ZÁPADOČESKÁ
UNIVERZITA
V PLZNI

Semestrální práce na předmět PPR

Výpočet hmotnosti kvádru

Jméno a příjmení: *Miroslav Exner*
Os. č.: *A02221*
Zadání: *08 (* 27. 11. 1982)*
Obor: *INI/IVT*
Email: *exac@exac.info*

Obsah

1 Zadání

Realizujte výpočet celkové hmotnosti zadaného kvádru prostoru $(0, 0, 0, x, y, z)$. Je dána funkce hustoty prostoru $\rho(x, y, z)$.

Povinné parametry:

- definice kvádru (x, y, z) (jeden roh kvádru se nachází v počátku souřadného systému)
- integrační krok $dx[, dy, dz]$ (může být stejný pro všechny osy)
- nějaký způsob vybírání funkcí hustoty (ne překompilováním programu!)

Povinné funkce hustoty:

$$\begin{aligned} \rho(x, y, z) &= 1 \\ \rho(x, y, z) &= x \\ \rho(x, y, z) &= x + y + z \end{aligned}$$

Vzorec pro výpočet hmotnosti:

$$\begin{aligned} M &= \sum dM \\ dM(x, y, z) &= dx \cdot dy \cdot dz \cdot \rho\left(x + \frac{dx}{2}, y + \frac{dy}{2}, z + \frac{dz}{2}\right) \end{aligned}$$

Úlohu zpracujte jako:

1. Paralelní program pro systém se sdílenou pamětí (použitelné prostředky: prog. jazyk Java nebo vlákna POSIX).
2. Paralelní program pro systém s distribuovanou pamětí (použitelné prostředky: PVM nebo MPI) .

2 Stručná analýza

Nejprve si musíme uvědomit, co bude aplikace ke svému běhu potřebovat za vstupní hodnoty, jak ji můžeme logicky rozdělit na několik kooperujících částí, jak samotný výpočet bude probíhat a co se na konci aplikace vlastně dovzvíme.

2.1 Vstupy

Vstupních hodnot budou oba dva programy potřebovat celkem sedm. Čtyři čísla v plovoucí čárce, *x, y, z* pro rozměr počítaného kvádru a integrační krok *d* výpočtu hmotnosti[1] a tři čísla celá *druh funkce* ρ[2], *počet vláken/dělníků* a *velikost přidělované práce* v procentech[3].

2.2 Výstupy

V podstatě jediné, co nám má aplikace říci, je hmotnost zadaného kvádru. Tento údaj můžeme doplnit jednoduchou statistikou např. dobou běhu programu.

2.3 Rozdělení na třídy/moduly

Celou úlohu si rozdělíme do několik samostatných částí (tříd resp. modulů).

2.3.1 Manažer

Vykonává činnost manažera/šéfa práce tzn. vytváří vlákna/dělníky, zadává jim jednotlivé díly práce, čeká na vrácení vypočítaných hodnot a opětovně přídeluje práci, nebo vydá pokyn k ukončení činnosti vlákna/dělníka. Průběžně sčítá všechny výsledky a rozdělení a spočítání veškeré práce ukončí vlákna (zašle pokyn k ukončení) a vypíše výsledek.

2.3.2 Samotný výpočet

Tato část programu, bude vykonávat samotný výpočet hmotnosti kvádru. Čili obdrží od manažera určitou část počítaného kvádru, integrační krok a příznak, kterou funkci ρ použít (viz zadání). Hlavní částí bude smyčka zpráv, která zachytává zprávy od manažera a podle toho spouští výpočet, nebo ukončí činnost.

2.3.3 Funkce ρ

Sekce, která pouze ze zadaných parametrů, což jsou hodnoty x, y, z a druh funkce vrací příslušnou funkční hodnotu. Mohla by být součástí vlákna/dělníka, ale pro přehlednost ji oddělíme.

