Міністерство освіти і науки України
НТУУ «КПІ ім. Ігоря Сікорського»
Навчально-науковий інститут атомної та теплової енергетики
Кафедра цифрових технологій в енергетиці

Лабораторна робота №4

з дисципліни «Технології паралельних обчислень в енергетичних комплексах»

Тема «Розв’язання систем лінійних алгебричних рівнянь»

Варіант №19

Студента 3-го курсу НН ІАТЕ гр. ТР-12
Ковальова Олександра
Перевірив: ас., Софієнко А. Ю.

КИЇВ 2024

Мета роботи. Розробити паралельну реалізацію алгоритму Гауса з допомогою технології MPI.

Завдання: Розробити паралельну реалізацію алгоритму Гауса в середовищі MPI.

Хід роботи

Програмний код:

```c
#include <math.h>
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/times.h>
#include <sys/time.h>
#include <time.h>
#include <unistd.h>

// MPI Variables
#define MASTER 0
int total_processes;
int rank_process;

// File Variables
char *output_filename;

// Matrix Variables
#define MAXN 5000
int N;
float A[MAXN][MAXN], B[MAXN], X[MAXN];

// Prototypes
void gauss();
void back_substitution();
void finalize();

// Function for random based on time
unsigned int time_seed()
{
    struct timeval t;
    struct timezone tzdummy;

    gettimeofday(&t, &tzdummy);
    return (unsigned int)(t.tv_usec);
}
```

```c
// Command line program parameters
void parameters(int argc, char **argv)
{
    // There must be atleast one argument -- matrix dimension
    if (argc < 2) {
        if (rank_process == MASTER)
        {
            printf("Use: %s <matrix_dimension> [output_filename] [seed]\n", argv[0]);
        }

        finalize();
    }


    // First argument - matrix dimension
    N = atoi(argv[1]);
    if (N < 1 || N > MAXN) {
        if (rank_process == MASTER) {
            printf("N = %i is out of range.\n", N);
        }

        finalize();
    }
    // Printing matrix dimensions
    if (rank_process == MASTER)
    {
        printf("- Matrix Dimension N = %i.\n", N);
    }


    // Second argument - output filename
    if (argc >= 3) {
        if (rank_process == MASTER) {
            int length = strlen(argv[2]);

            output_filename = (char *) malloc(length + 1);
            output_filename = argv[2];

            printf("- Output filename: %s\n", output_filename);
        }
    } else {
```

```
        output_filename = "output.txt";
        if (rank_process == MASTER) {
            printf("- Output filename: %s\n", output_filename);
        }
    }

    // Third argument - Seed
    if (argc == 4) {
        int seed = atoi(argv[3]);
        srand(seed);

        if (rank_process == MASTER)
        {
            printf("- Seed = %i\n", seed);
        }
    } else {
        int seed = time_seed();
        srand(seed);
        if (rank_process == MASTER) {
            printf("- Random seed = %i\n", seed);
        }
    }
}

// Initialize A, B and X
void initializeInput()
{
    int row, col;

    printf("\nInitializing input...\n");

    for (col = 0; col < N; col++)
    {
        for (row = 0; row < N; row++)
        {
            A[row][col] = (float)rand() / 32768.0;
        }
        B[col] = (float)rand() / 32768.0;
        X[col] = 0.0;
    }
}
```

