ZÁPADOČESKÁ
UNIVERZITA
V PLZNI

KIV/PPR

Semestrální práce

8 - Výpočet hmotnosti kvádru

Vít Lachner
Os. Číslo: A06093
Narozen 16. 7. 1984
lachner@students.zcu.cz

Zadání

Realizujte výpočet celkové hmotnosti zadaného kvádru prostoru **(0, 0, 0, x, y, z)**. Je dána funkce hustoty prostoru **ro(x, y, z)**.

Povinné parametry:

- počet vláken
- velikost kusu práce přidělované vláknům
- 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

Povinné funkce:

- ro(x,y,z) = 1
- ro(x,y,z) = x
- ro(x,y,z) = x + y + z

Vypracujte ve dvou verzích:

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

Analýza úlohy

Cílem úlohy je vypočítat hmotnost kvádru. Paralelizace spočívá v rozdělení kvádru na malé části, které budou spočteny samostatně. Jednotlivé části (kvádry) jsou rozděleny mezi procesy (vlákna) dynamicky – model *farmer-worker*. *Farmer (manažer)* tedy rozděluje práci mezi *workery (dělníky)*.

Vstupy

Program potřebuje pro výpočet hmotnosti kvádru sedm hodnot:

- počet vláken – celé kladné číslo *(počet vláken > 0)*
- díl práce přidělený vláknu - kladné číslo v plovoucí řádové čárce *(z intervalu (0; 100>)*

- okraj kvádru – tři čísla v plovoucí řádové čárce
- integrační krok – kladné číslo v plovoucí řádové čárce *(krok > 0)*
- funkce hustoty.

Výstupy

Primární výstupem programu je hmotnost zadaného kvádru. Dále je možné uvést různé statistiky např. dobu běhu programu.

Postup zpracování

Farmer - manažer

1 Načtení parametrů výpočtu
2 *Manažer* vytvoří *dělníky*
3 Dokončil nějaký dělník výpočet? Ano – krok 4. Ne - krok 5.
4 Přičti výsledek k celkové hmotnosti
5 Je ještě co počítat? Ano - krok 6., Ne - krok 8.
6 Určení dávky k výpočtu pro *dělníka* a zadání práce *dělníkovi*
7 Zpět na krok 3.
8 Výpis výsledků

Worker – dělník

1 Je přidělená dávka k výpočtu? Ano – krok 2., Ne – krok 4.
2 Výpočet hmotnosti zadaného kvádru.
3 Zpět na krok 2.
4 Konec dělníka

Implementace – Java

Program byl odladěn na JRE 1.6.03 od firmy Sun. Vývoj probíhal v IDE Eclipse 3.2.

Popis důležitých tříd

Třída ThreadManager

Farmer, který vytváří workery. Obsahuje metody pro rozdělení práce workerům a sbírání výsledků jejich činnosti.

Třída ThreadWeight

Worker čekající na práci od farmera, kterou následně vykoná a čeká na její vyzvednutí.

Ovládání programu

Zdrojové kódy programu se nachází v adresáři **java/src**, který je umístěn v kořenovém adresáři archívu. K překladu lze využít Ant skript **build.xml**, který lze spustit dávkovým souborem **build.bat**. Výsledkem překladu je soubor **ppr.jar,** který se vytvoří v adresáři **java/src**. Spustitelný soubor **ppr.jar** je umístěn v adresáři **java/bin**. Program lze spusti bez parametrů nebo s parametry.

bez parametrů java -jar ppr.jar

s parametry java -jar ppr.jar 10 1 100 100 100 0.1 1

formát parametrů

java -jar pp.jar <pocet vl.> <velikost prace> <x> <y> <z> <d> <funkce hustoty>

Po spuštění programu se objeví formulář, ve kterém se vyplní parametry výpočtu. Výpočet se spustí tlačítkem „Spustit výpočet“. V textovém poli budou po ukončení výpočtu vypsány výsledky.

