řešení úlohy, viz sekce 2.3. Dolní mez se vypočte podle vztahu:

$$LB = \sum_{i=0}^{n} d_{i,i+1} + min_{j \neq i+1}[c_{i+1,j}], a_{i+1} \tag{12}$$

Následuje postupný výpočet vektorů V pro $m = 2$, inicializace datové struktury *solution* představující stav ve stavovém prostoru a její předání větvící části algoritmu (funkci `void BaB(solution Sol)`).

Tato část nejprve ověří, zda je obdržený uzel přípustný (nejsou překročeny horní meze dob úprav materiálu v jednotlivých nádržích a délka cyklu není větší, než vrchní mez této délky). Pokud je uzel přípustný, ověří se, zda už není výsledkem (zda jsou všechny materiály na místě kde mají být na konci cyklu a jejich úpravy jsou dokončeny). Pokud uzel představuje výsledek a délka cyklu tohoto výsledku je nejlepší z dosud nalezených, je řešení uloženo do globalní proměnné $SoluMin$. Pokud uzel není výsledkem, jsou nagenerováni jeho potomci (vždy přesunem jednoho materiálu do dalšího stanoviště) a předáni funkci `BaB(solution)`. Po propočítání všech vektorů pro všechna jednotlivá m algoritmus končí výpisem řešení s nejkratší dobou cyklu T.
Na obrázku 3 je schema znázorňující větvení pro jeden ukázkový případ. Počet nádrží $m = 3$, vektor obsahující čísla stanovišť, ve kterých se nachází



Obrázek 3: Ukázka větvení algoritmu B&B

materiály přes hranici cyklu, $V = \{0, 3\}$. Dříve nalezené řešení, odpovídající teď vrchní mezi cyklu $UBofT = 231,5s$. Čísla v závorce vyjadřují aktuální stav - rozmístění jednotlivých materiálů ve stanovištích (v pořadí stanovišť: 0, 1, 2, 3). Podtržení čísla vyjadřuje, že materiál se nachází v nádrži, ve které má být na konci cyklu (dál už se nepřesouvá). Proměnná t odpovídá času uplynulému od začátku cyklu.
První přesun je určený definicí začátku cyklu (cyklus začíná přesunem materiálu z nakládací/vykládací nádrže, viz sekce 2.1). Přesun materiálu 1 je ukončen v čase $t = 20,7s$. Následuje přesun materiálu 1 do nádrže 2. Ten je ukončen v čase $t = 161,1s$. Další pohyb materiálu 1 blokuje materiál 2. Větvení tedy pokračuje přesunem materiálu 2 do nakládací/vykládací nádrže, který je ukončen v čase $t = 223,7s$. Oba materiály jsou již ve stanovištích, ve kterých mají být na konci cyklu, proto algoritmus jen počká na

dokončení úprav obou materiálů. Výsledná doba je ale vyšší, než aktuální vrchní mez délky cyklu, proto toto řešení není uloženo. Druhá větev pokračuje od společného stavu s první větví přesunem materiálu 2 a následnými dvěma přesuny materiálu 1. Tento stav je již konečný a proto algoritmus opět počká na dokončení úprav obou materiálů. Výsledný čas trvání této větve je $t = 229,5$, což je výsledek o vteřinu lepší, než nejlepší dosud nalezený. Proto ho algoritmus uloží a pokračuje ve výpočtech s následujícím vektorem ($V = \{0, 2, 3\}$).

5 Paralelní algoritmus

Pro paralelní implementaci se z obou předchozích algoritmů více hodí algoritmus větví a mezí. Proto jsem pravil a implementoval ten. Využil jsem standardu MPI.

