Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

PARALELNÍ PROGRAMOVÁNÍ

Semestrální práce

Hana Butková
A06076
h.butkova@seznam.cz

Zadání

Zadání číslo 3: Realizujte výpočet k-té mocniny celočíselné matice (n,n).

Povinné parametry:

- počet vláken
- velikost kusu práce přidělovaného vláknům
- vstupní matice
- hodnota k

Řešení

Zadaná úloha je vhodná pro aplikaci modelu SPMD (Single Program Muliple Data), tj. dekompozice podle dat. Několik procesů pracuje podle stejného programu nad strukturně stejnými, ale hodnotami se lišícími daty

Pro výpočet mocniny matice je použit nejjednodušší algoritmus. Zadaná matice se vynásobí sama sebou tolikrát podle toho, jaká je zadaná mocnina k.

Velikost práce v obou programech znamená počet prvků matice, které se přiřadí podřízenému procesu.

Paralelizuje se jedno násobení matic, tj. podřízené procesy dostanou za úkol spočítat několik prvků matice. Podřízené procesy jsou vždy synchronizovány na konci jednoho násobení matic. Čeká se na spočítání všech prvků matice. Potom se začne další kolo mocniny, výsledek minulého násobení se násobí zadanou maticí.

Paralelní program pro systém se sdílenou pamětí (Java)

Postup

Největší význam má třída PowerMonitor, která vytvoří zadaný počet vláken Worker. Na začátku výpočtu se připraví práce, která se bude vláknům přidělovat. Vytvoří se fronta objektů WorkPart, každá instance WorkPart obsahuje zadaný počet prvků matice (nebo menší, pokud jich už tolik nezbývá). Jako oddělovače mezi jednotlivými mocninami se vloží do fronty práce prázdný objekt. Po rozdělení práce na části se aktivují všichni dělníci.

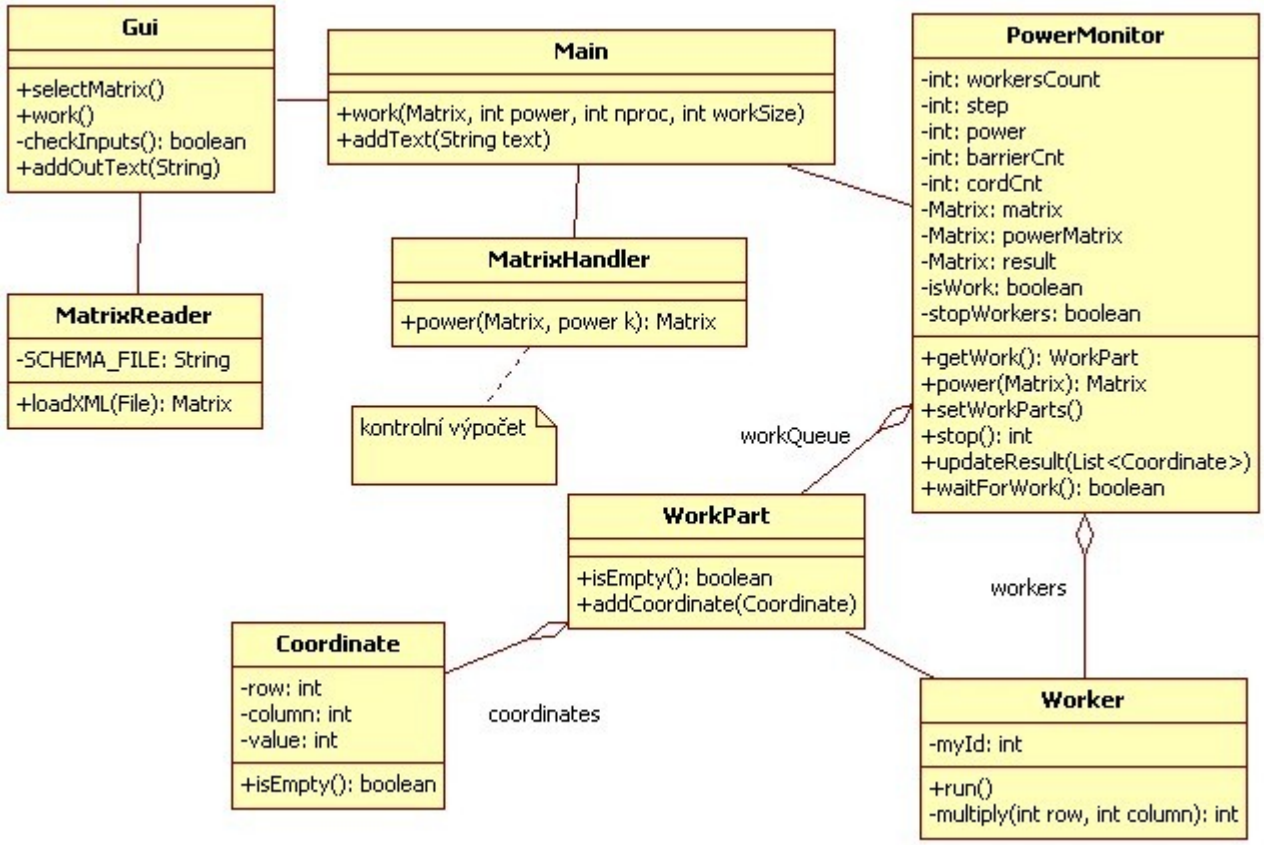
Dělník ve smyčce kontroluje, jestli je práce. Pokud není, uspí se. Po příjmu práce prochází přidělenou frontou a počítá zadané prvky součinu matic. Žádá další práci. Pokud je přidělená fronta prázdná, znamená to, že úkoly pro aktuální mocninu již byly rozdány, a dělník odešle všechny spočítané výsledky objektu PowerMonitor (metoda *updateResult()*).