<pocet vlaken> - počet vláken, musí být > 0, celé číslo

<velikost prace> - velikost kusu práce přidělené vláknu z intervalu (0;100>

<x> <y> <z> - definice kvádru

<d> - integrační krok, stejný pro všechny osy, d > 0

<funkce> - 1 – (1), 2 – (x), 3 – (x + y + z)

výpis nápovědy

java -jar ppr.jar -h

Implementace – PVM

Popis důležitých souborů

Soubor farmer.c

Farmer, který vytváří workery. Dynamicky rozděluje práci jednotlivým workerům a čeká na jejich výsledky. Po ukončení výpočtu vypíše výsledek.

Soubor worker.c

Worker příjmá práci od farmera a vypočtení výsledku odešle výsledek farmerovi.

Ovládání programu

Soubory se zdrojovými kódy se nachází v adresáři **pvm/bin**. Ke zdrojovým kódum je připojen soubor **makefile**, kterým se vytvoří spustutelné soubory. Překlad se provádí voláním příkazu **make** takto:

make EXE=farmer

- přeloží farmera

make EXE=worker

- přeloží workera

Výsledkem překladu jsou soubory **farmer, worker**. Samotný program se spustí příkazem:

farmer <pocet vl.> <velikost prace> <x> <y> <z> <d> <funkce hustoty>

- formát parametrů je popsán výše u verze pro jazyk Java

Závěr

Program by měl splňovat podmínky zadání. Výsledky výpočtu jsou korektní přihlédnutím k zaokrouhlovacím chybám. Při testování verze obou verzí programu se nevyskytly žádné problémy.

Příloha – Java

ThreadManager.java

```
package ppr.threads.app;

import java.util.ArrayList;

/**
 * Trida pro spravu vypocetnich vlaken.
 *
 */
public class ThreadManager extends Thread {
	private Parameters parameters = null; // parametry vypoctu
	private ArrayList<ThreadWeight> threads = null;  // vypocetni vlakna
	private long startTime = 0;  // zacatek vypoctu
	private long endTime = 0;  // konec vvpoctu

	private float countedWork = 0;  // spoctena prace
	private float totalWeight = 0;  // celkova vaha

	/**
	 * Vytvori instanci spravce.
	 * @param parameters parametry ulohy
	 */
	public ThreadManager(Parameters parameters) {
		this.parameters = parameters;
	}

	/**
	 *
	 */
	public void run() {
		this.createThreads();
		this.startThreads();

		int deadThred = 0;  // pocet zabitych vlaken

		startTime = System.currentTimeMillis(); // zacatek vypoctu

		while (deadThred < threads.size()) {   // dokud vlakna pocitaji
			for (ThreadWeight threadWeight : threads) {  // cykl pres vypocetni vlakna
				switch (threadWeight.getStateThread()) {  // kontrola stavu vlakna
					case COUNTED:  // vlakno dokoncilo vypocet
						Result result = threadWeight.getResult();  // vysledek vypoctu
						totalWeight += result.getValue();  // scitani mezivysledku
						// neni zde break, protoze lze vyuzit nasledujici cast kodu
					case NEW:   // vlakno je nove nebo je vysledek spocten
						if (countedWork < 100f)  // kontrola jestli je co pocitat
							setWork(threadWeight); // nastaveni pracovni davky
						else {
							threadWeight.setStateWork(StateWork.DIE);   // neni co
pocitat --> ukonceni vlakna
							deadThred++;
						}
					break;
				}
			}
		}
		endTime = System.currentTimeMillis();  // konec vypoctu
	}

	/**
```

```
	 * Vybranemu vypocetnimu vlaknu nastavi parametry pro vypocet.
	 * @param threadWeight vypocetni vlakno
	 */
	private void setWork(ThreadWeight threadWeight) {
		Work work = new Work(parameters.getRo(), parameters.getD());  // davka k vypoctu

		// krajni body vnitrniho kvadru
		Point3D startPoint = new Point3D();
		Point3D endPoint = new Point3D();

		startPoint.setX(parameters.getX() * (countedWork / 100f));   //  vypocet prvni x-souradnice vnitrniho kvadru

		if ((countedWork + parameters.getPart()) <= 100f)  // kontrola jestli neprekrocime okraje ulohy
			countedWork += parameters.getPart();
		else
			countedWork = 100f;

		endPoint.setX(parameters.getX() * (countedWork / 100f));//     vypocet druhe x-souradnice vnitrniho kvadru

		// y a z se pocitaji az k okrajum.
		endPoint.setY(this.parameters.getY());
		endPoint.setZ(this.parameters.getZ());

		work.setStartPoint(startPoint);
		work.setEndPoint(endPoint);

		threadWeight.setWork(work);  // nastaveni davky vlaknu
		threadWeight.setStateWork(StateWork.SET);     // nastaveni priznaku, ze je davka nastavena a vlakno muze pocitat
	}

	/**
	 * Podle parametru ulohy vytvori urceny pocet vlaken.
	 *
	 */
	private void createThreads() {
		this.threads = new ArrayList<ThreadWeight>();

		for (int i = 0; i < parameters.getCountThreads(); i++)
			threads.add(new ThreadWeight());
	}

	/**
	 * Spusti vsechny vypocetni vlakna.
	 *
	 */
	private void startThreads() {
		for (ThreadWeight threadWeight : threads)
			threadWeight.start(); // spusteni vlaken
	}

	/**
	 * Vraci cas, kdy byl vypocet ukoncen.
	 * @return systemovy cas v ms
	 */
	public long getEndTime() {
		return endTime;
	}

	/**
	 * Vraci cas, kdy byl vypocet zahajen.
	 * @return systemovy cas v ms
	 */
	public long getStartTime() {
		return startTime;
```

```
	}

	/**
	 * Vraci celkovou hmotnost kvadru.
	 * @return hmotnost
	 */
	public float getTotalWeight() {
		return totalWeight;
	}
}
```