2.4 Výpočet dílčí hmotnosti

Výpočet dílčí hmotnost kvádru probíhá podle rovnice (viz Zadání). Jedná se o tři do sebe zanořené for-cykly, které zajišťují průchod zadaným kvádrem (nebo jeho částí) vždy o daný integrační krok a výpočet hmotnost daného integračního kroku. Tato hodnota se přičítá k

[1] Integrační krok bude stejný pro všechny osy

[2] hodnoty 1, 2 a 3

[3] Když je zde hodnota 100, tak se celá práce „navalí“ na jedno vlákno

výsledné hmotnosti. Po ukončení všech cyklů známe výslednou hmotnost (její přibližnou hodnotu).

2.5 Přidělování práce

Jakmile řídící modul zjistí, že některé z vláken/dělníků nepracuje, zjistí jestli přidělená práce je menší než 100%, čili ještě je práce k přidělování. Pokud je již všechna práce rozdělená, vydá pokyn k ukončení vlákna resp. nechá dělníka čekat na zprávu (blokující čekání), aby zbytečně neplýtvalo systémovými prostředky. Pokud je práce k přidělování, vypočítá se příslušný díl kvádru a všechny potřebné parametry se předají vláknu/dělníkovi.

2.6 Příklad průběhu výpočtu

Nástíníme si schematický průběh výpočtu manažera i dělníka.

2.6.1 Manažer

Obdržení vstupních hodnoty (z příkazové řádky, od uživatele . . .)

Vytvoří všechny dělníky

Začátek smyčky zpráv

Čekajícím dělníkům rozdá dostupnou práci

Čeká na výsledky

Obdrží zprávu s výsledkem, který příčte k celkovému výsledku

Zadá dělníkovi další práci (je-li dostupná)

Obdrží zprvu s výsledkem od dalšího dělníka

Práce už není, řekne dělníkovi, aby se ukončil

Opakuje smyčku od začátku, dokud dělníci pracují

Konec smyčky, výpis výsledků, konec

2.6.2 Dělník

Dělník je vytvořen

Začátek smyčky zpráv

Obdrží zprávu s parametry výpočtu

Provede výpočet

Odešle výsledky manažerovi

Čeká na nove parametry výpočtu, nebo na pokyn k ukončení

3 Implementace

3.1 Systém se sdílenou pamětí

Pro zpracování úlohy v systému se sdílenou pamětí jsem volil programovací jazyk Java, na konkrétně verzi 1.5.0_06. Vývoj probíhal v IDE Eclipse 3.2 na operačním systému Microsoft Windows XP home.

3.2 Systém s distribuovanou pamětí

Pro zpracování úlohy v systému s distribuovanou jsem volil prostředí PVM a programovací jazyk C. Vývoj probíhal v editory PSPad na 4.5.2 na operačním systému Microsoft Windows XP home. Program byl kompilován pomocí překladače GCC na stroji Jumbo.

3.2.1 Překlad

Ke zdrojovým kódům je připojem soubor `makefile`, který pomůže s překladem. Jednoduchým zavoláním příkazu `make` přeložíte řídící modulu resp. dělníka takto:

```
make EXE=exac-main
make EXE=exac-worker
```

Získáte soubory `exac-main.exe` a `exac-worker.exe`.

4 Uživatelský manuál

4.1 Java

Samotná aplikace se nachází v souboru `ppr-08.jar`, který samozřejmě můžeme spustit pomocí:

```
java -jar ppr-08.jar
```

Pro ulehčení je k dispozici dávkový soubor `runme.bat`, který výše uvedený příkaz vykoná sám.

Dále musíme programu zadat všechny parametry výpočtu (viz Analýza). Jedná se celkem o sedm hodnot a to v tomto pořadí rozměry kvádru x, y, z, *integrační krok*, *druh funkce* ρ, *počet vláken* a *velikost přidělované práce* v procentech. Příklad:

```
runme 20.1 10.5 1.3 0.01 2 5 10
```

Čili rozměr kvádru je 20.1 x 10.5 x 1.3, integrační krok je 0.01, jako funkce hustoty se použije $\rho(x, y, z) = x$, počítat bude celkem 5 vláken a každé z nich bude dostávat 10% celkového rozsahu práce.

4.2 PMV

Tato aplikace se skládá ze dvou spustitelných souborů. Důležitejším - a který chceme spouštět - je soubor `exac-main.exe`. Obsahuje řídící část výpočtu. Druhý soubor `exac-worker.exe` slouží k samotnému výpočtu a je volán řídícím modulem. Samostatné spouštění tohoto souboru pozbývá jakéhokoliv smyslu.