5.1 Message Passing Interface

MPI je knihovna sloužící k usnadnění tvorby paralelních programů (komunikace mezi procesory) a snadné přenositelnosti takto napsaných programů v jazyce C nebo Fortran 77. Tento standard se začal vyvíjet v roce 1992, nyní je již k dispozici verze 2.0. Standart specifikuje hlavně:

- dvoubodovou komunikaci (point to point communication)
- hromadnou komunikaci (collective communication)
- definuje skupiny, souvislosti, a komunikátory (groups, contexts and communicators)
- topologii procesů (process topologies)
- správu prostředí (MPI environmental management)

Ve verzi 2.0 pak dále přibylo:

- jednostranná komunikace (one-sided communications)
- rozšíření hromadné komunikace (extended collective operations)
- rozhraní jazyka C++
- vytváření procesů za chodu (dynamic processes)

Tento výčet není úplný. V rámci práce jsem MPI použil pouze pro vytvoření a rozlišení jednotlivých procesů a pro dvoubodovou komunikaci mezi nimi. Více informací lze najít v [8]

5.1.1 Dvoubodová komunikace v MPI

Příkazy dvoubodové komunikace použité v mém algoritmu:

- `MPI_Send(buf, count, datatype, dest, tag, comm)`
  Odeslání dat, význam parametrů:
  - `buf` : buffer (v C představuje pole), ve kterém jsou uložená data k odeslání
  - `count` : počet elementů bufferu (pole), které budou odeslány
  - `datatype` : datový typ odesílaných elementů
  - `dest` : číslo procesu, kterému jsou data odesílána
  - `tag` : příznak zprávy pro jejich vzájemné odlišení
  - `comm` : komunikátor (procesy jsou ve skupinách, každá má svůj komunikátor, tímto parametrem specifikujeme, ve které skupině se nachází přijímací proces)
- `MPI_Recv(buf, count, datatype, source, tag, comm, status)`
  Příjem dat, většina parametrů má jen obrácený význam oproti MPI_Send (místo bufferu pro odeslání je nyní buffer pro příjem, ...).
  - `source` : číslo procesu, od kterého očekáváme data
  - `status` : po přenosu obsahuje případný chybový kód
- `MPI_Sendrecv(sendbuf, sendcount, sendtype, dest, sendtag, recvbuf, recvcount, recvtype, source, recvtag, comm, status)`
  Kombinace obou předchozích příkazů, nejprve odešle data ze `sendbuf` a poté uloží přijatá do `recvbuf`
- `MPI_Iprobe(source, tag, comm, flag, status)`
  Příkaz slouží ke zjištění požadavku komunikace od procesu `source`. Návratová hodnota je uložena v proměnné `flag`. Ostatní parametry mají shodný význam jako v předešlých příkazech.

Všechny tyto příkazy mimo `MPI_Iprobe` jsou blokující. To znamená, že např. příkaz `MPI_Send` neskončí, dokud cílový proces zprávu nepřijme. Stejně tak příkaz `MPI_Recv` čeká, dokud mu zdrojový proces zprávu nepošle. Z toho důvodu bylo nutné v hlavním procesu (který komunikuje se všemi ostatními, viz sekce 5.2) použít příkaz `MPI_Iprobe`, protože jinak by čekal na komunikaci s jedním procesem a ostatní by museli čekat. Celý výpočet by se tak zastavil.

5.2 Popis algoritmu

V paralelních algoritmech bývá (časově) nejdražší komunikace. Jednak je náročný vlastní přenos zpráv a jednak čekání, až bude druhá strana ke komunikaci připravena. Proto jsem se rozhodl, že jeden procesor bude uchovávat nejlepší dosažené řešení a ostatní procesory budou komunikovat jenom s ním (bude zaneprázdněn pouze komunikací a ne vlastními výpočty). V případě napočítání vektorů `V` (vstupní parametr pro větvící funkci) jsem zvažoval dvě možnosti. První z nich, že všechny vektory `V` napočítá na začátku komunikační procesor (s identifikačním číslem 0) a rozešle je výpočetním procesorům, nebo si je napočítají výpočetní procesory sami a vyberou vektory jim náležející. Zvolil jsem druhou možnost, protože v prvním případě by při napočítávání vektorů komunikačním procesorem by ostatní stejně jen čekaly a trvání algoritmu by se prodloužilo i o komunikaci při distribuci těchto vektorů. Paralelní algoritmus tedy popíši z pohledu komunikačního procesoru a z pohledu výpočetního procesoru.