ThreadWeight.java

```
package ppr.threads.app;


/**
 * Vypocetni vlakno. Ze zadane davky (kvadru) spocita objem.
 *
 */
public class ThreadWeight extends Thread {
	private StateWork stateThread = null; // stav vlakna
	private Result result = null;	// vysledek vypoctu
	private Work work = null;  // pracovni davka

	/**
	 * Vytvori instanci vypocetniho vlakna.
	 *
	 */
	public ThreadWeight() {
		this.stateThread = StateWork.NEW; // nastaveni stavu vlakna
	}

	@Override
	public void run() {
		while (this.getStateThread().compareTo(StateWork.DIE) != 0) {  // dokud je nejaka prace
			if (this.getStateThread().compareTo(StateWork.SET) == 0) {  // pracovni davka nastavena
				float weight = 0;

				//System.out.println(this.getName());

				this.result = new Result();  // instance objektu pro vysledek
				this.result.setTimeStart(System.currentTimeMillis());  // zacatek casu vypoctu

				for (float x = work.getStartPoint().getX(); x < work.getEndPoint().getX(); x += work.getD()) {
					for (float y = work.getStartPoint().getY(); y < work.getEndPoint().getY(); y += work.getD()) {
						for (float z = work.getStartPoint().getZ(); z < work.getEndPoint().getZ(); z += work.getD()) {
							float mX = x;
							float mY = y;
							float mZ = z;

							float dX = work.getD();
							float dY = work.getD();
							float dZ = work.getD();

							// osetreni pripadu, pokud je soucasna pozice + integracni krok vetsi nez okraj zadaneho vypoctu

							if ((x + work.getD()) > work.getEndPoint().getX())
								dX = mX = work.getEndPoint().getX() - x;
```

```
                                                                        if   ((y  +  work.getD())  >    work.getEndPoint().getY())

                                                                                dY = mY = work.getEndPoint().getY() - y;


                                                                        if   ((z  +  work.getD())  >    work.getEndPoint().getZ())

                                                                                dZ = mZ = work.getEndPoint().getZ() - z;

                                                                        // vypocet vahy
                                                                        weight += dX * dY * dZ * work.getFncRo().getRo(mX
+ dX / 2f, mY + dY / 2f, mZ + dZ / 2f);
                                                                                }
                                                                        }
                                                                }

                                                                        this.result.setValue(weight);  // ulozeni vysledku
                                                                        this.result.setTimeEnd(System.currentTimeMillis());  // konec vypoctu
                                                                        this.setStateWork(StateWork.COUNTED);  // nastaveni priznaku ze je vse spocteno
                                                        }

                                                        try {
                                                                Thread.sleep(0); // uspani...
                                                        } catch (InterruptedException e) {}
                                        }
                        }

                        /**
                         * Vraci stav vlakna.
                         * @return
                         */
                        public StateWork getStateThread() {
                                return stateThread;
                        }

                        /**
                         * Nastavi stav vlakna. Ridici vlakno na konci vypoctu nastavi StateWork.DIE a vlakno skonci.
                         * @param state novy stav
                         */
                        public void setStateWork(StateWork state) {
                                synchronized (state) {
                                        this.stateThread = state;
                                }
                        }

                        /**
                         * Vrati vysledek vypoctu.
                         * @return vysledek vypoctu
                         */
                        public Result getResult() {
                                return result;
                        }

                        /**
                         * Nastaveni pracovni davky.
                         * @param work
                         */
                        public void setWork(Work work) {
                                this.work = work;
                        }
}
```

