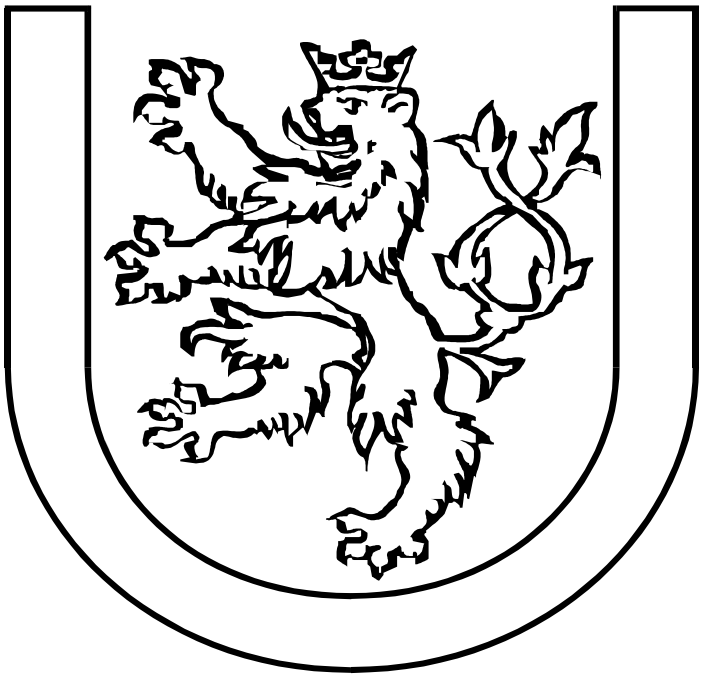
ZÁPADOČESKÁ
UNIVERZITA



Katedra informatiky a výpočetní techniky

Paralelní procesy

Zadání číslo 9

Pavel Tyll
Datum narozeni : 3.6.1984
Email: ptyll@students.zcu.cz

OBSAH:

1. Zadání

Realizujte statistické zpracování dlouhého vektoru reálných čísel (počet položek **n**). Určete aritmetický průměr a směrodatnou odchylku.

Povinné parametry:

- počet vláken (není nutné u MPI, tam se zadává počet vláken při spuštení prostředí)
- velikost kusu práce přidělovaného vláknum (opět není nutné u některých úloh v MPI)
- vektor reálných čísel

Vzorec pro výpočet aritmetického průměru:

`prum = sum(x)/n` *(pro všechny hodnoty x)*

- vzorec pro výpočet směrodatné odchylky:

```
so = sqrt(roz)
roz = (sum(x^2) - (sum(x))^2)/n
```
(pro všechny hodnoty x)

2. Řešení problému

2.1 Programové řešení

Tento problém byl řešen pro dva systémy:

1) Paralelní program pro systém se sdílenou pamětí řešený pomocí programovacího jazyka JAVA.
2) Paralelní program pro systém s distribuovanou pamětí řešený pomocí programovacího jazyka C a systému PVM.

2.2 Řešení vstupních dat programu

Jako vstupní data pro oba programy slouží textový soubor čísel, kde na řádce je vždy umístěno jedno číslo.

3. Popis řešení v Javě

Program byl odladěn na jdk 1.6
Výpis programu v jazyku Java:

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.LinkedList;
import java.util.List;

public class Hlavni {

	/**
	 * @param args
	 * @throws IOException
	 */
	public static void main(String[] args) throws IOException {
		int threads_number = 0; // implicitni pocet vytvorenych vlaken

		if (args.length < 3) {
			System.out
					.println("argumenty : pocet vlaken, velikost kusu prace
pridelovaneho vlaknum, soubor");
			System.exit(0);
		}

		//pocdt vlaken
		threads_number = Integer.valueOf(args[0]).intValue();

		File soub = new File(args[2]); // zjisteni, zda soubor existuje
		if (!soub.isFile()) {
			System.out.println("Soubor neexistuje!");
			return;
		}

		Double[] vektor; //pole nactenych hodnot
		//nacitani hodnot ze souboru
		List<Double> pomList = new LinkedList<Double>();
		FileReader fr = new FileReader(soub);
		BufferedReader br = new BufferedReader(fr);
		String radka;
		while ((radka = br.readLine()) != null) {
			pomList.add(Double.valueOf(radka));
		}
		br.close();
		fr.close();
```