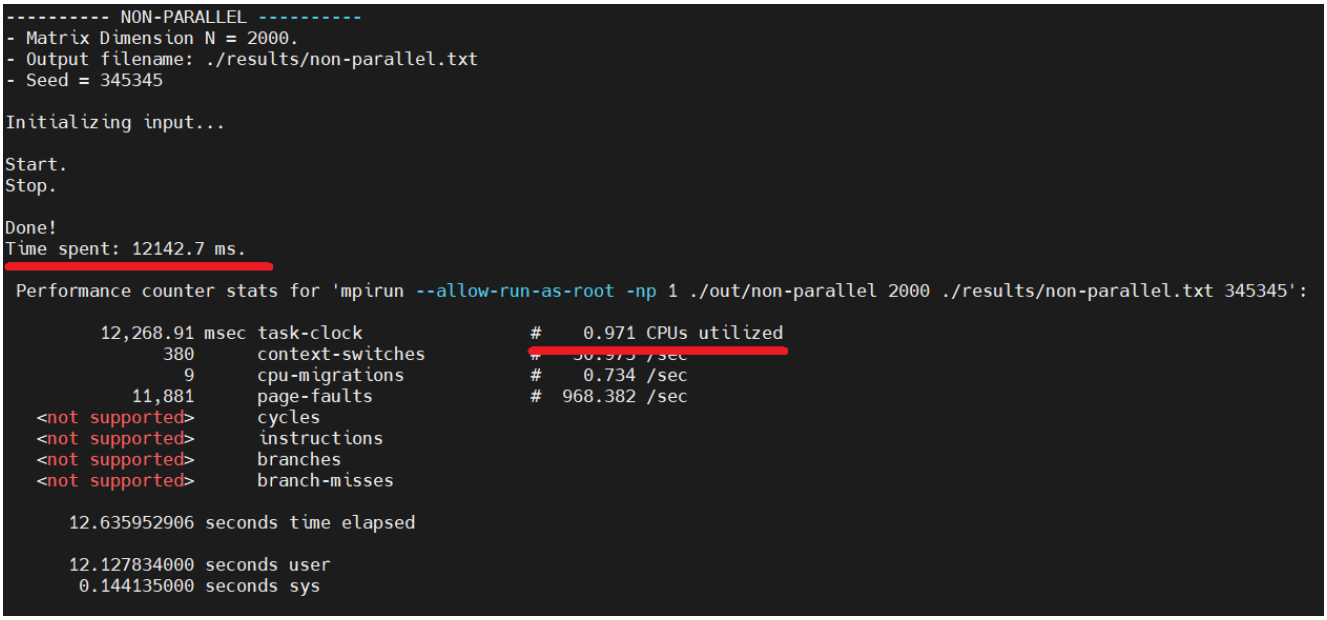
```
// Write input matrices
void write_entries(FILE *filePtr)
{
    int row, col;

    fprintf(filePtr, "A =\n\t");
    for (row = 0; row < N; row++)
    {
        for (col = 0; col < N; col++)
        {
            fprintf(filePtr, "%9.1f%s", A[row][col], (col < N - 1) ? ", " : ";\n\t");
        }
    }
    fprintf(filePtr, "\nB = [");
    for (col = 0; col < N; col++)
    {
        fprintf(filePtr, "%9.1f%s", B[col], (col < N - 1) ? "; " : "]\n");
    }
}

void print_X(FILE *filePtr)
{
    int row;

    fprintf(filePtr, "\nX = [");
    for (row = 0; row < N; row++)
    {
        fprintf(filePtr, "%9.1f%s", X[row], (row < N - 1) ? "; " : "]\n");
    }

    fprintf(filePtr, "\n");
}

int main(int argc, char **argv)
{
    // Variables to calculate time
    struct timeval etstart, etstop;
    struct timezone tzdummy;
    clock_t etstart2, etstop2;
    unsigned long long usecstart, usecstop;
    struct tms cputstart, cputstop;
```

```
    // Initialize MPI
    MPI_Init(&argc, &argv);

    // Number of processes
    MPI_Comm_size(MPI_COMM_WORLD, &total_processes);

    // Process rank
    MPI_Comm_rank(MPI_COMM_WORLD, &rank_process);

    parameters(argc, argv);

    // File to write the results
    if (rank_process == MASTER)
    {
        filePtr = fopen(output_filename, "w+");
        if (filePtr == NULL)
        {
            printf("There was an error while file opening. Exiting program...");

            finalize();
        }

        initializeInput();

        printf("\nStart.\n");
        gettimeofday(&etstart, &tzdummy);
        etstart2 = times(&cputstart);
    }

    // Gaussian elimination
    gauss();

    if (rank_process == MASTER)
    {

        write_entries(filePtr);

        gettimeofday(&etstop, &tzdummy);
        etstop2 = times(&cputstop);
        printf("Stop.\n");
        usecstart = (unsigned long long)etstart.tv_sec * 1000000 + etstart.tv_usec;
        usecstop = (unsigned long long)etstop.tv_sec * 1000000 + etstop.tv_usec;

        // Show result
        print_X(filePtr);

        // Show time results
        printf("\nDone!\n");
        printf("Time spent: %g ms.\n", (float)(usecstop - usecstart) / (float)1000);
    }

    finalize();
}

void gauss()
{

    int norm, row, col, multiplier[N], rownum[N];

    // Process master (0) broadcast data for all processes
    MPI_Bcast(&A[0][0], MAXN * MAXN, MPI_FLOAT, 0, MPI_COMM_WORLD);
    MPI_Bcast(B, N, MPI_FLOAT, 0, MPI_COMM_WORLD);

    for (row = 0; row < N; row++)
    {
        rownum[row] = row % total_processes;
    }

    for (norm = 0; norm < N; norm++)
    {
        // Processor 0 transmits the line of the outer loop being processed for all other processors
        MPI_Bcast(&A[norm][norm], N - norm, MPI_FLOAT, rownum[norm], MPI_COMM_WORLD);
        MPI_Bcast(&B[norm], 1, MPI_FLOAT, rownum[norm], MPI_COMM_WORLD);

        for (row = norm + 1; row < N; row++)
        {
            if (rownum[row] == rank_process)
            {
                multiplier[row] = A[row][norm] / A[norm][norm];
            }
        }
```

```
                multiplier[row] = A[row][norm] / A[norm][norm];
            }
        }
        for (row = norm + 1; row < N; row++)
        {
            if (rownum[row] == rank_process)
            {
                for (col = 0; col < N; col++)
                {
                    A[row][col] = A[row][col] - (multiplier[row] * A[norm][col]);
                }
                B[row] = B[row] - (multiplier[row] * B[norm]);
            }
        }
    }

    back_substitution();
}

void back_substitution()
{
    int row, col;
    if (rank_process == MASTER)
    {
        for (row = N - 1; row >= 0; row--)
        {
            X[row] = B[row];
            for (col = N - 1; col > row; col--)
            {
                X[row] -= A[row][col] * X[col];
            }
            X[row] /= A[row][row];
        }
    }
}

void finalize() {
    MPI_Finalize();

    exit(0);
}
```

