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

Лабораторна робота №3

з дисципліни «Технології паралельних обчислень в енергетичних комплексах»

Тема «Паралельні обчислення для мультикомп’ютерів на основі технології MPI»

Варіант №19

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

КИЇВ 2024

Мета роботи. Опанувати техніку розроблення паралельних програм у мультикомп’ютерному середовищі.

Завдання: Розробити програмну реалізацію задачі *N* тіл в середовищі MPI.

Хід роботи

Програмний код (***послідовний*** варіант):

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

#define SOFTENING 1e-9f

typedef struct
{
    float x;
    float y;
    float z;
    float vx;
    float vy;
    float vz;
} Body;

const char* filePath = "./results/non-parallel.txt";

int iteration = 20;
const float dt = 0.1f;

int bodies;

Body *collection;

void bodyForce(Body *p, int length);
void randomizeBodies(int bodies);
void outputResults(FILE* file, Body *body, double time);
```

```c
int main(int argc, char *argv[])
{
    if (argc != 2)
    {
        fprintf(stderr, "Usage: %s <number_of_bodies>\n", argv[0]);
        return 1;
    }

    bodies = atoi(argv[1]);

    clock_t start_time;

    start_time = clock();

    // Allocating memory for collection
    collection = malloc(sizeof(Body) * bodies);

    randomizeBodies(bodies);

    for (int iter = 0; iter < iteration; iter++)
    {
        bodyForce(collection, bodies);
    }

    double cpu_time_used = ((double) (clock() - start_time)) / CLOCKS_PER_SEC;

    // Write data to the file
    FILE *outputFile = fopen(filePath, "w");
    if (outputFile == NULL)
    {
        fprintf(stderr, "Error opening the file for writing.\n");
        return 1;
    }

    outputResults(outputFile, collection, cpu_time_used);
    fclose(outputFile);

    // Free memory
    free(collection);

    return 0;
}
```

```
void bodyForce(Body *p, int length)
{
    for (int i = 0; i < length; i++)
    {
        float Fx = 0.0f;
        float Fy = 0.0f;
        float Fz = 0.0f;

        for (int j = 0; j < bodies; j++)
        {
            float dx = p[j].x - p[i].x;
            float dy = p[j].y - p[i].y;
            float dz = p[j].z - p[i].z;
            float distSqr = dx * dx + dy * dy + dz * dz + SOFTENING;
            float invDist = 1.0f / sqrtf(distSqr);
            float invDist3 = invDist * invDist * invDist;

            Fx += dx * invDist3;
            Fy += dy * invDist3;
            Fz += dz * invDist3;
        }
        p[i].vx += dt * Fx;
        p[i].vy += dt * Fy;
        p[i].vz += dt * Fz;
    }

    for (int l = 0; l < length; l++)
    {
        p[l].x += p[l].vx * dt;
        p[l].y += p[l].vy * dt;
        p[l].z += p[l].vz * dt;
    }
}
```

```
void randomizeBodies(int bodies)
{
    for (int i = 0; i < bodies; i++)
    {
        collection[i].x = 2.0f * (rand() / (float)RAND_MAX) - 1.0f;
        collection[i].y = 2.0f * (rand() / (float)RAND_MAX) - 1.0f;
        collection[i].z = 2.0f * (rand() / (float)RAND_MAX) - 1.0f;
        collection[i].vx = 2.0f * (rand() / (float)RAND_MAX) - 1.0f;
        collection[i].vy = 2.0f * (rand() / (float)RAND_MAX) - 1.0f;
        collection[i].vz = 2.0f * (rand() / (float)RAND_MAX) - 1.0f;
    }
}

void outputResults(FILE *file, Body *body, double time)
{
    for (int i = 0; i < bodies; i++)
    {
        fprintf(file, "Body %d\n", i);
        fprintf(file, "x  = %f\ny  = %f\nz  = %f\nvx = %f\nvy = %f\nvz = %f\n\n", body[i].x, body[i].y, body[i].z, body[i].vx, body[i].vy, body[i].vz);
        fflush(file);
    }

