Vilniaus Universitetas
Matematikos ir informatikos fakultetas
Programų sistemų katedra

Lygiagrečiojo programavimo

Laboratorinio darbo #3 ataskaita

Autoriai: Monika Gasparaitė, 3 kursas, 4 grupė
Mantas Petrikas, 3 kursas, 5 grupė

Vilnius, 2017

Užduotis

1. Parsisiųskite programos kodą (lab_03_src.cpp). Programa generuoja 256 eilučių atsitiktinių skaičių matricą, kurios eilutės suskirstytos blokais po 64. Pirmojo bloko eilutėse generuojama po 2500, antrojo bloko eilutėse – po 5000, trečiojo bloko eilutės – po 7500 ir paskutiniojo bloko eilutėse – po 10000 atsitiktinių skaičių. Kiekvienos eilutės elementai surikiuojami didėjimo tvarka.
2. Papildykite programą kiekvienos surikiuotos eilutės medianos skaičiavimu.
3. Sudarykite lygiagrečią paskirstytos atminties (MPI) programos versiją. Pagrindinis procesas (master) turi padalinti kiekvieną eilučių bloką visiems procesams (įskaitant ir save) ir surinkti paskaičiuotas kiekvienos eilutės medianas. Procesai-darbininkai (slaves) priima eilučių bloką, surikiuoja kiekvieną eilutę, paskai- čiuoja jos elementų medianą ir siunčia medianą pagrindiniam procesui. Procedūra taikoma kiekvienam eilučių blokui (4 kartus). Pagal galimybes, naudokite MPI kolektyvinės komunikacijos šablonus MPI_Scatter, MPI_Gather ir kt.
4. Vykdykite skaičiavimus naudodami 1, 2 ir 4 procesus, fiksuodami lygiagrečiojo algoritmo pagreitėjimą.
5. Ataskaitoje pateikite:
    - užduoties sąlygą;
    - skaičiavimų laiko vidurkių priklausomybės nuo procesų skaičiaus grafiką (naudojant 1, 2 ir 4 procesus);
    - lygiagrečiojo algoritmo pagreitėjimo priklausomybės nuo procesų skaičiaus grafiką (naudojant 1, 2 ir 4 procesus);
    - MPI programos kodą.

Skaičiuodami algoritmo pagreitėjimą laikykite kad $T_0 = T_1$, t.y., kad nuosekliojo algoritmo vykdymo laikas sutampa su lygiagrečiojo algoritmo vykdymo laiku naudojant vieną procesą.

Gauti rezultatai

Algoritmo kodas buvo išlygiagretintas naudojant Open MPI biblioteką. Norint įvertinti lygiagretaus algoritmo pagreitėjimą, jis buvo po tris kartus leidžiamas prisijungus prie MIF Linux serverio kompiuterio, turinčio 4 branduolius. Paleidžiant algoritmą buvo keičiamas naudojamų procesorių skaičius.