V metodě *updateResult()* se aktualizuje mocnina zaslanými výsledky. Pokud výsledky nedodala ještě všechna vlákna, aktuální vlákno se uspí metodou *wait()*. Až pošle výsledky i

poslední dělník, pokračuje se dále. Pokud se už došlo k zadané mocnině, nastaví se příznak práce na *false* a vzbudí se všichni dělníci, aby se mohli ukončit. V opačném případě se zkopíruje výsledek dosavadní mocniny do násobené matice, odstraní se prázdný objekt WorkPart z fronty práce a probudí se všichni dělníci, aby mohli pokračovat v práci.

Před paralelním výpočtem je navíc mocnina vypočítána pro kontrolu běžným sekvenčním postupem.

Diagram tříd



Spuštění

Program se spustí příkazem:

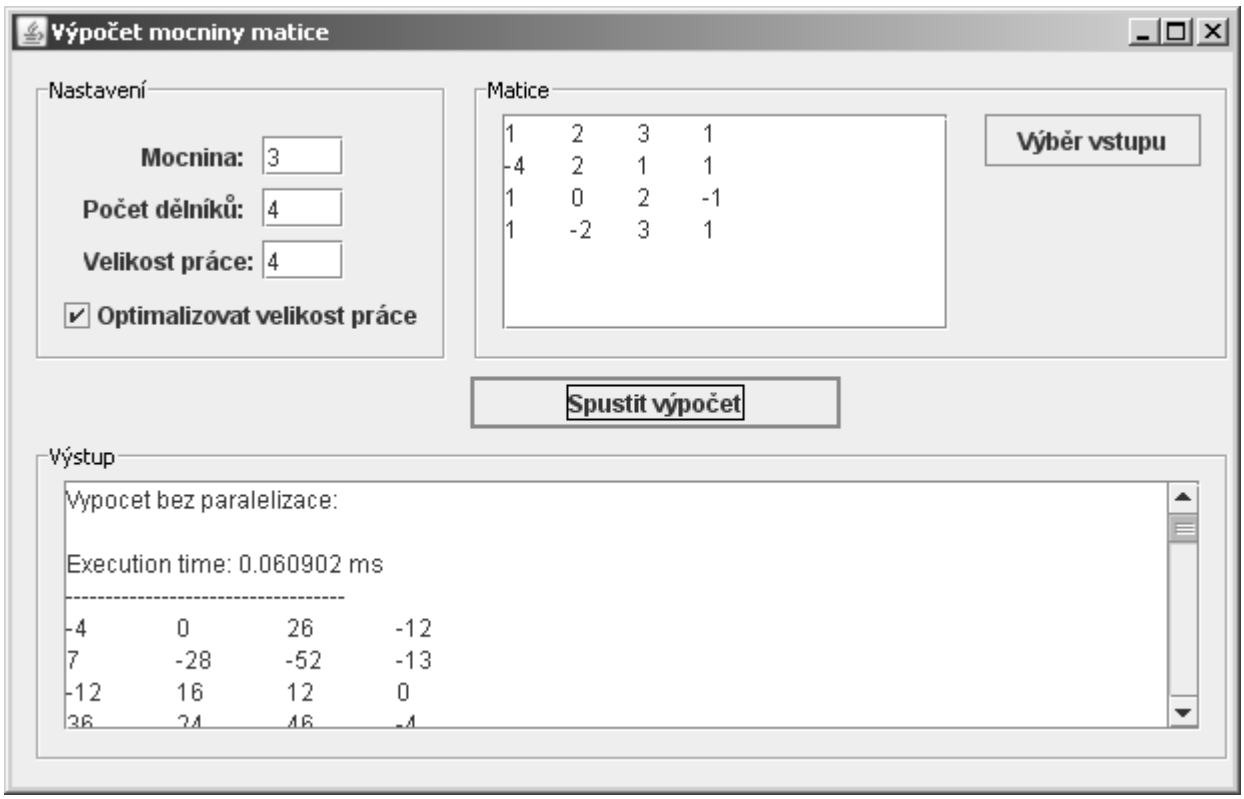
```
java -jar matrixPower.java
```

Matice jsou uložené v přiložených XML dokumentech. Jejich struktura je popsána ve schématu *matrix.xsd*.

Formát zadávaných vstupů je kontrolován. Hodnoty a struktura vybrané matice je kontrolována podle přiloženého XSD schématu.

Program po spuštění výpočtu zobrazí, jestli výpočet proběhl správně. (Porovnává výsledek s kontrolném sekvenčním výpočtem.)

Program byl odladěn na verzi Javy 1.6.0.



Výsledky

Program měří čas výpočtu, následující hodnoty jsou ale změřeny jen na jednoprocesorovém stroji (spuštěno bez výpisů). Na školních serverech aplikace nelze spustit kvůli GUI.

Rozměr matice	4	4	4
Mocnina	30	30	30
Počet dělníků	4	8	2
Velikost práce	2	2	8
Čas [ms]	2.9529	1.9729	0.8141

Paralelní program pro systém s distribuovanou pamětí (PVM)

PVM je programovací nástroj určený zejména k využití v heterogenní síti konvenčních pracovních stanic pracujících pod operačním systémem Unix. Prostřednictvím PVM jsou tyto stanice softwarově integrovány do jednoho multiprocesoru s distribuovanou pamětí. Jako způsob interakce je využívána asynchronní komunikace s přímým adresováním.