K samotnému spuštění výpočtu potřebujeme mít spuštěné prostředí PVM, přesněji démona PVM[4].

Hlavnímu programu musíme stejně jako programu v Javě zadat všechny parametry výpočtu (viz Analýza). Jedná se celkem o sedm hodnot a to v tomto pořadí rozměry kvádru x, y, z, *integrační krok*, *druh funkce* ρ, *počet vláken* a *velikost přidělované práce* v procentech. Příklad:

```
./exac-main.exe 20.1 10.5 1.3 0.01 2 5 10
```

Čili rozměr kvádru je 20.1 x 10.5 x 1.3, integrační krok je 0.01, jako funkce hustoty se použije $\rho(x, y, z) = x$, počítat bude celkem 5 vláken a každé z nich bude dostávat 10% celkového rozsahu práce.

Tyto soubory byly vytvořeny překladem zdrojových kódů na stroji Jumbo, tudíž doporučuji spouštět je tamtéž.

5 Závěr

Zadání bylo splněno oba programu fungují a vracejí správné výsledky. Nicméně jsou zatíženy chybou způsobenou zobrazováním čísel v počítač - při rozdělování kvádru na části pro jednotlivá vlákna, rozdělování kvádru po integračních krocích atd.

V Javě by se dal problém vyřešit použitím třídy `BigDecimal`[5], která by přesnost výpočtu velmi zlepšila. Nicméně její použití je trochu „přes ruku“, protože není možné použít ji v matematických výrazech a matematické operace s ní trvají v průměru 15 krát déle.

V jazyku C by se na ukládání čísel v plovoucí řádce mohl použít typ `long double`, ale bohužel s tímto typem nemám žádné zkušenosti a byl by problém, jak tento typ kódovat do zpráv PVM. V dokumentaci jsem objevil, že umí do zprávu jednoduše přidat pouze `double` obyčejný.

[4] viz webové stránky předmětu PPR

[5] Používá se např. k peněžním výpočtům, kde i drobná odchylka může způsobit velmi rozdílné hodnoty úroků apod.

6 Použitá literatura

- **Stanislav Racek** Skripta k předmětu PPR
- Materiály a příklady na webu předmětu PPR
- **Pavel Herout** (2003) Učebnice jazyka Java, Kopp
- **Pavel Herout** (2001) Učebnice jazyka C, Kopp
- **David Brackeen** (2004) Vývoj her v jazyku Java, Grada