Příloha – PVM

Soubor farmer.c

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

#include "def.h"

T_POINT point;  // krajni bod kvadru
float d;   //  diference
int count_workers; // pocet delniku
float part;  // cast prace pridelena delikovi
int fnc; // funkce hustoty

float counted_work = 0;

int working = 0;

int mytid;  // identifikator procesu

void print_help() {
        printf("Popis formát příkazu pro spuštění programu:\n\n");
        printf("Worker: \n");
        printf("\tworker\n\n\n");
        printf("Farmer: \n");
        printf("\farmer <pocet vl.> <velikost prace> <x> <y> <z> <d> <funkce hustoty>\n");
        printf("Formát parametrů:\n");
        printf("\t<pocet vl.>       - počet vláken - musi byt > 0\n");
        printf("\t<velikost prace>  - velikost kusu práce přidělené vláknu z intervalu (0;100>\n");
        printf("\t<x> <y> <z>       - definice kvádru\n");
        printf("\t<d>               - integrační krok (stejny pro vsechny osy) - musi platit d > 0\n");
        printf("\t<funkce>          - \n");
        printf("\t\t1:              - 1\n");
        printf("\t\t2:              - x\n");
        printf("\t\t3:              - x + y + z\n\n");
        printf("Příklad spuštění programu s parametry:\n");
        printf("\tfarmer 10 0.1 1.0 1.0 1.0 0.1 1\n");
}

void set_work(int tid) {
        if ((counted_work + part) <= 100.0) {
                float start_x = point.x * (counted_work / (float) 100);
    float start_y = 0.0;
    float start_z = 0.0;

                counted_work += part;

    float end_x = point.x * (counted_work / (float) 100);
    float end_y = point.y;
    float end_z = point.z;

       // prideleni prace delnikovi
    pvm_initsend(PvmDataDefault);   // inicializace bufferu
    pvm_pkfloat(&start_x, 1, 1);
    pvm_pkfloat(&start_y, 1, 1);
    pvm_pkfloat(&start_z, 1, 1);
    pvm_pkfloat(&end_x, 1, 1);
    pvm_pkfloat(&end_y, 1, 1);
    pvm_pkfloat(&end_z, 1, 1);
    pvm_pkfloat(&d, 1, 1);
    pvm_pkint(&fnc, 1, 1);
```