Klíčovou částí prostředí PVM je proces PVMD, který běží na pozadí na každém počítači a

realizuje komunikační a synchronizační funkce. Toto aplikační rozhraní je k dispozici ve formě knihoven pro jazyk C.

Prostředí PVM lze dobře využít pro paralelní aplikace s procesy s velkým objemem výpočtu a malou komunikací, tj. použitý model *processor farm*. Řídící proces *farmer* zadává příkazy (zprávy) svým podřízeným procesům – dělníkům (*workers*). Dělníci po vykonání příkazů zašlou zpět výsledek.

Postup

Každá zpráva je označena kódem, může obsahovat libovolná data. Pro odesílání a příjem zprávy se používají funkce:

- *pvm_mcast* (id příjemců, počet příjemců, kód zprávy) – zpráva skupině procesů
- *pvm_send* (id příjemce, kód zprávy) – zpráva konkrétnímu procesu
- *pvm_recv* (id odesilatele, kód zprávy) – blokující příjem zprávy od daného procesu

Význam kódů zpráv v programu:

- 4 – úvodní komunikace farmera s dělníky
- 5 – odpověď dělníka, že je připraven
- 6 – zadání úkolu (počet souřadnic a čísla souřadnic)
- 7 – zaslání výsledku dělníkem
- 8 – farmer zasílá dělníkům matice, které se budou násobit

Farmer vytvoří své dělníky, zašle jim zprávu s kódem 4 se svým ID. Dělníci potvrdí připravenost zprávou s kódem 5. V zadanou velikost mocniny se v cyklu provádí následující činnosti.

Farmer zašle zprávou s kódem 8 dělníkům matice, které se budou násobit. Dělníci potvrdí příjem matic zprávou s kódem 7. Poté farmer přiřadí všem dělníkům úkol, tj. pošle jim zprávu s kódem 6, která obsahuje počet prvků matice a jejich souřadnice. Souřadnice jsou vyjádřeny číslem pozice v matice, např. pro matici velikosti 4x4:

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Farmer čeká na výsledek od dělníků – na zprávu s kódem 7. Zpráva obsahuje data:

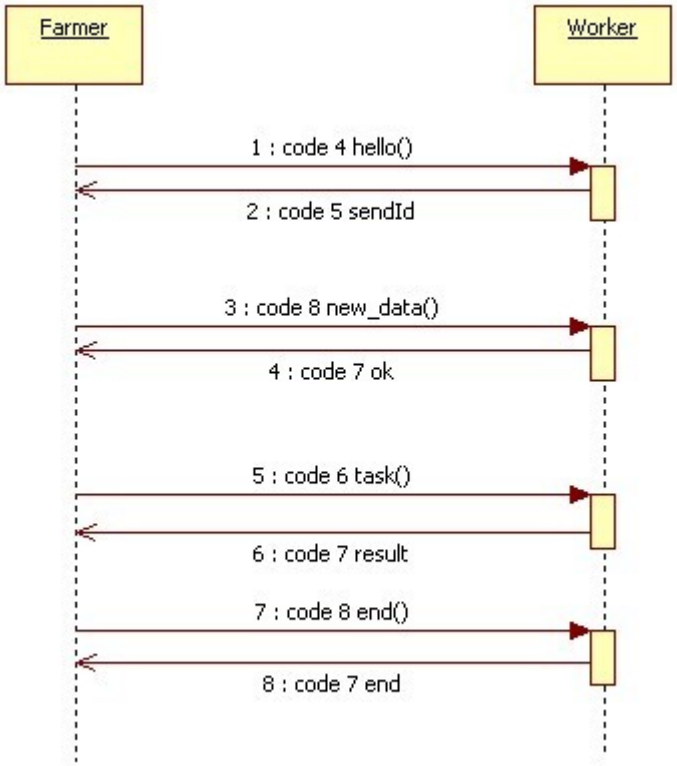
počet prvků, souřadnice prvku, hodnota prvku, souř. dalšího prvku, …

Zbývá-li další práce, znovu se přidělí. Pokud jsou všechny prvky matice spočítané, zašle se

dělníkům zpráva s kódem 6 a zápornou hodnotou (konec jednoho násobení). A pokračuje se další mocninou.

Po vypočítání zadané mocniny se zašle dělníkům zpráva s kódem 8 a záporným číslem, která znamená konec práce.

Diagram zasílání zpráv



Spuštění

1. Připojte se přes ssh na *hydra.fav.zcu.cz* (nebo IP 147.228.67.30)
2. Do domovského adresáře nakopírujte soubor (pomocí WinScp, resp. scp) hostfile.
3. Ve vašem domovském adresáři na Hydře si založte podadresář `pvm` - pozor je to case-sensitive.
4. Spusťte PVM daemona posloupností příkazů

   ```
   pvmd $HOME/hostfile
   CTRL+Z
   bg
   ```
5. Do vytvořeného adresáře pvm nakopírujte soubory farmer.c, worker.c a makefile.
6. V adresáři pvm pøeložte soubory příkazy

   ```
   make EXE=farmer
   make EXE=worker
   ```
7. Přeložený program spustťe příkazem `farmer`.
8. Pokud se výpočet programu řádně neukončí, je třeba:

   - suspendovat program pomocí `CTRL+Z`
   - vypsat stav příkazem `ps -a` a zjistit číselné ID suspendovaného programu

- "zabít" suspendovaný program příkazem `kill -9 ID`

9. Před ukončením práce je nutné ukončit PVM daemona - spustit ovládací konzoli příkazem pvm.
   V ní napsat příkazy:
   `reset` - ukonèí všechny zbylé "visící" procesy
   `halt` - ukončí samotného PVM daemona

Výsledky

```
butkova@hydra:~/pvm$ farmer "inputExample.txt" 20 3 7

----------------------------
 Vypocet n-te mocniny matice
----------------------------
Matice 4 x 4
1       2       3       1
-4      2       1       1
1       0       2       -1
1       -2      3       1

Mocnina : 20
Pocet delniku : 3
Kus prace : 7
Vytvoren delnik 524289.
Vytvoren delnik 786433.
Vytvoren delnik 1048577.
Delnik cislo 524289 k praci pripraven!
Delnik cislo 786433 k praci pripraven!
Delnik cislo 1048577 k praci pripraven!
Delnik 524289 konci s hodnotou -16 ! Vykonal prace: 19
Delnik 786433 konci s hodnotou -16 ! Vykonal prace: 19
Delnik 1048577 konci s hodnotou -16 ! Vykonal prace: 19

Vysledek - 20. mocnina:
Matice 4 x 4
493881856       271069184       -1918243840     1466411520
-1535237632     1653982208      1106859520      -1923267072
-1566998528     -1273661440     -1530171392     2011949056
1940394496      -1160970240     -1979433984     493881856
```

Závěr

U programu v PVM se nepodařilo změřit čas, protože příkazy z knihovny <time.h> při pouštění na Hydře nevracely žádné hodnoty. Program v Javě nešel kvůli GUI odzkoušet na Eryxu, časy výpočtu tedy nejsou reprezentativní.