A Java

A.1 Třída ThreadManager

```
package info.exac.ppr;

import java.util.ArrayList;
import java.util.GregorianCalendar;

public class ThreadManager extends Thread {

/* Parametry ulohy */
private int pocetVlaken;
private int velikostPrace;
private double d;
private int jakaFunkce;
private Bod3D kvadr = null;


private ArrayList<HmotnostThread> vlakna = null;

private double hmotnost = 0;

private int pridelenaPrace = 0;

private boolean hotovo = false;

private long startTime = 0;
private long endTime = 0;


public ThreadManager(Bod3D kvadr, double d, int jakaFunkce,
int pocetVlaken, int velikostPrace) {

super();
this.kvadr = kvadr;
this.d = d;
this.jakaFunkce = jakaFunkce;
this.pocetVlaken = pocetVlaken;
this.velikostPrace = velikostPrace;

this.vlakna = new ArrayList<HmotnostThread>();
```

```
vytvorVlakna();

}



private void vytvorVlakna() {
for (int i = 0; i < pocetVlaken; i++) {
vlakna.add(new HmotnostThread());
}
}



public void vypis() {

/* Vypise stav vlaken */
System.out.println("ThreadManager:\trozdeleno " + pridelenaPrace + "%");
for (int i = 0; i < pocetVlaken; i++) {
vlakna.get(i).vytiskni();
}
System.out.println();
}



private synchronized void pridelPraciVlaknu(int index) {

HmotnostThread vlakno = vlakna.get(index);


/* Je vubec co pridelovat ? */
if (pridelenaPrace < 100) {


if (pridelenaPrace + velikostPrace < 100) {
vlakno.nastavVypocet((double)pridelenaPrace/100 *kvadr.x, 0, 0,
(double)(pridelenaPrace+velikostPrace)/100 *kvadr.x, kvadr.y,
kvadr.z, d, jakaFunkce);
vlakno.vypocti();
pridelenaPrace += velikostPrace;
} else {
vlakno.nastavVypocet( (double)pridelenaPrace /100 *kvadr.x, 0, 0,
```

```
kvadr.x , kvadr.y, kvadr.z, d, jakaFunkce);
vlakno.vypocti();
pridelenaPrace = 100;
}


} else {
vlakno.umri();
}
}


public void run() {

startTime = new GregorianCalendar().getTimeInMillis();

/* Odstartuje vlakna */
for (int i = 0; i < pocetVlaken; i++) {
vlakna.get(i).start();
}



boolean ukoncitSmycku = false;
while (!ukoncitSmycku) {

ukoncitSmycku = true;
for (int i = 0; i < pocetVlaken; i++) {

HmotnostThread vlakno = vlakna.get(i);



if (vlakno.getStav() == HmotnostThread.STAV_VYTVORENE) {
pridelPraciVlaknu(i);
}

if (vlakno.getStav() == HmotnostThread.STAV_VYPOCTENE) {
hmotnost += vlakna.get(i).getHmotnost();
pridelPraciVlaknu(i);
}
```

```
if (vlakno.getStav() != HmotnostThread.STAV_UMIRAJICI) {
ukoncitSmycku = false;
}
}

}

endTime = new GregorianCalendar().getTimeInMillis();
this.hotovo = true;
}


public boolean isHotovo() {
return hotovo;
}


public double getHmotnost() {
return hmotnost;
}


public long getCasVypoctu() {
return ( (endTime - startTime) > 0) ? endTime - startTime : 0;
}

}
```

A.2 Třída HmotnostThread

```
package info.exac.ppr;


public class HmotnostThread extends Thread {

public static boolean DEBUG = false;

public static int STAV_VYTVORENE = 1;
public static int STAV_NASTAVENE = 2;
public static int STAV_POCITAJICI = 3;
public static int STAV_VYPOCTENE = 4;
public static int STAV_UMIRAJICI = 5;
```

```
private int stav = 0;
private int prubeh = 0;
private long iterace = 0;

private double startX = 0;
private double startY = 0;
private double startZ = 0;

private double konecX = 0;
private double konecY = 0;
private double konecZ = 0;

private double d = 0;

private FunkceRo funkceRo = null;

private double hmotnost = 0;




public HmotnostThread() {
super();
stav = STAV_VYTVORENE;
if (DEBUG) System.out.println("Vlakno " + getId() + ": Vytvoreno");
}


public void setDebugMode() {
DEBUG = true;
}

public void setNormalMode() {
DEBUG = false;
}

public void nastavVypocet(double startX, double startY, double startZ,
double konecX, double konecY, double konecZ, double d, int jakaFunkce) {

this.startX = startX;
this.startY = startY;
```

```
this.startZ = startZ;

this.konecX = konecX;
this.konecY = konecY;
this.konecZ = konecZ;

this.d = d;

this.funkceRo = new FunkceRo(jakaFunkce);

this.hmotnost = 0;

this.stav = STAV_NASTAVENE;

if (DEBUG) System.out.println("Vlakno " + getId() + ": Nastaveno");
}



public void vypocti() {
stav = STAV_POCITAJICI;
prubeh = 0;
}



public void umri() {
if (DEBUG) System.out.println("Vlakno " + getId() + ": Zabijim se");
stav = STAV_UMIRAJICI;
}



public double getHmotnost() {
return hmotnost;
}


public int getStav() {
return stav;
}

public int getPrubeh() {
```

```
return prubeh;
}



public void vytiskni() {

System.out.println("Vlakno " + getId() + ":\tprubeh " + prubeh + "%\t m = " +
hmotnost + "\t" + getState() + "(" + getStav() + ")");
}



private void vypocet() {

iterace = 0;

if (DEBUG) System.out.println("Vlakno " + getId() + ": Zacatek vypoctu");

for (double x = startX; x < konecX; x += d ) {
for (double y = startY; y < konecY; y += d ) {
for (double z = startZ; z < konecZ; z += d ) {

double dM = d * d * d * funkceRo.getRo(x + d/2, y + d/2, z + d/2);
this.hmotnost += dM;

iterace++; }
}
prubeh = (int) (  (x - startX) / ((konecX - startX) / 100)  ) ;
}

prubeh = 100;
stav = STAV_VYPOCTENE;
if (DEBUG) System.out.println("Vlakno " + getId() + ":
Konec vypoctu (iteraci " + iterace +")");
}



public void run() {

if (DEBUG) System.out.println("Vlakno " + getId() + ": Start pracovni smycky");

/* Temer nekonecna smycka */
```

```
while (stav != STAV_UMIRAJICI) {

if (stav == STAV_POCITAJICI) {
vypocet();
}

}
// System.out.println( this.getId() + ": Umiram");

if (DEBUG) System.out.println("Vlakno " + getId() + ":
Konec smycky, konec vlakna");
}

}
```