5.2.1 Popis funkce komunikačního procesoru

Komunikační procesor vypočítá vrchní mez cyklu `T` (primitivní řešení úlohy, viz sekce 2.3) a uloží si ji jako nejlepší doposud nalezené řešení. V případě, že mu výpočetní procesory nepředají žádné řešení, je toto řešení výsledné. Pak, dokud nedostane ode všech výpočetních procesorů zprávu, že dokončily výpočty, se jich postupně ptá, zda žádají o komunikaci. Pokud žádají, přijme zpávu a podle prvního znaku zprávy zjistí, co zpráva obsahuje. Pokud obsahuje vypočtené řešení a toto řešení je lepší než dosud uložené, uloží si ho. Pokud výpočetní proces žádá o zaslání aktuálního nejlepšího výsledku, odešle mu jej komunikační procesor zpět. Posledním typem zprávy je oznámení o dokončení výpočtů.

5.2.2 Popis funkce výpočetního procesoru

Výpočetní procesor si vypočítá obě meze cyklu `T`, maximální počet materiálů přesunutelných v jednom cyklu `mMax` a stejně jako v sériové verzi algoritmu všechny vektory `V`. Tentokrát je ale nejprve ukládá do pole vektorů. Podle svého identifikačního čísla pak procesor vybírá, které vektory mu přísluší a ty počítá. V průběhu výpočtů si pravidelně každých `REFR` větvení vyžádá od komunikačního procesoru aktuální nejkratší dobu cyklu, aby případně dříve poznal, že počítané řešení už nebude nejlepší (`REFR` je konstanta, kterou je možné nastavit četnost dotazování). Pokud vypočítá výsledek lepší, než je jeho lokálně známé nejlepší řešení, odešle ho komunikačnímu procesoru. Po

dokončení výpočtu všech vektorů odešle o tom komunikačnímu procesoru zpávu.

6 Experimentální výsledky

6.1 Testovací sestava počítačů

Algoritmus jsem testoval za pomoci vedoucího mé práce Ing. Šůchy. Nainstalovali jsme a nastavili prostředí MPICH2 (prostředí pro běh programů využívajících MPI, více v [9]) na čtyřech počítačích připojených k síti Internet pomocí Wi-Fi v případě notebooků a pomocí Ethernetu v případě desktopového počítače:

Tabulka 1: Procesory experimentálních počítačů

název	takt	typ	umístění
Intel P4	2.4Ghz	jednojádrový, hyperthreading	desktop
Intel Pentium M	2Ghz	jednojádrový	notebook
Intel Core Solo	1.86Ghz	jednojádrový	notebook
Intel Core 2	2Ghz	dvoujádrový	notebook

6.2 Doby běhů implementovaných algoritmů

6.2.1 Paralelní algoritmus B&B

Na této nehomogenní soustavě jsme měřili závislost dobu běhu algoritmu na počtu procesorů pro jedno pseudonáhodně vygenerované zadání s 53 stanovišti. Měřili jsme vždy jen jednou. Výpis programu běžícího na pěti procesorech je v příloze. Díky dvěma procesorům schopným počítat dvě úlohy souběžně jsme měli k dispozici 6 výpočetních jednotek. Počet procesorů jsme v prostředí MPICH2 nastavovali od 2, kdy jeden procesor počítá výsledky a posílá je druhému, až po 9, kdy počítají všechny procesory a nejrychlejší po skončení práce počítá ještě zadání, pro které už procesor nezbyl. Naměřené časy jsou v tabulce 2.

6.2.2 Sériový algoritmus B&B

Závislost doby běhu sériového algoritmu větví a mezí na počtu nádrží jsem měřil na svém počítači (notebook s procesorem Intel Core Solo, 1.86Ghz). V tomto případě už bylo zadání pokaždé jiné, měřil jsem proto vždy 50 běhů a z nich vypočítal aritmetický průměr. Začal jsem měřil od počtu 25 stanovišť, při nižším počtu byly časy velmi nízké (většinou neměřitelné). Naměřené časy jsou v tabulce 3.