Programy fungují správně, pro velké mocniny ale mohou hodnoty prvků přetéct maximální hodnotu typu integer.

Literatura

Ježek K., Matějovic P., Racek S.: *Paralelní architektury a programy*, skripta ZČU

Herout P.: *Učebnice jazyka JAVA*, nakladatelství Kopp, České Budějovice 2000

Výpis částí kódu

Java

PowerMonitor.java

```
package matrixPower;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
import java.util.Queue;

/* Farmer spoustejici workery a zadavajici jim praci */
public class PowerMonitor {

  /* Trida delnika */
  private class Worker extends Thread {

    private PowerMonitor monitor;    /* reference monitoru */
    private int myId;                /* id delnika */

   ...

    /* vypocet jedne souradnice mocniny matice */
    private int multiply(int row, int column){
        int sum = 0;
      int grade = monitor.getMatrix().getGrade();

      for (int m = 0; m < grade; m++){
        sum += monitor.getPowerMatrix().get(row , m)*monitor.getMatrix().get(m, column);
       }
       return sum;
    }

    @Override
    /* prace delnika */
    public void run () {
        WorkPart part;  /* prace, kterou ma delnik spocitat */
        /* spocitane vysledky */
        List<Coordinate> locResult = new ArrayList<Coordinate>();

        while (monitor.waitForWork(this.myId)) { // blocking
         /* prideleni prace */
          while (!(part = monitor.getWork()).isEmpty()) {
            /* pocitani prvku mocniny matice */
            while (!part.isEmpty()) {
                Coordinate cord = part.getCoordinates().poll();
                cord.setValue(multiply(cord.getRow(), cord.getColumn()));
                locResult.add(cord);
                yield();
            }
          }
         /* zaslani vysledku rodici */
          monitor.updateResult(locResult, this.myId);
          locResult = new ArrayList<Coordinate>();
        }
      Main.addText("worker " + myId + " terminated");
    }
  }


  private int workersCount;   /* pocet delniku */
  private int barrierCnt;     /* pocet hotove prace */
  private int cordCnt;        /* pocet spocitanych souradnic */
  private int step;           /* velikost prirazovane prace */
  private int power;          /* stupen mocniny */

  private Matrix matrix;      /* puvodni matice */
  private Matrix powerMatrix; /* n-ta mocnina matice */
  private Matrix result;      /* vysledek mocneni matice */
  private Queue<WorkPart> workQueue;    /* fronta prirazovane prace */

  private boolean isWork;     /* jestli je nejaka prace */
  private boolean stopWorkers;  /* zda jsou delnici zastaveni */

  private Worker[] workers;    /* pole delniku */
```

```
public PowerMonitor (int n_proc, int step, int power) {
  ...
  workers = new Worker[getWorkersCount()];
  for (int i = 0; i < getWorkersCount(); i++) {
    workers[i] = new Worker (this, i);
    workers[i].start();
  }
}

/* vypocet mocniny matice */
public synchronized Matrix power (Matrix matrix) {
  this.setMatrix(matrix);
  this.setPowerMatrix(matrix);
  setResult(new Matrix(matrix.getGrade()));
  barrierCnt = 0;
  cordCnt = 0;
  isWork = true;

  setWorkParts();
  notifyAll();                // it awakes waiting workers
  while (isWork)
    try {       // wait for final result
       wait ();
     }
    catch (InterruptedException e) {};
  return getResult();
}

/* pocatecni generovani fronty prace */
private void setWorkParts(){
  int grade = this.getMatrix().getGrade();
  workQueue = new LinkedList<WorkPart>();
  WorkPart workPart = new WorkPart();

  /* vkladani souradnic do fronty */
  for (int m = 1; m < this.power; m++){
     workPart = new WorkPart();
     for (int i = 0; i < grade; i++){
       for (int j = 0; j < grade; j++){
        workPart.addCoordinate(new Coordinate(i, j));
        if (workPart.getCoordinates().size() == this.getStep()){
          workQueue.add(workPart);
          workPart = new WorkPart();
        }
       }
     }
    if (!workPart.isEmpty()){
      workQueue.add(workPart);
      workPart = new WorkPart();
    }
    /* vlozeni prazdne souradnice mezi jednotlivymi mocninami */
     if (m != power - 1){
      workQueue.add(new WorkPart());
     }
  }
  isWork = true;
}

/* zastaveni delniku */
public synchronized int stop () {
  stopWorkers = true;
  notifyAll();
  return 0;
}

/* pozastaveni delniku, dokud neni prace */
private synchronized boolean waitForWork(int id) {
  while ((!isWork) && (!stopWorkers)) try {
     wait ();
  } catch (InterruptedException e) {};

  if (stopWorkers) {
     return false;  // workers end their working loop
  }
  else {
     return true;  // worker has something to do
  }
}
```

```
  /* prirazeni prace delnikovi, tj. nekolik souradnic */
  private synchronized WorkPart getWork () {

    if ((workQueue.isEmpty()) || (workQueue.peek().isEmpty())){
      return new WorkPart();
    }
    return workQueue.poll();
  }

  /* prijmuti vysledku od delnika */
  /* blokuje delniky, dokud neni dokoncena konkretni mocnina */
  private synchronized void updateResult (List<Coordinate> coordinates, int id) {
    cordCnt += coordinates.size();
    Main.addText("\t" + coordinates + ", sum: " + cordCnt);
    /* aktualizovani vysledne matice */
    for (Coordinate cord: coordinates){
       getResult().set(cord);
    }
    barrierCnt++;
    /* pokud jsou spocitany vsechny souradnice, ukoncit vypocet */
    if (cordCnt == (power - 1) * matrix.getGrade() * matrix.getGrade()){
       isWork = false;
      notifyAll();
      return;
    }
    /* pokud neposlali vysledky vsichni delnici, uspat delnika */
    while (barrierCnt % getWorkersCount() != 0) {
       try {
         wait();
       } catch (InterruptedException e) {};
    }
    /* priprava dalsi mocniny;
     * nasobit se bude vysledna matice koncici mocniny
     */
    this.setPowerMatrix(this.getResult());

    /* vyjmout prazdnou souradnici z fronty prace,
     * aby se mohlo pokracovat v dalsi mocnine */
    if (!this.workQueue.isEmpty() && this.workQueue.peek().isEmpty()){
       this.workQueue.poll();
    }

    /* vzbuzeni vsech delniku */
    notifyAll();
  }
       ...
}
```