A.3 Třída FunkceRo

```
package info.exac.ppr;

public class FunkceRo {

public static int RO_KONSTANTNI_1 = 1;
public static int RO_PROMENNA_X = 2;
public static int RO_PROMENNA_XYZ = 3;


private int druhFunkce = 0;




public FunkceRo(int druhFunkce) {
super();

this.druhFunkce = druhFunkce;
}




public double getRo(double x, double y, double z) {
```

```
double ro = 0;

switch (this.druhFunkce) {
case 1:
ro = 1;
break;
case 2:
ro = x;
break;
case 3:
ro = x + y + z;
break;
default:
break;
}


return ro;
}
}
```

B PVM

B.1 Modul exac-main.c

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <pvm3.h>

#include "exac-def.h"
#include "exac-ro.h"



T_BOD3D kvadr;            /* Rozmery kvadru, ktery chceme spocitat */
int velikost_prace;       /* kolik prace se bude workerovi pridelovat v % */
int zadana_prace = 0;     /* Zadany podil prace */
int druh_funkce;          /* druh funkce Ro */
double d;                 /* integracni krok pro vypocet */
int pracujicich_workeru = 0;    /* Kolik delniku pracuje */


int mytid;          /* identifikatro procesu */




void log_start() {
    FILE *fout;
    fout = fopen(LOG_FILE, "w");
    fclose(fout);
}




void pridel_praci(int tid)
{

    if (zadana_prace < 100) {
        printf("Hlavni %d: Prideluji praci delnikovi %d,
prideleno %d\%\n", mytid, tid, zadana_prace);
```

```
        if ( (zadana_prace + velikost_prace) < 100 ) {

            double start_x = kvadr.x / (double) 100 * (double) zadana_prace;
            double start_y = 0.0;
            double start_z = 0.0;

            double konec_x = kvadr.x / (double) 100 * (double)
(zadana_prace + velikost_prace);
            double konec_y = kvadr.y;
            double konec_z = kvadr.z;

            /* Zaslani prace workerovi */
            pvm_initsend(PvmDataDefault);   /* inicializace bufferu */
            pvm_pkdouble(&start_x, 1, 1);
            pvm_pkdouble(&start_y, 1, 1);
            pvm_pkdouble(&start_z, 1, 1);
            pvm_pkdouble(&konec_x, 1, 1);
            pvm_pkdouble(&konec_y, 1, 1);
            pvm_pkdouble(&konec_z, 1, 1);
            pvm_pkdouble(&d, 1, 1);
            pvm_pkint(&druh_funkce, 1, 1);
            pvm_send(tid, MSG_PRACE);

            zadana_prace += velikost_prace;
        } else if ( (zadana_prace + velikost_prace) >= 100 ) {

            double start_x = kvadr.x / (double) 100 * (double) zadana_prace;
            double start_y = 0.0;
            double start_z = 0.0;

            double konec_x = kvadr.x;
            double konec_y = kvadr.y;
            double konec_z = kvadr.z;

            /* Zaslani prace workerovi */
            pvm_initsend(PvmDataDefault);   /* inicializace bufferu */
            pvm_pkdouble(&start_x, 1, 1);
            pvm_pkdouble(&start_y, 1, 1);
            pvm_pkdouble(&start_z, 1, 1);
            pvm_pkdouble(&konec_x, 1, 1);
            pvm_pkdouble(&konec_y, 1, 1);
            pvm_pkdouble(&konec_z, 1, 1);
```