```
		pvm_send(tid, STATE_WORK);
				working++;
			}
			else {
				if (counted_work >= 100.0)
					return;

		float start_x = point.x * (counted_work * (float) 100);
		float start_y = 0.0;
		float start_z = 0.0;

				counted_work = 100;

		float end_x = point.x;
		float end_y = point.y;
		float end_z = point.z;

		/* Zaslani prace workerovi */
		pvm_initsend(PvmDataDefault);   /* inicializace bufferu */
		pvm_pkfloat(&start_x, 1, 1);
		pvm_pkfloat(&start_y, 1, 1);
		pvm_pkfloat(&start_z, 1, 1);
		pvm_pkfloat(&end_x, 1, 1);
		pvm_pkfloat(&end_y, 1, 1);
		pvm_pkfloat(&end_z, 1, 1);
		pvm_pkfloat(&d, 1, 1);
		pvm_pkint(&fnc, 1, 1);
		pvm_send(tid, STATE_WORK);
				working++;
	}
}

int main(int argc, char *argv[]) {
		int tids[32];	// identifikatory procesu
	int* tidc;		// pomocny ukazatel pro vytvareni delniku
	int create_workers = 0;	// pocet vytvorenych workeru
	int tid;		// pomocny identifikator procesu


		float weight = 0, total_weight = 0;

		int i;

		printf("Výpočet hmotnosti kvádru\n");
		printf("------------------------\n");
		printf("Semestrální práce z předmětu KIV/PPR\n\n");
		printf("\tAutor:	Vít Lachner\n");
		printf("\tSt. číslo: A06093\n");
		printf("\tE-mail:	lachner@students.zcu.cz\n\n\n");

		if (argc == 8) {
			count_workers = atoi(argv[1]);
			part = atof(argv[2]);
			point.x = atof(argv[3]);
			point.y = atof(argv[4]);
			point.z = atof(argv[5]);
			d = atof(argv[6]);
			fnc = atoi(argv[7]);

			if ((count_workers <= 0) ||
				(part <= 0) ||
				(part > 100) ||
				(d <= 0) ||
				(fnc < 0) ||
				(fnc > 3)) {
				printf("chybne parametry\n");
				return;
```

```
                }
        }
        else {
    print_help();
                return 0;
        }

        mytid = pvm_mytid();

        tidc = tids;

        // vytvareni delniku
    while (count_workers > create_workers) {
        if (pvm_spawn("worker", (char**)0, 0, "", 1, tidc)) {  // vytvoreni delnika
            create_workers++;
            tidc++;
        }
                else {  // neuspesne vytvoreni delnika
            printf("chyba %d pri vytvareni delnika.", *tidc);
        }
    }

        // uvodni komunikace farmare s workery
        pvm_initsend(PvmDataDefault);                   // inicializace bufferu
        pvm_pkint(&mytid, 1, 1);                        // ulozeni identifikace
        pvm_mcast(tids, count_workers, STATE_REG);  // broadcast zpravy

        // cekani na zpravu STATE_READY
        for(i = 0; i < count_workers; i++) {
    pvm_recv(-1, STATE_READY);              // delnik jse pripraven
    pvm_upkint(&tid, 1, 1);         // id delnika
    }

        for (i = 0; i < count_workers; i++) {
    set_work(tids[i]);
    }

    while (working != 0) {
        if (pvm_nrecv(-1, STATE_COUNTED) > 0) {
            pvm_upkint(&tid, 1, 1);
            pvm_upkfloat(&weight, 1, 1);

                                --working;

            total_weight += weight;

            set_work(tid);  // zadani prace
        }
    }

    pvm_mcast(tids, count_workers, STATE_DIE);  // ukonceni delniku

    // cekani na ukonceni delniku
    for(i = 0; i < count_workers; i++) {
        pvm_recv(-1, STATE_DIE);        // delnik
        pvm_upkint(&tid, 1, 1);         // id delnika
    }