PVM

Farmer.c

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

/* PVM - model Farmer - Workers                                    */
/* Pouzivane kody zprav:                                           */
/*   4 ... broadcast (id sefa  -> delnici)                         */
/*   5 ... odpoved   (id delnika -> sef)                           */
/*   6 ... zadani ukolu (sef -> delnik),                           */
/*         zaporna hodnota cisla - konec jednoho nasobeni          */
/*   7 ... vysledek     (delnik -> sef),                           */
/*  8 ... nova data                                                */
/*        zaporne prvni cislo - konec cinnosti                      */

void printMatrixWithGrade(int * matrix);
void printMatrix(int * matrix, int grade);
int* readMatrix(char file[]);

main(int argc, char *argv[]) {
  int mytid;      /*  identifikator procesu sef  */
  int tids[32];   /*  identifikatory procesu delnici */
  int nproc;      /*  pocet delniku */
  int m;          /*  kolik souradnic ma matice */
  int workSize;   /* velikost pridelovane prace */
  int power;      /* mocnina */
  int* rslt;      /*  ukazatel na pole vysledku */
  int last_rslt;
```

```
  int worker_cnt = -1894;   /* nastaveni pro kontrolu smysluplnosti */
  int rslt_num = 0;   /*  kolik prvocisel uz se naslo */
  int cnt_sent = 0;    /*  kolik prikazu odeslano */
  int cnt_received = 0;/*  kolik vysledku prijato */
  int i, tid, work;  /*  pomocne promenne */
  int k = 0;      /* cislo souradnice, ktera jeste neni prirazena */
  int end_param = 0;
  int vytvoreno = 0;  /*  pocet uspesne vytvorenych delniku */
  int* tidc;      /* pomocny ukazatel pro vytvareni delniku */

  int grade;      /* stupen matice */
       int * matrix;    /* vstupni matice */
       int tmp;
       int j = 0;

       ...
/* kontrola zadani spravneho poctu argumentu */
       ...
 /* kontrola spravnosti zadanych cisel */
       ...

  matrix = readMatrix(file);   /* nacteni matice ze textoveho souboru */
  grade = *matrix; /* prvni hodnota matice je jeji stupen */

  m = grade * grade;

  /* inicializace pole pro vysledky vypoctu */
  rslt = (int *) malloc ((m + 1) * sizeof(int));

  mytid = pvm_mytid();  /* prihlaseni do PVM  */

  /* vytvoreni delniku */
  tidc = tids;
  while (nproc > vytvoreno) {
   if (pvm_spawn("worker", (char**)0, 0, "", 1, tidc)) {
      printf("Vytvoren delnik %d.\n", *tidc);
      vytvoreno++;
      tidc++;
   } else {
      printf("Chyba pri vytvareni delnika. Cislo chyby %d\n", *tidc);
   }
  }