Tabulka 2: Doby běhu paralelního algoritmu v závislosti na různém počtu procesorů

počet procesorů [-]	**doba běhu [s]**
2	98,6
3	54,5
4	30,9
5	38
6	26,6
7	35,4
8	21,3
9	27,6

Tabulka 3: Průměrné doby běhu sériového algoritmu větví a mezí pro různý počet stanovišť

počet nádrží [-]	**doba běhu [ms]**
25	257,5
26	264,5
27	634,5
28	1065,8
29	3567,9
30	4175,8
31	9753,7
32	13086,5

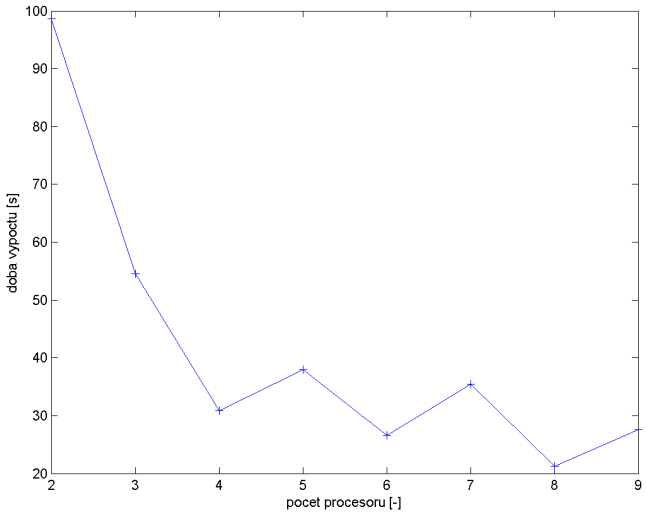
Tabulka 4: Průměrné doby běhu sériového algoritmu MIP pro různý počet stanovišť

počet nádrží [-]	doba běhu [s]
8	0,27
9	0,77
10	1,82
11	5,38
12	15,54
13	58,61
14	176,01

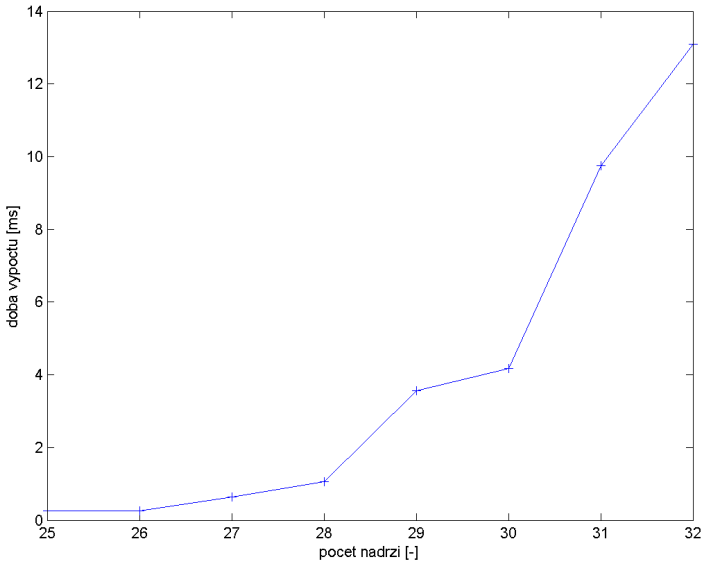
6.2.3 Sériový algoritmus MIP

Také jsem změřil závislost doby výpočtů algoritmu MIP (smýšené lineární programování) v závislosti na počtu nádrží. Generátor opět při každém běhu vytvořil unikátní zadání, proto bylo opět třeba změřit 50 běhů programu a z výsledků udělat aritmetický průměr. Naměřené časy jsou v tabulce 4.