```
            pvm_pkdouble(&d, 1, 1);
            pvm_pkint(&druh_funkce, 1, 1);
            pvm_send(tid, MSG_PRACE);


            zadana_prace = 100;
        }
        pracujicich_workeru++;
    }
}




int main(int argc, char *argv[])
{

    log_start();

    /* Parametry vypoctu */
    int pocet_workeru;        /* Pocet workeru */
    double hmotnost = 0;      /* vysledna hmotnost - vystup */


    /* Prommenne programu */
    clock_t start_time, end_time;   /* Casove udaje pro zmereni
doby behu programu */

    int tids[32];       /*  identifikatory procesu delnici */
    int* tidc;          /* pomocny ukazatel pro vytvareni delniku */
    int vytvoreno_workeru = 0;      /* pocet doposud vytvorenych workeru */
    int tid;            /* pomocny identifikator procesu */

    double dM;

    int i;

    if (argc == 8) {
        kvadr.x = atof(argv[1]);
        kvadr.y = atof(argv[2]);
        kvadr.z = atof(argv[3]);
        d = atof(argv[4]);
        sscanf(argv[5], "%d", &druh_funkce);
```

```
        sscanf(argv[6], "%d", &pocet_workeru);
        sscanf(argv[7], "%d", &velikost_prace);
    } else {
printf("\nIntegralni vypocet hmotnosti kvadru\n");
printf("===================================\n\n");
printf("Semestralni prace na PPR\n");
printf("Miroslav Exner\tA02221\n");
printf("exac@exac.info\twww.exac.info\n\n");
printf("Musite zadat celkem 7 argumentu prikazove radky. Musite zadat\n\n");
printf("exac-main.exe <x> <y> <z> <d> <c. funkce> <pocet vlaken>
<velikost prace>\n");
printf("   <x> <y> <z>      - rozmery kvadru\n");
printf("   <d>              - integracni krok pro vsechny osy\n");
printf("   <c. funkce>      - cislo funkce hustoty 1, 2, nebo 3\n");
printf("                         1 - ro(x, y, z) = 1\n");
printf("                         2 - ro(x, y, z) = x\n");
printf("                         3 - ro(x, y, z) = x+y+z\n");
printf("   <pocet vlaken>   - pocet vlaken, do kterych rozdelujeme ulohu\n");
printf("   <velikost prace> - v % dil pridelovane prace vlaknu\n\n");
printf("napr.:\n");
printf("exac-main.exe 20 10 1 0.005 1 5 5\n\n");
return ;

    }

    printf("\nIntegralni vypocet hmotnosti kvadru\n");
printf("===================================\n\n");
printf("Semestralni prace na PPR\n");
printf("Miroslav Exner\tA02221\n");
printf("exac@exac.info\twww.exac.info\n\n");
    printf("-----------------------------------------------------------------\n");
    printf("Rozmery kvadry <x, y, z>: %6.2f %6.2f %6.2f\n",
kvadr.x, kvadr.y, kvadr.z);
    printf("Integracni krok: %6.8f\n", d);
    printf("Druh funkce: ");
    switch (druh_funkce) {
        case 1 :
            printf("ro(x, y, z) = 1\n");
            break;
        case 2 :
            printf("ro(x, y, z) = x\n");
            break;
        case 3 :
```

```
            printf("ro(x, y, z) = x + y + z\n");
            break;
        default :
            break;
    }
    printf("Pocet delniku: %d\n", pocet_workeru);
    printf("Rozsah pridelovane prace: %d\%\n", velikost_prace);
    printf("----------------------------------------------------------------\n");


    start_time = clock();

    mytid = pvm_mytid();
    printf("Hlavni %d: Prihlasen do PVM\n", mytid);



    /* vytvoreni delniku */
    tidc = tids;
    while ( pocet_workeru > vytvoreno_workeru) {
        if (pvm_spawn("exac-worker.exe", (char**)0, 0, "", 1, tidc)) {
            printf("Hlavni %d: Vytvoren delnik %d.\n", mytid, *tidc);
            vytvoreno_workeru++;
            tidc++;
        } else {
            printf("Hlavni %d: Chyba pri vytvareni delnika. Cislo chyby %d\n",
mytid, *tidc);
        }
    }