  /* uvodni komunikace sefa s delniky (registrace prac. sil) */
  pvm_initsend(PvmDataDefault);   // inicializace bufferu
  pvm_pkint(&mytid, 1, 1);    // ulozeni identifikace
  pvm_mcast(tids, nproc, 4);  // broadcast zpravy s kodem 4 vsem delnikum

  // cekani na odezvu od delniku
  for(i = 0; i < nproc; i++) {
    pvm_recv(-1, 5);         // 5 je kod zpravy
    pvm_upkint(&tid, 1, 1);  // cislo delnika
    pvm_upkint(&worker_cnt, 1, 1);  // pocet zpracovanych kousku prace delnika
    printf("Delnik cislo %d k praci pripraven!\n", tid);
  }

  /* -------------------------- vypocet --------------------------------- */

  int cykl;
  for (cykl = 0; cykl < power - 1; cykl++){

    // zaslani matic
    pvm_initsend(PvmDataDefault);   // inicializace bufferu
    pvm_pkint(matrix, grade * grade + 1, 1);    // ulozeni prvni matice
    /* ulozeni druhe matice */
    if (cykl == 0){
      pvm_pkint(matrix, grade * grade + 1, 1);
    }
    else {
      pvm_pkint(rslt, grade * grade + 1, 1);
    }
    pvm_mcast(tids, nproc, 8);  // broadcast zpravy s kodem 8 vsem delnikum

    *rslt = grade;
    rslt_num = 0;

    k = 0;


    // zaslani prace vsem delnikum
```

```
  i = 0;
  while ((i < nproc) && (k < m)) {
    pvm_initsend(PvmDataDefault);   // inicializace bufferu
    /* velikost prace - zadana nebo to, co zbyva */
    work = (k + workSize > m) ? (m - k) : workSize;
    pvm_pkint(&work, 1, 1);
    for (j = 0; (j < work); j++){
      pvm_pkint(&k, 1, 1);            // ulozeni souradnice k pocitani
      k++;
    }
    pvm_send(tids[i], 6);        // zaslani zpravy s kodem 6 (zaslani prace)
    cnt_sent++;
    i++;
  }

  // kdo dokoncil, dostane dalsi ukol
  while (k < m) {               // pokud jsou neprirazene souradnice
    pvm_recv (-1, 7);           // cekani na vysledek
    cnt_received++;
    pvm_upkint (&tid, 1, 1);
    pvm_upkint (&tmp, 1, 1);    // prijmuti poctu vypocitanych souradnic
    for (i = 0; i < tmp; i++){
      pvm_upkint (&j, 1, 1);    // prijmuti cisla souradnice
       // kopirovani hodnoty prvku do pole vysledku na spravnou pozici
      pvm_upkint (rslt + j + 1, 1, 1);
      rslt_num++;
    }

    pvm_upkint(&worker_cnt, 1, 1);  // pocet zpracovanych kousku prace delnika

    // zadani dalsiho ukolu
    pvm_initsend(PvmDataDefault);   // inicializace bufferu

    work = (k + workSize > m) ? (m - k) : workSize; // velikost prace
    pvm_pkint(&work, 1, 1);
    for (j = 0; (j < work); j++){
      pvm_pkint(&k, 1, 1);            // zadani souradnice k praci
      k++;
    }
    cnt_sent++;
    pvm_send(tid, 6);               // zaslani zpravy s kodem 6
  }

  // dobeh rozdelanych ukolu
  while (cnt_sent > cnt_received) {
    pvm_recv (-1, 7);           // cekani na vysledek
    cnt_received++;
    pvm_upkint (&tid, 1, 1);
    pvm_upkint (&tmp, 1, 1);    // pocet vypocitanych souradnic

    for (i = 0; i < tmp; i++){
      pvm_upkint (&j, 1, 1);    // pozice prvku
       // kopirovani hodnoty prvku do pole vysledku
      pvm_upkint (rslt + j + 1, 1, 1);
      rslt_num++;
    }
    pvm_upkint (&worker_cnt, 1, 1);
  }

  // oznameni ukonceni jednoho nasobeni
  for(i = 0; i < nproc; i++) {
    k = -m;
    pvm_initsend(PvmDataDefault);   // inicializace bufferu
    pvm_pkint(&k, 1, 1);            // ulozeni hodnoty k
    pvm_send (tids[i], 6);          // zaslani zpravy s kodem 6
  }