    // ukonceni PVM
    pvm_exit();

    // tisk vysledku
    printf("\nhmotnost kvadru: %.10f\n", total_weight);

        return 0;
}
```

Soubor worker.c

```
#include <pvm3.h>

#include "def.h"

int mytid;      /* Identifikator delnika */
int farmer_tid;  /*  identifikator procesu sef  */

float fnc_ro(float x, float y, float z, int fnc)
{
   switch (fnc) {
      case (FNC_CONSTANT):
         return 1;
      case (FNC_X):
         return x;
      case (FNC_XYZ):
         return x + y + z;
      default:
         return 0;
   }
}

float count_weight(float start_x, float start_y, float start_z, float end_x, float end_y, float end_z, float d, int fnc)
{
   float x, y, z;
   float weight = 0;

            for (x = start_x; x < end_x; x += d) {
                        for (y = start_y; y < end_y; y += d) {
                                    for (z = start_z; z < end_z; z += d) {
                                                float mX = x;
                                                float mY = y;
                                                float mZ = z;

                                                float dX = d;
                                                float dY = d;
                                                float dZ = d;

                                                // osetreni pripadu, pokud je soucasna pozice + integracni krok vetsi nez okraj
zadaneho vypoctu

                                                if  ((x + d) > end_x)
                                                            dX = mX = end_x - x;


                                                if  ((y + d) > end_y)

                                                            dY = mY = end_y - y;


                                                if  ((z + d) > end_z)

                                                            dZ = mZ = end_z - z;

                                                // vypocet vahy
                                                weight += dX * dY * dZ * fnc_ro(mX + dX / (float) 2, mY + dY / (float) 2, mZ + dZ /
(float) 2, fnc);
                                    }
                        }
            }

            return weight;
}
```

```
main(int argc, char *argv[]) {
   mytid = pvm_mytid();        // prihlaseni do pvm

   pvm_recv(-1, STATE_REG);                    // prijeti zpravy s registraci
   pvm_upkint(&farmer_tid, 1, 1);       // cislo farmera

            if (farmer_tid == pvm_parent()) {
      pvm_initsend(PvmDataDefault);              // inicializace bufferu
      pvm_pkint(&mytid, 1, 1);                                 // ulozeni identifikace
      pvm_send(farmer_tid, STATE_READY);   // potvrzeni pripravenosti
   }

            while (1) {
      if (pvm_nrecv(farmer_tid, STATE_WORK) > 0) {  // pridelena prace
         float start_x, start_y, start_z, end_x, end_y, end_z, d, weight;
         int fnc;

                                    // prijeti
         pvm_upkfloat(&start_x, 1, 1);
         pvm_upkfloat(&start_y, 1, 1);
         pvm_upkfloat(&start_z, 1, 1);
         pvm_upkfloat(&end_x, 1, 1);
         pvm_upkfloat(&end_y, 1, 1);
         pvm_upkfloat(&end_z, 1, 1);
         pvm_upkfloat(&d, 1, 1);
         pvm_upkint(&fnc, 1, 1);

         weight = count_weight(start_x, start_y, start_z, end_x, end_y, end_z, d, fnc);

                                    // odeslani vysledku
         pvm_initsend(PvmDataDefault); // inicializace bufferu
         pvm_pkint(&mytid, 1, 1);
         pvm_pkfloat(&weight, 1, 1);  // vysledek
         pvm_send(farmer_tid, STATE_COUNTED);
      }
                        else if (pvm_nrecv(farmer_tid, STATE_DIE) > 0) {  // ukonceni workera
         pvm_pkint(&mytid, 1, 1);            // ulozeni identifikace
                                    pvm_send(farmer_tid, STATE_DIE);      // potvrzeni ukonceni

                                    pvm_exit();  // ukonceni PVM
                                    return 0;
                        }
   }

   return 0;
}
```