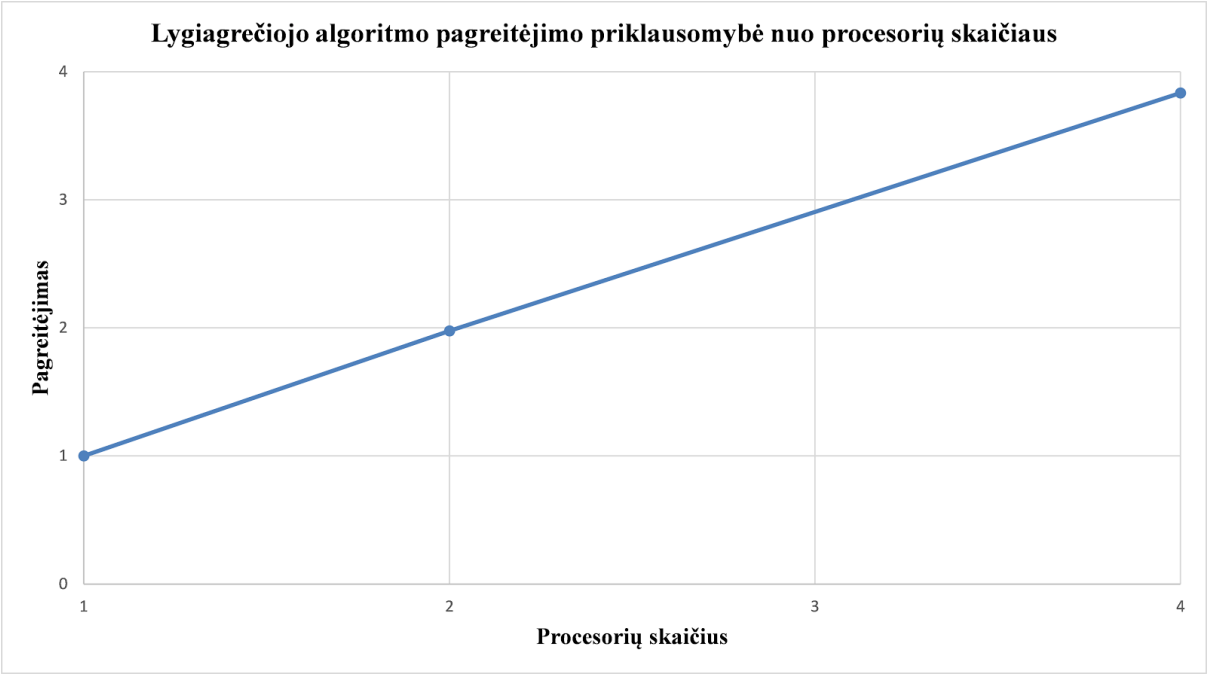
1 lentelėje ir 1 bei 2 grafikuose pateikiamas lygiagrečiojo algoritmo vykdymo laikas bei praktinis pagreitėjimas atlikus bandymus kintant procesorių skaičiui. Praktinis pagreitėjimas apskaičiuotas pagal formulę:

$$S_p = \frac{T_1}{T_p} \ ,$$

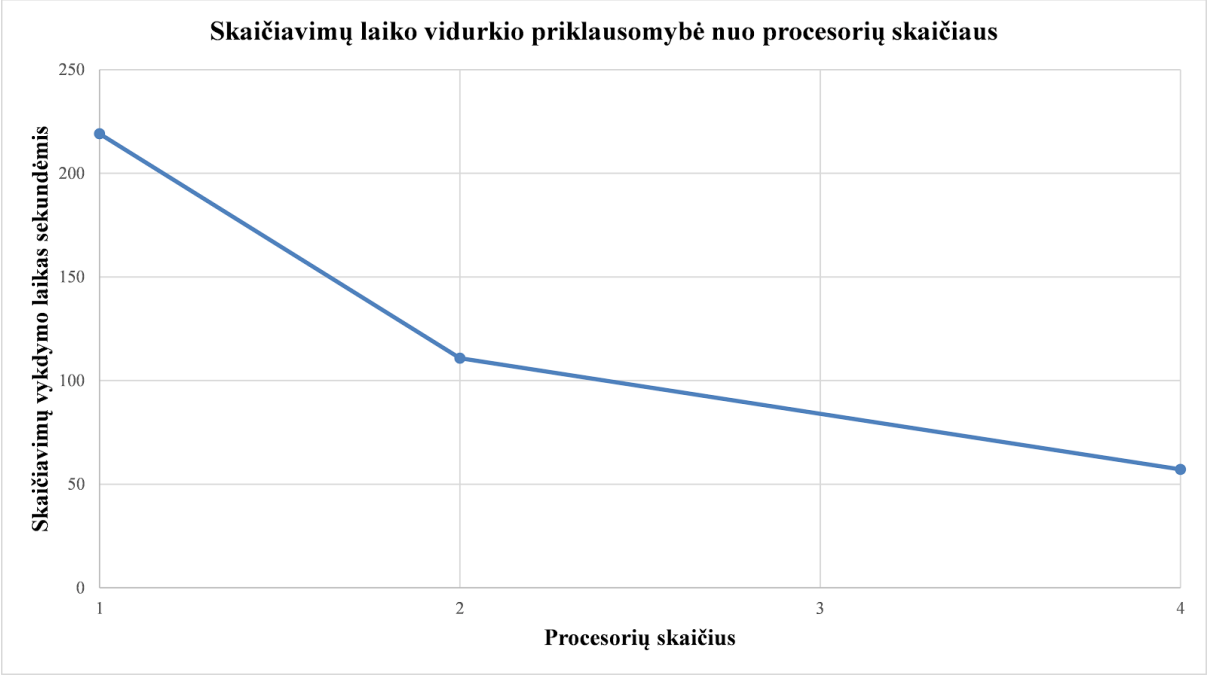
kur T_1 – algoritmo vykdymo laikas naudojant vieną procesorių, T_p – algoritmo vykdymo laikas naudojant p procesorių.