```
                vektor=new Double[pomList.size()];
                 pomList.toArray(vektor);//ziskani hodnot do pole z arraylistu

                int double_number = vektor.length; //pocet hodnot

                if (double_number < threads_number) {
                        System.out.println("Pocet vlaken je vyssi nez pocet cisel!");
                        return;
                }

                int double_per_thread = Integer.valueOf(args[1]).intValue(); //pocet
zpracovavanych hodnot pro vlakno

                VypocteneHodnoty vh = new VypocteneHodnoty();

                Zpracovavani[] threadsfield = new Zpracovavani[threads_number];

                // vytvareni jednotlivych vlaken
                int od = 0;
                for (int i = 0; i < threads_number; i++) {
                        threadsfield[i] = new Zpracovavani(double_per_thread, od, vektor,
                                        vh);
                        threadsfield[i].start();
                        od += double_per_thread;
                        if ((od + double_per_thread) > double_number) {
                                double_per_thread=(int) (double_number-od);
                        }
                }

                // cekani na ukonceni vsech vlaken
                for (int i = 0; i < threads_number; i++) {
                        try {
                                threadsfield[i].join();
                        } catch (InterruptedException e) {
                                System.out.println("Thread is interrupted!");
                        }
                }

                double avg = vh.getSum() / double_number;
                double roz = (vh.getSumX2() - (vh.getSum() * vh.getSum()))
                                / double_number;
                double so = Math.sqrt(roz);

                System.out.println("Prumer je: " + avg);
                System.out.println("Smerodatna odchylka je: " + so);
        }
}
```

```
public class VypocteneHodnoty {
	private double sum;
	private double sumX2;

	public VypocteneHodnoty() {
		this.sum=0;
		this.sumX2=0;
	}

	//pricteni do celkovych vysledku
	public synchronized void pricti(double sumx, double sumx2){
		sum+=sumx;
		sumX2+=sumx2;
	}

	public double getSum() {
		return sum;
	}

	public double getSumX2() {
		return sumX2;
	}
}


public class Zpracovavani extends Thread{

	int pocetCisel;
	int od;
	Double[] vektor;
	VypocteneHodnoty vystup;

	public Zpracovavani(int pocetCisel, int od, Double[] vektor,
			VypocteneHodnoty vystup) {
		super();
		this.pocetCisel = pocetCisel;
		this.od = od;
		this.vektor = vektor;
		this.vystup = vystup;
	}

	public void run() {
		double sumx=0;
		double sumx2=0;

		//pocitani
		for (int i=0;i<pocetCisel;i++){
			sumx+=vektor[od+i];
			sumx2+=(vektor[od+i]*vektor[od+i]);
		}