Результати:

```
┌─xairaven@host ~/PCT/Lab4
└─$ sudo ./script.sh
----------   PARALLEL   ----------
- Matrix Dimension N = 2000.
- Output filename: ./results/parallel.txt
- Seed = 345345

Initializing input...

Start.
Stop.

Done!
Time spent: 5706.9 ms.

 Performance counter stats for 'mpirun --allow-run-as-root -np 4 ./out/parallel 2000 ./results/parallel.txt 345345':

         19,907.60 msec task-clock                #    3.248 CPUs utilized
             7,241      context-switches          #  363.750 /sec
                26      cpu-migrations            #    1.306 /sec
            90,084      page-faults               #    4.525 K/sec
   <not supported>      cycles
   <not supported>      instructions
   <not supported>      branches
   <not supported>      branch-misses

       6.130077822 seconds time elapsed

      19.653093000 seconds user
       0.386791000 seconds sys
```



Контрольні запитання:

1) **У чому полягає постановка задачі розв'язання системи лінійних рівнянь?**
   Постановка задачі розв'язання системи лінійних рівнянь полягає в формалізації процесу знаходження значень невідомих змінних, які задовольняють усі рівняння, що складають дану систему. У системі лінійних рівнянь кожне рівняння є лінійною функцією змінних, тобто кожне рівняння містить коефіцієнти, які множаться на змінні, а потім додаються або віднімаються. Постановка задачі включає в себе визначення кількості рівнянь та невідомих змінних у системі, а також представлення рівнянь у матричній або векторній формі для подальшого застосування алгоритмів розв'язання, таких як метод Гаусса, метод оберненої матриці тощо. Ця постановка дозволяє визначити, чи існують розв'язки системи, і як їх знайти.

2) **Опишіть схему програмної реалізації паралельного варіанта методу Гауса.**
   У паралельній програмній реалізації методу Гауса використовується розподіл роботи між різними обчислювальними вузлами. Кожен вузол отримує підматрицю або частину даних, над якими він виконує операції методу Гауса локально. Після цього вузли можуть обмінюватись даними для вирішення залежностей між різними частинами матриці. Такий підхід дозволяє розпаралелити обчислення та зменшити час виконання алгоритму, особливо при великих розмірах матриць.