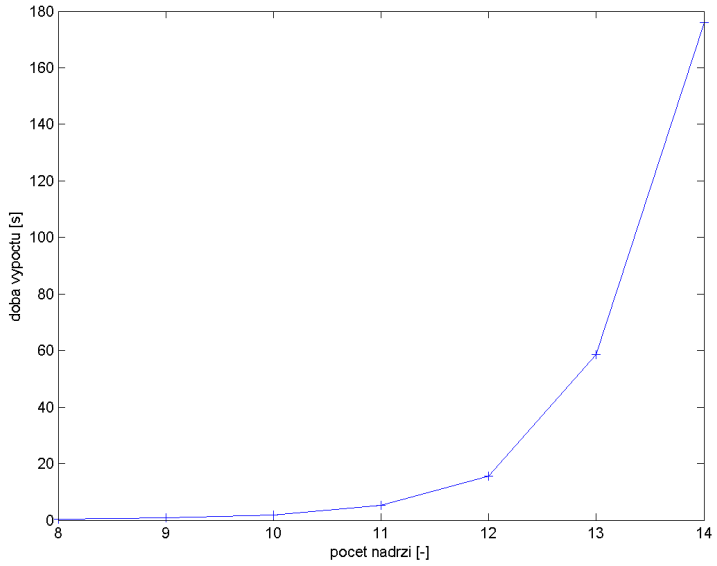
6.3 Grafy závislostí dob běhů algoritmů na počtu procesorů nebo na počtu nádrží



Obrázek 4: Závislost doby běhu programu na počtu procesorů - paralelní B&B algoritmus



Obrázek 5: Závislost průměrné doby běhu programu na počtu stanovišť v zadání - seriový B&B algoritmus



Obrázek 6: Závislost průměrné doby běhu programu na počtu stanovišť v zadání - seriový MIP algoritmus

7 Vyhodnocení experimentálních výsledků - Závěr

V grafu závislosti doby výpočtu na počtu procesorů pro paralelní algoritmus je vidět, že doba výpočtu se podle předpokladů snižuje. Kolísání grafu je způsobeno různorodostí procesorů na kterých bylo měření provedeno a různém rozložení výpočtů na ně.
Porovnání rychlostí obou sériových algoritmů není lehké, protože jsou oba částečně psány v jiném jazyku [1]. Z porovnání jejich grafů je ale vidět, že algoritmus MIP je časově náročnější. Jeho graf doby trvání výpočtu začíná prudce stoupat už při úlohách s více jak 13ti nádržemi, zatímco u algoritmu větví a mezí toho stoupání začíná až při zadáních s více jak 30ti nádržemi.
Předpokládal jsem, že paralelní algoritmus B&B bude pro zadání s malým počtem nádrží pomalejší, než sériový B&B, protože se projeví čas komunikace mezi procesory. Obě doby že se budou přibližovat a od určitého množství nádrží v zadání bude paralelní algoritmus rychlejší. Toto "určité množství" nemělo význam určit kvůli nesourodé sestavě testovacích počítačů a malému počtu procesorů (výsledky by neměly vypovídající hodnotu). To také neumožňovalo zjistit, jaké nastavení četnosti aktualizace nejlepšího dosud nalezeného výsledku ve výpočetních procesorech povede k nejrychlejšímu výpočtu výsledku konečného.
Úkol daný zadáním se mi podařilo splnit - navrhl a implementoval jsem funkční paralelní algoritmus pro rozvrhování činnosti přepravníku materiálu. Funkčnost algoritmu jsem ověřil na čtyřech strojích propojených přes síť Internet. Různorodost těchto strojů mi neumožnila naměřit data, ze kterých by se dalo usuzovat na zrychlení výpočtů oproti algoritmu sériovému. Algoritmus není optimální, úpravami (např. nalezením vhodné četnosti komunikace mezi hlavním procesorem a výpočetními procesory) by se jeho rychlost dala zvýšit.

[1] MIP algoritmus je sice psaný v kódu Matlabu, ale nástroj GLPK [5], který provádí vlastní výpočet, je psaný v jazyku C

Reference

[1] L. Phillips and P. Unger *Mathematical programming solution of a hoist scheduling problem, AIIE Transactions, 8(2):219–225, 1976*

[2] Jiyin Liu, Yun Jiang and Zhili Zhou *Cyclic scheduling of a single hoist in extended electroplating lines: a comprehensive integer programming solution, IIE Transactions 34, 905–914, 2002*

[3] W.C. NG *A Branch and Bound Algorithm for Hoist Scheduling of a Circuit Board Production Line, ISSN 0920-6299 (Print) 1572-9370 (Online), 1996*

[4] Přemysl Šůcha *Celočíselné lineární programování, http://dce.felk.cvut.cz/sucha/articles/ilp.pdf, 2004*

[5] A. Makhorin *GLPK (GNU Linear Programming Kit), http://www.gnu.org/software/glpk/*