		vystup.pricti(sumx, sumx2);
	}

}
```

4. Popis řešení v PVM

zdrojový kód :

Farmer :

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

#define LINE   1000


main(int argc, char *argv[]) {
  FILE *soubor;          /* vstupni soubor */
  int mytid;       /*  identifikator procesu sef  */
  int nproc;       /*  pocet delniku */
  int i;           /*  pomocne promenne */
  int vytvoreno = 0;  /*  pocet uspesne vytvorenych delniku */
  int* tidc;       /* pomocny ukazatel pro vytvareni delniku */
  int item_per_worker;   /*pocet cisel na delnika */
  int pointer = 0;       /*ukazatel na pozici v souboru */
  double suma=0;         /* celkovy soucet */
  double qsuma=0;        /* celkovy soucet mocnin */
  double avg;       /* celkovy prumer */
  double odchylka;  /* smerodatna odchylka */
  int items_number=0;         /* pocet cisel v souboru */
  double presult;        /* castecny souset od delnika */
  char str[LINE];   /* pomocny retezec na cteni ze souboru */
  int *wtids;            /* pole id delniku */
  int ch; /* pomocna promena pro cteni ze souboru */


  //zjisteni argumentu
  if (argc != 4)
  {
        printf("Argumenty : pocet delniku, velikost kusu
pridelovane prace, soubor\n");
        exit(1);
  }

  //pocet cisel na delnika
  item_per_worker = atoi(argv[2]);

  //pocet procesu
  nproc= atoi(argv[1]);

  //alokace vektoru delniku
  if ((wtids = (int *)malloc(nproc * sizeof(int))) == NULL)
  {
     fprintf(stderr, "Nepodarilo se alokovat pamet!\n");
     exit(1);
  }
```

```
  //zjisteni, zda soubor existuje
  if ((soubor = fopen(argv[3], "r")) == NULL)
  {
        printf("Soubor neexistuje!\n");
        exit(1);
  }

  //zjisteni poctu radku souboru
  while ( (ch = fgetc(soubor)) != EOF )
  {
        if ( ch == '\n' )
        {
          ++items_number; // Zvyseni poctu radku //
        }
 }
  fclose(soubor);

  double *pole;
  if ((pole = (double *)malloc(items_number * sizeof(double)))
== NULL)
  {
     fprintf(stderr, "Nepodarilo se alokovat pamet!\n");
     exit(1);
  }



  //nacitani hodnot ze souboru
  i=0;
  soubor = fopen(argv[3], "r");
  while(fgets(str, LINE, soubor) != NULL)
  {
        pole[i]=atof(str);
        i++;
  }

  fclose(soubor);







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

  /* vytvoreni delniku */

  tidc=wtids;
  while (nproc>vytvoreno) {
   if (pvm_spawn("worker", (char **)0, 0, "", 1, tidc)) {
      vytvoreno++;
      tidc++;
   }
  }
```

```
  /* poslani udaju delnikum */
  for(i=0; i<nproc; i++) {
    pvm_initsend(PvmDataDefault);   /* inicializace bufferu */
    pvm_pkint(&mytid, 1, 1);    /* ulozeni identifikace */
    pvm_pkint(&items_number, 1, 1);    /* pocet cisel */
    pvm_pkdouble(pole,items_number,1); /*pole cisel */
    pvm_pkint(&item_per_worker, 1, 1);    /* kolik jich
zpracovat */
    pvm_pkint(&pointer, 1, 1);    /* odkud zpracovat */
    pvm_send(wtids[i], 4);     /* kod zpravy 4 */
    pointer += item_per_worker;
  }


  /* prijmani vysledku */
  for (i=0; i<nproc; i++)
     {
       pvm_recv(-1, 7);  //byla-li prijata zprava
          pvm_upkdouble(&presult, 1, 1);
          suma += presult;
          pvm_upkdouble(&presult, 1, 1);
          qsuma += presult;
     }

  avg=suma/items_number;
  printf("prumer :%f\n",avg);
  odchylka=(qsuma-(suma*suma))/items_number;
  odchylka=sqrt(odchylka);
  printf("odchylka :%f\n",odchylka);

  /* ukonceni cinnosti v PVM */
  pvm_exit();
}

Worker :
#include "pvm3.h"
#include <stdio.h>
#include <stdlib.h>

#define LINE   1000

main(int argc,char *argv[]) {
  int mytid;       /*  identifikator procesu delnik     */
  int chief_tid;  /*  identifikator procesu sef        */
  double suma=0;    /*  souctova promenna                 */
  double qsuma=0;   /*  soucet mocnin                     */
  int from;             /* od jakeho prvku pocitat           */
  int number;      /* kolik prvku pocitat               */
  int zaznamu;    /* pocet nactenych hodnot ze souboru */

  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 */