    /* Rekneme delnikum kdo je tu sef */
    pvm_initsend(PvmDataDefault);   /* inicializace bufferu */
    pvm_pkint(&mytid, 1, 1);    /* ulozeni identifikace */
    pvm_mcast(tids, pocet_workeru, MSG_REGISTRACE);
/* broadcast zpravy s kodem 4 */



    /* cekani na odezvu od delniku */
    for(i = 0; i < pocet_workeru; i++) {
        pvm_recv(-1, MSG_PRIPRAVEN);        /* kod PRIPRAVEN od delniku*/
```

```
        pvm_upkint(&tid, 1, 1);             /* id delnika */
        //pvm_upkint(&worker_cnt, 1, 1);      /* pocet zpracovanych
kousku prace delnika */
        printf("Hlavni %d: Delnik cislo %d k praci pripraven\n", mytid, tid);
    }



    /* Prideleni prace vsem workerum na pocatku */
    for (i = 0; i < pocet_workeru; i++) {
        pridel_praci(tids[i]);
    }



    printf("Hlavni %d: Spoustim smycku zprav\n", mytid);
    while (1) {

        if (pvm_nrecv(-1, MSG_VYSLEDEK) > 0) {

            /* Vyhodnoceni zaslaneho vysledku */
            pvm_upkint(&tid, 1, 1);
            pvm_upkdouble(&dM, 1, 1);
            printf("Hlavni %d: Parcialni hmotnost z delnika %d: %6.10f\n",
mytid, tid, dM);
            hmotnost += dM;
            pracujicich_workeru--;

            /* Zadani nove praci - je-li */
            pridel_praci(tid);
        }

        if ( (pracujicich_workeru <= 0) && (zadana_prace >= 100)) {
            break;
        }

    }
    printf("Hlavni %d: Ukoncuji smycku zprav\n", mytid);
```

```
    /* Ukonceni delniku */
    printf("Hlavni %d: Ukoncuji delniky. Cekam na odezvu...\n", mytid);
    pvm_mcast(tids, pocet_workeru, MSG_KONEC);  /* broadcast zpravy KONEC */
    /* cekani na odezvu od delniku */
    for(i = 0; i < pocet_workeru; i++) {
        pvm_recv(-1, MSG_KONEC);        /* kod KONEC od delniku*/
        pvm_upkint(&tid, 1, 1);             /* id delnika */
        //pvm_upkint(&worker_cnt, 1, 1);      /* pocet zpracovanych kousku
prace delnika */
        printf("Hlavni %d: Delnik cislo %d ukoncen\n", mytid, tid);
    }


    end_time = clock();


    /* Ukonceni PVM */
    pvm_exit();
    printf("Hlavni %d: Koncim s behem\n", mytid);



    /* Vysledky */
    printf("-------------------------------------------\n");
    printf("Hmotnost kvadru: %6.10f\n", hmotnost);
    printf("-------------------------------------------\n");


    return 0;
}
```

B.2 Modul exac-worker.c

```
#include <stdio.h>
#include <pvm3.h>

#include "exac-ro.h"
#include "exac-def.h"




int mytid;      /* Identifikator delnika */
```

```
int chief_tid;  /*  identifikator procesu sef  */
BOOL ukoncit;   /* Ma delnik uz skoncit cinnost */



double vypocti(double start_x, double start_y, double start_z,
double konec_x, double konec_y, double konec_z, double d, int druh_funkce)
{
    double x, y, z;
    double hmotnost, dm;
    int novy_prubeh, stary_prubeh;

    novy_prubeh = 0;
    stary_prubeh = 0;
    hmotnost = 0;

    for (x = start_x; x < konec_x; x += d) {
        for (y = start_y; y < konec_y; y += d) {
            for (z = start_z; z < konec_z; z +=d) {
                dm = d * d * d * ro(x + d / (double) 2,
y + d / (double) 2, z + d / (double) 2, druh_funkce );
                hmotnost += dm;