}
/* -------------------------- konec vypoctu ---------------------------- */

// uvolneni delniku
for(i = 0; i < nproc; i++) {
  k = -m;
  pvm_initsend(PvmDataDefault);   // inicializace bufferu
  pvm_pkint(&k, 1, 1);            // ulozeni hodnoty k
  pvm_send (tids[i], 8);          // zaslani zpravy s kodem 6


  pvm_recv (tids[i], 7);          // cekani na ukonceni
  pvm_upkint (&tid, 1, 1);
```

```
    pvm_upkint (&last_rslt, 1, 1);
    pvm_upkint (&worker_cnt, 1, 1);
    printf("Delnik %d konci s hodnotou %d ! Vykonal prace: %d \n",
      tids[i], last_rslt, worker_cnt);
  }

  printf("\nVysledek - %d. mocnina:\n", power);
  // tisk vysledku
  printMatrixWithGrade(rslt);

  // ukonceni cinnosti v PVM
  pvm_exit();

}
```

Worker.c

```
#include "pvm3.h"

main() {

  int mytid;      /*  identifikator procesu delnik  */
  int chief_tid;  /*  identifikator procesu sef  */
  int x_num;      /*  pocet souradnic */
  int result;
  int cnt_work = 0; /*  velikost prace - pocet souradnic */
  int k;          /*  pomocna promenna */

  int *matrix1, *matrix2; /* matice, ktere se maji nasobit */
  int grade;    /* stupen matice */
  int *rslt;    /* vysledek */
  int tmp, sum = 0, m = 0, row, col, i;

  mytid = pvm_mytid();  /* prihlaseni do PVM */

  /* cekani na zpravu od sefa */
  pvm_recv(-1, 4);               /* 4 je 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, 5);     /* kod odpovedi 5 */
  }

  /* delnik pracuje */
  while (1) {

  pvm_recv(chief_tid, 8);               /* 8 - prijem matic */

  /* nacteni stupne matice */
  pvm_upkint(&grade, 1, 1);
  if (grade < 0){
      pvm_initsend(PvmDataDefault);// inic. bufferu
      pvm_pkint(&mytid, 1, 1);    // ulozeni identifikace
      pvm_pkint(&x_num, 1, 1);    // ulozeni ukoncovaciho zpravy
      pvm_pkint(&cnt_work, 1, 1); // ulozeni citace prace
      pvm_send(chief_tid, 7);
      break;                    // zaporna hodnota znamena konec prace
  }

  // ulozeni prvni matice
  matrix1 = (int *) malloc(grade * grade * sizeof(int));
  for (i = 0; i < grade * grade; i++){
    pvm_upkint(&tmp, 1, 1);
    *(matrix1 + i) = tmp;
  }
  // ulozeni druhe matice
  pvm_upkint(&grade, 1, 1);
  matrix2 = (int *) malloc(grade * grade * sizeof(int));
  for (i = 0; i < grade * grade; i++){
    pvm_upkint(&tmp, 1, 1);
    *(matrix2 + i) = tmp;
  }

  while (1) {
```

```
    pvm_recv (chief_tid, 6);   /* cteni  prikazu */
    pvm_upkint (&x_num, 1, 1); /* ulozeni poctu zasilanych souradnic */
    if (x_num < 0) {
      break;                    // zaporna hodnota znamena konec prace
    }

    /* alokovani mista pro vysledek */
    rslt = (int *) malloc(x_num * 2 * sizeof(int));
    /* vypocet hodnot prvku matice */
    for (i = 0; i < x_num; i++){
      pvm_upkint (&tmp, 1, 1);
      *(rslt + 2 * i) = tmp;         /* ulozeni cisla souradnice do vysledku */

      sum = 0;
      row = tmp / grade;
      col = tmp % grade;
      /* vypocet hodnoty prvku */
       for (m = 0; m < grade; m++){
         sum += *(matrix1 + row * grade + m) *  *(matrix2 + m * grade + col);
        }
      *(rslt + 2 * i + 1) = sum; /* ulozeni hodnoty souradnice do vysledku */
    }
    cnt_work++;

    /* odeslani vysledku */
    pvm_initsend(PvmDataDefault);/* inic. bufferu */
    pvm_pkint(&mytid, 1, 1);     /* ulozeni identifikace */
    pvm_pkint(&x_num, 1, 1);     /* pocet souradnic */
    pvm_pkint(rslt, x_num * 2, 1);     /* ulozeni vysledku */
    pvm_pkint(&cnt_work, 1, 1);  /* ulozeni citace prace */
    pvm_send(chief_tid, 7);      /* kod odpovedi 7 */
  }

  }
  /* ukonceni cinnosti */
  pvm_exit();
}
```