[6] Ed Hall *pMatlab: Parallel Matlab Toolbox, http://www.itc.virginia.edu/research/talks/pmatlab2.pdf , 2006*

[7] Filip Bláha *Paralelní řešení NP optimalizačních problémů, 2007*

[8] *The Message Passing Interface (MPI) standard, http://www-unix.mcs.anl.gov/mpi/*

[9] *MPICH-A Portable Implementation of MPI, http://www-unix.mcs.anl.gov/mpi/mpich1/*

[10] *TORSCHE Scheduling Toolbox for Matlab, http://rtime.felk.cvut.cz/scheduling-toolbox/*

[11] W. Shapiro and W. Nuttle, G.W *Hoist scheduling for a PCB electroplating facility, IIE Transactions, 20(2), 157-167, 1988*

Příloha - výpis běhu paralelního algoritmu na pěti procesorech

```
1: Vypocet vsech vektoru V pro m=2 uspesne dokoncen
0: UBofT=3824.2
0: nProcNo=5
1: Vypocet vsech vektoru V pro m=3 uspesne dokoncen
1: Vypocet vsech vektoru V pro m=4 uspesne dokoncen
1: Vypocet vsech vektoru V pro m=5 uspesne dokoncen
1: Vypocet vsech vektoru uspesne dokoncen (494 vektoru celkem)
2: Vypocet vsech vektoru V pro m=2 uspesne dokoncen
2: Vypocet vsech vektoru V pro m=3 uspesne dokoncen
2: Vypocet vsech vektoru V pro m=4 uspesne dokoncen
2: Vypocet vsech vektoru V pro m=5 uspesne dokoncen
2: Vypocet vsech vektoru uspesne dokoncen (494 vektoru celkem)
4: Vypocet vsech vektoru V pro m=2 uspesne dokoncen
4: Vypocet vsech vektoru V pro m=3 uspesne dokoncen
4: Vypocet vsech vektoru V pro m=4 uspesne dokoncen
4: Vypocet vsech vektoru V pro m=5 uspesne dokoncen
4: Vypocet vsech vektoru uspesne dokoncen (494 vektoru celkem)
3: Vypocet vsech vektoru V pro m=2 uspesne dokoncen
3: Vypocet vsech vektoru V pro m=3 uspesne dokoncen
3: Vypocet vsech vektoru V pro m=4 uspesne dokoncen
3: Vypocet vsech vektoru V pro m=5 uspesne dokoncen
3: Vypocet vsech vektoru uspesne dokoncen (494 vektoru celkem)
0: Proces 4 zaslal vysledek Tin = 3166.9 (ulozene Tmin = 3824.2)
0: Proces 4 zaslal vysledek Tin = 3110.8 (ulozene Tmin = 3166.9)
0: Proces 4 zaslal vysledek Tin = 3082.9 (ulozene Tmin = 3110.8)
0: Proces 2 zaslal vysledek Tin = 3079.6 (ulozene Tmin = 3082.9)
0: Proces 2 zaslal vysledek Tin = 3056.7 (ulozene Tmin = 3079.6)
0: Proces 2 zaslal vysledek Tin = 3031.3 (ulozene Tmin = 3056.7)
0: Proces 4 zaslal vysledek Tin = 3078.5 (ulozene Tmin = 3031.3)
0: Proces 2 zaslal vysledek Tin = 3022.9 (ulozene Tmin = 3031.3)
0: Proces 2 zaslal vysledek Tin = 3015.3 (ulozene Tmin = 3022.9)
0: Proces 2 zaslal vysledek Tin = 3010.3 (ulozene Tmin = 3015.3)