  pvm_upkint(&zaznamu, 1, 1);    /* zjisti pocet hodnot */
  double pole[zaznamu];
  pvm_upkdouble(pole,zaznamu,1);
  pvm_upkint(&number, 1, 1);  /* zjisti prvku pocitat */
  pvm_upkint(&from, 1, 1);  /* zjisti od jakeho prvku */


  int i=0;
  for (i=from;i<(number+from);i++)
  {
    if(i>zaznamu) /* kdyz je i vetsi nez celkovy pocet prvku
*/
    {
      break;
    }
    suma += pole[i];
       qsuma = qsuma + pole[i] * pole[i];
  }

  /* vytvoreni a poslani zpravy */
  pvm_initsend(PvmDataDefault);/* inic. bufferu */
  pvm_pkdouble(&suma, 1,  1);
  pvm_pkdouble(&qsuma, 1,  1);
  pvm_send(chief_tid, 7);     /* kod odpovedi 7 */

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

5. Uživatelský manuál

Java :
Přeložení programu pomocí příkazu javac *.java v adresáři se zdrojovými kódy
Spuštění programu pomocí příkazu java Hlavni
Argumenty : pocet vlaken, velikost kusu prace pridelovaneho vlaknum, soubor
př: java Hlavni 3 4 vstup.txt

PVM:
Přeložení programu pomocí příkazu make v adresáři se zdrojovými kódy. Předpokládá nastavenou proměnou PVM_ROOT (na Hydře /usr/lib/pvm3)
Spuštění programu pomocí příkazu farmer
Argumenty : pocet vlaken, velikost kusu prace pridelovaneho vlaknum, soubor
př: farmer 3 4 vstup.txt

6. Příkazy pro PVM

int pvm_spawn (char *task, char **argv, int flag, char *where, int ntask, int *tids);

Vytvoří a spustí ntask procesů podle programu task a předá jim parametry uložené v argv (obvykle ntask=l).

Vytvoření a odeslání zprávy

int pvm_initsend (int encoding) -Volání ruší starou a inicializuje novou aktivní vysílací vyrovnávací paměť a určuje způsob kódování. Parametr encoding může mít hodnoty:

PvmDataDefault - data ukládaná do vysílací paměti se kódují (XDR) a jdou tudíž poslat na libovolný stroj v heterogenní síti.

PvmDataRaw - data se nekódují, **PvmDatalnPlace** - data se nekopírují (další ušetření času), tudíž ani nekódují

int pvmjpktype (...) - Množina funkcí pro uložení datového prvku s primitivním typem

int pvm_pkint(int* ip, int nitem, int stride); Zde ip je ukazatel na první prvek pole, nitem je počet prvků pole, které se kopírují do zprávy a parametr stride umožňuje „krokovat" v poli

int pvm_send (int tid, int msgtag) - Odeslání obsahu aktivní vysílací vyrovnávací paměti procesu-příjemci tid s označením zprávy msgtag.

int pvmjncast(int *tids, int ntask, int msgtag) - Odeslání obsahu aktivní vysílací vyrovnávací paměti s kódem msgtag skupině procesů, jejichž tid jsou uloženy v poli tids.

int pvm_bcast(char* group, int msgtag) - Odeslání zprávy - obsahu aktivní vysílací vyr.paměti s kódem msgtag skupině procesů se jm. group.

Příjem a čtení zprávy

int pvm_probe (int tid, int msgtag); - Test přítomnosti zprávy.

int pvm_recv (int tid, int msgtag); - Blokující (tj. volající proces čeká) příjem zprávy typu msgtag od procesu-odesilatele tid. Hodnota tid=-l znamená „libovolný odesilatel".

int pvm_nrecv (int tid, int msgtag)- Neblokující varianta k pvm_recv(). Nebyla li požadovaná zpráva dosud přijata, volání funkce (bez čekání) vrátí hodnotu 0. Je—li zpráva k dispozici, vrací volání bufid (nové) aktivní přijímací vyrovnávací paměti nebo záporný chybový kód.

int pvm_upkint (int* ip, int nitem, int stride) - Čtení dat ze zprávy v aktivní přijímací paměti je realizováno množinou funkcí pvm_upk

Synchronizace procesu

int pvmJbarrier (char *group, int count) - Synchronizace skupiny group procesů na bariéře. Zablokuje volající proces do doby, dokud count procesů patřících do skupiny group nezavolá tutéž funkci.

int pvm_sendsig (int tid, int signum); -Volání zasílá signál číslo signum procesu tid.

Ukončení výpočtu

int pvm_kill (int tid) - Volání likviduje proces s identifikátorem tid. Volající proces pokračuje dál ve výpočtu.

int pvm_exit(void) - Tímto voláním proces opouští PVM (jeho PVMD jej vyřadí z registrace), může ale pokračovat dále ve výpočtu jako normální unixovský proces. Pokud je toto volání použito jako poslední příkaz v programu, proces je po vyřazení z PVM OS likvidován.