            }
        }
        novy_prubeh = (int) ( (x - start_x) / ((konec_x - start_x) / 100) );
        if (novy_prubeh != stary_prubeh) {
            printf("Delnik %d: Pocitam dilci hmotnost [%d\%]\n", mytid,
novy_prubeh);
            stary_prubeh = novy_prubeh;
        }
    }

    return hmotnost;
}



void log(char *str)
{
    FILE *fout;
    fout = fopen(LOG_FILE, "a");
```

```
    fprintf(fout, "%s", str);
    fclose(fout);
}



int main(int argc, char *argv[])
{

    char pomstr[100];

    /* Prihlaseni do PVM */
    mytid = pvm_mytid();
    sprintf(pomstr, "Delnik %d: Vytvoren\n", mytid);
    log(pomstr);

    /* cekani na registraci od sefa */
    pvm_recv(-1, MSG_REGISTRACE);               /* REGISTRACE kod zpravy */
    pvm_upkint(&chief_tid, 1, 1);  /* zjisti cislo sefa */



    /* kontrola zda je sef ten pravy - totozny s otcem */
    if (chief_tid == pvm_parent()) {

        /* odpoved sefovi */
        pvm_initsend(PvmDataDefault);   /* inic. bufferu */
        pvm_pkint(&mytid, 1, 1);        /* ulozeni identifikace */
        //pvm_pkint(&cnt_work, 1, 1);     /* ulozeni citace */
        pvm_send(chief_tid, MSG_PRIPRAVEN);      /* kod odpovedi PRIPRAVEN */
    }


    ukoncit = FALSE;
    while (ukoncit == FALSE) {


        /* Pokud obdrzime zpravu s kodem PRACE */
        if (pvm_nrecv(chief_tid, MSG_PRACE) > 0) {

            double m;

            double start_x, start_y, start_z;
```

```
            double konec_x, konec_y, konec_z;
            double d;
            int druh_funkce;

            pvm_upkdouble(&start_x, 1, 1);
            pvm_upkdouble(&start_y, 1, 1);
            pvm_upkdouble(&start_z, 1, 1);
            pvm_upkdouble(&konec_x, 1, 1);
            pvm_upkdouble(&konec_y, 1, 1);
            pvm_upkdouble(&konec_z, 1, 1);
            pvm_upkdouble(&d, 1, 1);
            pvm_upkint(&druh_funkce, 1, 1);


            m = vypocti(start_x, start_y, start_z,
konec_x, konec_y, konec_z, d, druh_funkce);

            sprintf(pomstr, "Delnik %d: Dilci hmotnost: %6.10f\n", mytid, m);
            log(pomstr);

            /* Poslani vysledku */
            pvm_initsend(PvmDataDefault);  /* inic. bufferu */
            pvm_pkint(&mytid, 1, 1);
            pvm_pkdouble(&m, 1, 1);
            pvm_send(chief_tid, MSG_VYSLEDEK);
        }


        /* Pokud obrdrzime zpravu s kodem ukoncit */
        if (pvm_nrecv(chief_tid, MSG_KONEC) > 0) {
            ukoncit = TRUE;
            sprintf(pomstr, "Delnik %d: Ukoncuji smycku\n", mytid);
            log(pomstr);
        }
    }


    /* Zaslani potvrzujici KONEC zpravy */
    pvm_pkint(&mytid, 1, 1);             /* ulozeni identifikace */
    pvm_send(chief_tid, MSG_KONEC);      /* kod odpovedi KONEC */
```

```
    /* ukonceni prace s PVM */
    pvm_exit();
    sprintf(pomstr, "Delnik %d: Koncim s behem\n", mytid);
    log(pomstr);

    return 0;
}
```

B.3 Modul exac-ro.c

```
#include "exac-ro.h"


double ro(double x, double y, double z, int druh_funkce)
{
    double vysledek;

    switch (druh_funkce) {
        case (RO_KONSTANTNI) :
            vysledek = 1;
            break;
        case (RO_PROMENNA_X) :
            vysledek = x;
            break;
        case (RO_PROMENNA_XYZ) :
            vysledek = x + y + z;
            break;
        default :
            vysledek = 0;
            break;
    }

    return vysledek;
}
```