0: Proces 2 zaslal vysledek Tin = 3002.7 (ulozene Tmin = 3010.3)
0: Proces 2 zaslal vysledek Tin = 2999.6 (ulozene Tmin = 3002.7)
0: Proces 2 zaslal vysledek Tin = 2998.4 (ulozene Tmin = 2999.6)
0: Proces 2 zaslal vysledek Tin = 2990.8 (ulozene Tmin = 2998.4)
0: Proces 2 zaslal vysledek Tin = 2983.6 (ulozene Tmin = 2990.8)
```

```
0: Proces 2 zaslal vysledek Tin = 2943.4 (ulozene Tmin = 2983.6)
0: Proces 2 zaslal vysledek Tin = 2935.5 (ulozene Tmin = 2943.4)
0: Proces 3 zaslal vysledek Tin = 3061.2 (ulozene Tmin = 2935.5)
0: Proces 3 zaslal vysledek Tin = 3033.3 (ulozene Tmin = 2935.5)
0: Proces 2 zaslal vysledek Tin = 2934.6 (ulozene Tmin = 2935.5)
0: Proces 3 zaslal vysledek Tin = 2922.3 (ulozene Tmin = 2934.6)
0: Proces 3 zaslal vysledek Tin = 2912.7 (ulozene Tmin = 2922.3)
0: Proces 3 zaslal vysledek Tin = 2897.4 (ulozene Tmin = 2912.7)
0: Proces 3 zaslal vysledek Tin = 2895.8 (ulozene Tmin = 2897.4)
0: Proces 3 zaslal vysledek Tin = 2880.5 (ulozene Tmin = 2895.8)
2: Vsechny vektory propocitany, koncim
0: Proces 2 dokoncil vypocty (finProcNo=1 of 5)
0: Proces 4 dokoncil vypocty (finProcNo=2 of 5)
4: Vsechny vektory propocitany, koncim
3: Vsechny vektory propocitany, koncim
0: Proces 3 dokoncil vypocty (finProcNo=3 of 5)
1: Vsechny vektory propocitany, koncim
0: Proces 1 dokoncil vypocty (finProcNo=4 of 5)
0: All processes were finished.

a[0]=20.0 a[1]=28.0 a[2]=44.6 a[3]=23.3 a[4]=26.8 a[5]=36.4 a[6]=48.7
a[7]=42.3 a[8]=22.5 a[9]=24.6 a[10]=46.9 a[11]=31.2 a[12]=43.1
a[13]=20.7 a[14]=21.5 a[15]=38.9 a[16]=47.1 a[17]=34.7 a[18]=26.9
a[19]=36.7 a[20]=35.0 a[21]=29.0 a[22]=39.8 a[23]=38.0 a[24]=49.5
a[25]=38.1 a[26]=48.9 a[27]=38.4 a[28]=25.4 a[29]=48.7 a[30]=25.2
a[31]=45.7 a[32]=29.8 a[33]=29.6 a[34]=42.0 a[35]=49.9 a[36]=22.1
a[37]=22.0 a[38]=34.5 a[39]=23.1 a[40]=33.7 a[41]=49.0 a[42]=40.4
a[43]=39.7 a[44]=48.2 a[45]=26.1 a[46]=49.4 a[47]=36.1 a[48]=31.4
a[49]=30.9 a[50]=33.3 a[51]=21.2 a[52]=21.5 a[53]=33.4

b[0]=120.8 b[1]=104.7 b[2]=147.0 b[3]=140.2 b[4]=146.0 b[5]=126.7
b[6]=155.4 b[7]=147.7 b[8]=131.9 b[9]=130.4 b[10]=160.8 b[11]=116.1
b[12]=126.6 b[13]=139.9 b[14]=102.6 b[15]=170.8 b[16]=171.0
b[17]=141.6 b[18]=153.0 b[19]=159.3 b[20]=148.6 b[21]=144.5
b[22]=116.6 b[23]=131.9 b[24]=145.0 b[25]=120.0 b[26]=150.5
b[27]=101.3 b[28]=118.7 b[29]=152.1 b[30]=104.3 b[31]=162.7
b[32]=140.6 b[33]=107.3 b[34]=180.0 b[35]=103.9 b[36]=153.8