1 lentelė. Gauti rezultatai

Procesorių skaičius	**Vykdymo laikas sekundėmis**				**Pagreitėjimas**
	1 bandymas	**2 bandymas**	**3 bandymas**	**Vidurkis**	
1	219,134	219,180	219,139	219,157	1
2	110,841	110,798	110,85	110,819	1,977
4	57,076	57,244	57,141	57,153	3,834



1 grafikas. Lygiagrečiojo algoritmo pagreitėjimo priklausomybė nuo procesorių skaičiaus



2 grafikas. Skaičiavimų laiko vidurkio priklausomybė nuo procesorių skaičiaus

Išvados

Gauti eksperimentų rezultatai rodo, jog didinant naudojamų procesorių skaičių algoritmo laikas mažėja, o jo pagreitėjimas didėja. Padvigubinus procesorių kiekį vykdymo laikas sumažėja beveik du kartus, o pagreitėjimas atitinkamai padidėja tiek pat kartų. Taip yra dėl to, nes algoritmas yra tinkamai išlygiagretintas, t.y. visi procesoriai yra apkraunami vienodai, nėra didelių prastovų.

Priedai

1 priedas. Lygiagrečiojo algoritmo kodas.

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <time.h>

#define blockCount 4

double GetTime() {
   struct timeval laikas;
   gettimeofday(&laikas, NULL);
   return (double)laikas.tv_sec+(double)laikas.tv_usec/1000000;
}

void genMatrix(int *A, int N, int M) {
   // Clean matrix
   for (int i = 0; i < N * M; i++) {
       A[i] = 0;
   }
   // Generate matrix
   int m = M / blockCount;
   for (int i = 0; i < N; i++) {
       for (int j = 0; j < m; j++)
           A[i * M + j] = (int)((double)rand() / RAND_MAX * 99) + 1;
       if (i > 0 && (i + 1) % (N / blockCount) == 0) m += M / blockCount;
   }
}

void calculateMedian(int M, int oneBlockSize, int procCount, int* workMatrix,
double* median) {
    int t,n;
    for (int rowNo = 0; rowNo < oneBlockSize; rowNo++) {
       n = 0;
       while (workMatrix[rowNo * M + n] != 0 && n < M) {
           n++;
       }
       for (int i = 0; i < n - 1; i++) {
           for (int j = 0; j < n - 1; j++) {
               if (workMatrix[rowNo * M + j] > workMatrix[rowNo * M + j + 1]) {
                   t = workMatrix[rowNo * M + j];
```

```
                    workMatrix[rowNo * M + j] = workMatrix[rowNo * M + j + 1];
                    workMatrix[rowNo * M + j + 1] = t;
                }
            }
        }
        if (n % 2 == 1) {
            median[rowNo] = (double)workMatrix[rowNo * M + (n / 2)];
        } else {
            median[rowNo] = (double(workMatrix[rowNo * M + (n / 2)] +
                workMatrix[rowNo * M + (n / 2) - 1]) / 2);
        }
    }

}

int main(int argc, char* argv[]) {
    double startTime = GetTime();
    int id, procCount;
    int N = 256;
    int M = 10000;
    int *Matrix = new int[N * M];
    double *median;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &id);
    MPI_Comm_size(MPI_COMM_WORLD, &procCount);
    int *workMatrix = new int[N*M/procCount/blockCount];
    int oneBlockSize = N/procCount/blockCount;
    if (id == 0) {
        srand(time(NULL));
        genMatrix(Matrix, N, M);
        median = new double[N];
    } else {
        median = new double[N/procCount/blockCount];
    }
    for (int i=0; i<blockCount; i++) {
        MPI_Scatter(Matrix+N*M/blockCount*i,M *oneBlockSize , MPI_INT, workMatrix,
             M*oneBlockSize, MPI_INT, 0, MPI_COMM_WORLD);
        calculateMedian(M, oneBlockSize, procCount, workMatrix, median);
        MPI_Gather(median, oneBlockSize, MPI_DOUBLE, median+N/blockCount*i,
            oneBlockSize, MPI_DOUBLE, 0, MPI_COMM_WORLD);
    }
    MPI_Finalize();
    double endTime = GetTime();
    if (id == 0) {
        printf("-------------------/n time: %.3f\n-------------------/n" ,
             endTime - startTime);
```

```
    }
}
```