b[37]=84.5 b[38]=152.2 b[39]=125.1 b[40]=154.3 b[41]=145.3
b[42]=148.1 b[43]=148.9 b[44]=160.4 b[45]=133.5 b[46]=137.2
b[47]=120.6 b[48]=153.0 b[49]=125.6 b[50]=141.6 b[51]=127.9
b[52]=75.4 b[53]=160.6
```

```
c[0][1]=29.3 c[1][2]=16.3 c[2][3]=19.3 c[3][4]=9.2 c[4][5]=19.0
c[5][6]=27.1 c[6][7]=14.9 c[7][8]=6.6 c[8][9]=25.7 c[9][10]=9.5
c[10][11]=27.1 c[11][12]=23.2 c[12][13]=33.6 c[13][14]=21.1
c[14][15]=24.0 c[15][16]=21.7 c[16][17]=32.5 c[17][18]=16.8
c[18][19]=30.2 c[19][20]=31.8 c[20][21]=21.0 c[21][22]=17.4
c[22][23]=5.9 c[23][24]=25.6 c[24][25]=27.8 c[25][26]=27.4
c[26][27]=14.3 c[27][28]=7.5 c[28][29]=34.2 c[29][30]=21.8
c[30][31]=29.3 c[31][32]=12.1 c[32][33]=29.6 c[33][34]=28.8
c[34][35]=9.6 c[35][36]=19.0 c[36][37]=21.7 c[37][38]=27.7
c[38][39]=12.9 c[39][40]=16.3 c[40][41]=25.1 c[41][42]=20.4
c[42][43]=16.8 c[43][44]=13.6 c[44][45]=28.3 c[45][46]=24.1
c[46][47]=28.1 c[47][48]=25.7 c[48][49]=23.7 c[49][50]=15.2
c[50][51]=28.2 c[51][52]=7.1 c[52][53]=10.4 c[53][0]=24.7

d[0]=44.3 d[1]=31.3 d[2]=34.3 d[3]=24.2 d[4]=34.0 d[5]=42.1
d[6]=29.9 d[7]=21.6 d[8]=40.7 d[9]=24.5 d[10]=42.1 d[11]=38.2
d[12]=48.6 d[13]=36.1 d[14]=39.0 d[15]=36.7 d[16]=47.5 d[17]=31.8
d[18]=45.2 d[19]=46.8 d[20]=36.0 d[21]=32.4 d[22]=20.9 d[23]=40.6
d[24]=42.8 d[25]=42.4 d[26]=29.3 d[27]=22.5 d[28]=49.2 d[29]=36.8
d[30]=44.3 d[31]=27.1 d[32]=44.6 d[33]=43.8 d[34]=24.6 d[35]=34.0
d[36]=36.7 d[37]=42.7 d[38]=27.9 d[39]=31.3 d[40]=40.1 d[41]=35.4
d[42]=31.8 d[43]=28.6 d[44]=43.3 d[45]=39.1 d[46]=43.1 d[47]=40.7
d[48]=38.7 d[49]=30.2 d[50]=43.2 d[51]=22.1 d[52]=25.4 d[53]=39.7

Nejkratsi vypoctena delka cyklu: Tmin=2880.5
Cas, kdy carrier opusti nadrz:
t[0]=0.0 t[1]=116.9 t[2]=210.4 t[3]=329.6 t[4]=380.7 t[5]=475.1
t[6]=614.5 t[7]=709.6 t[8]=753.7 t[9]=869.0 t[10]=974.4
t[11]=1102.6 t[12]=1218.4 t[13]=1320.2 t[14]=1377.7 t[15]=1478.1
t[16]=1575.7 t[17]=1674.2 t[18]=1733.0 t[19]=1836.3 t[20]=1927.6
t[21]=2102.6 t[22]=2202.9 t[23]=2287.6 t[24]=2449.1 t[25]=2595.8
t[26]=2708.1 t[27]=2832.7 t[28]=53.8 t[29]=162.7 t[30]=262.7
t[31]=423.1 t[32]=546.2 t[33]=649.6 t[34]=812.6 t[35]=910.3
t[36]=1039.8 t[37]=1146.2 t[38]=1285.2 t[39]=1431.9 t[40]=1519.9
t[41]=1632.5 t[42]=1783.8 t[43]=1890.5 t[44]=1969.4 t[45]=2038.8
t[46]=2148.5 t[47]=2229.4 t[48]=2341.4 t[49]=2411.0 t[50]=2498.6
t[51]=2563.0 t[52]=2653.2 t[53]=2769.8
Doba vypoctu = 38015.0 ms
```