    fprintf(file, "Time: %.6f seconds", time);
    fflush(file);
}
```

Програмний код (***паралельний*** варіант):

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

#define SOFTENING 1e-9f

typedef struct
{
    float x;
    float y;
    float z;
    float vx;
    float vy;
    float vz;
} Body;

const char* filePath = "./results/parallel.txt";

int rank;
int process;
MPI_Datatype MPIbody;

int iteration = 20;
const float dt = 0.1f;

int bodies;

Body *collection;

void bodyForce(Body *p, int start, int length);
void randomizeBodies(Body *collection, int bodies);
void outputResults(FILE* file, Body *body, double time);
```

```c
int main(int argc, char *argv[])
{
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &process);
    MPI_Type_contiguous(6, MPI_FLOAT, &MPIbody);
    MPI_Type_commit(&MPIbody);

    bodies = atoi(argv[1]);

    // Allocating memory for collection
    collection = malloc(sizeof(Body) * bodies);
    int part = bodies / process;
    int rest = bodies % process;
    int sum = 0;
    int *dspl, *sc;
    dspl = malloc(sizeof(int) * process);
    sc = malloc(sizeof(int) * process);
    double times;

    times = MPI_Wtime();

    for (int i = 0; i < process; i++)
    {
        sc[i] = part;
        if (rest > 0)
        {
            sc[i]++;
            rest--;
        }
        dspl[i] = sum;
        sum += sc[i];
    }

    randomizeBodies(collection, bodies);

    for (int i = 0; i < iteration; i++) {
        MPI_Bcast(collection, bodies, MPIbody, 0, MPI_COMM_WORLD);

        bodyForce(collection, dspl[rank], sc[rank]);

        MPI_Gatherv(&collection[dspl[rank]], sc[rank], MPIbody, collection, sc, dspl, MPIbody, 0, MPI_COMM_WORLD);
    }

    if (rank == 0)
    {
        double timee = MPI_Wtime();
        double end = timee - times;
        // Write data to the file
        FILE *outputFile = fopen(filePath, "w");
        if (outputFile == NULL)
        {
            fprintf(stderr, "Error opening the file for writing.\n");
            return 1;
        }

        outputResults(outputFile, collection, timee);

        fclose(outputFile);
    }

    MPI_Type_free(&MPIbody);
    free(collection);
    free(sc);
    free(dspl);
    MPI_Finalize();
    return 0;
}

void bodyForce(Body *p, int start, int length)
{
    for (int i = start; i < start + length; i++)
    {
        float Fx = 0.0f;
        float Fy = 0.0f;
        float Fz = 0.0f;

        for (int j = 0; j < bodies; j++)
        {
            float dx = p[j].x - p[i].x;
            float dy = p[j].y - p[i].y;
            float dz = p[j].z - p[i].z;
            float distSqr = dx * dx + dy * dy + dz * dz + SOFTENING;
            float invDist = 1.0f / sqrtf(distSqr);
            float invDist3 = invDist * invDist * invDist;

            Fx += dx * invDist3;
            Fy += dy * invDist3;
            Fz += dz * invDist3;
```

```
        }
        p[i].vx += dt * Fx;
        p[i].vy += dt * Fy;
        p[i].vz += dt * Fz;
    }

    for (int l = start; l < start + length; l++)
    {
        p[l].x += p[l].vx * dt;
        p[l].y += p[l].vy * dt;
        p[l].z += p[l].vz * dt;
    }
}

void randomizeBodies(Body *collection, int bodies)
{
    for (int i = 0; i < bodies; i++)
    {
        collection[i].x = 2.0f * (rand() / (float)RAND_MAX) - 1.0f;
        collection[i].y = 2.0f * (rand() / (float)RAND_MAX) - 1.0f;
        collection[i].z = 2.0f * (rand() / (float)RAND_MAX) - 1.0f;
        collection[i].vx = 2.0f * (rand() / (float)RAND_MAX) - 1.0f;
        collection[i].vy = 2.0f * (rand() / (float)RAND_MAX) - 1.0f;
        collection[i].vz = 2.0f * (rand() / (float)RAND_MAX) - 1.0f;
    }
}

void outputResults(FILE *file, Body *body, double time)
{
    for (int i = 0; i < bodies; i++)
    {
        fprintf(file, "Body %d\n", i);
        fprintf(file, "x  = %f\ny  = %f\nz  = %f\nvx = %f\nvy = %f\nvz = %f\n\n", body[i].x, body[i].y, body[i].z, body[i].vx, body[i].vy, body[i].vz);
        fflush(file);
    }

    fprintf(file, "Time: %.6f seconds", time);
    fflush(file);
}
```

Результати роботи програми (**одна машина**):

```
┌─xairaven@host ~/PCT/Lab3
└─$ sudo ./task.sh
----------   PARALLEL   ----------
Time: 7.477339 seconds
CPUs utilized (PROCESSES VARIABLE): 2
CPUs utilized (REAL PERF): 1.921


---------- NON-PARALLEL ----------
Time: 13.852946 seconds
CPUs utilized (REAL PERF): 0.999
┌─xairaven@host ~/PCT/Lab3
└─$
```

Результати роботи програми (**дві машини**):

```
┌─xairaven@host ~/PCT/Lab3
└─$ ./multi_task.sh
---------- HOSTS ----------
127.0.0.1 localhost
127.0.1.1 host
192.168.0.106 testmachine test

--------- HOSTFILE ---------
localhost
testmachine

(Last Result) Time: 7.936976 seconds
```

Контрольні запитання:

1) **Що слід розуміти під паралельною програмою?**
   Паралельна програма – це програма, яка виконується одночасно кількома обчислювальними чи оброблювальними одиницями для прискорення виконання завдань.

2) **Що розуміють в MPI під комунікатором?**
   Комунікатор – це об'єкт, який визначає групу процесів, які можуть взаємодіяти між собою при виконанні паралельної програми.

3) **Як можна організувати приймання повідомлень від конкретних процесів?**
   Використовуючи функцію *MPI_Recv*, ви вказуєте ранг процесу, від якого очікуєте отримати повідомлення.

4) **Як визначити час виконання MPI-програми?**
   Можна використати функцію MPI_Wtime.

5) **У чому відмінність парних і колективних операцій передачі даних?**
   Парні операції передачі даних в MPI виконуються між двома конкретними процесами, тоді як колективні операції об'єднують групу процесів для спільної взаємодії.

6) **Яка функція MPI забезпечує передачу даних від одного процесу всім процесам?**
   MPI_Bcast.

7) **Що розуміють під операцією редукції?**
   Операція редукції в MPI – це колективна операція, яка об'єднує дані з усіх процесів в групі за допомогою певної операції (наприклад, суми чи максимуму).

8) **У яких ситуаціях слід застосовувати бар'єрну синхронізацію?**
   Бар'єрну синхронізацію в MPI застосовують у випадках, коли потрібно забезпечити, щоб всі процеси дочекалися до певного пункту виконання програми перед продовженням виконання.

9) **Які режими передачі даних підтримуються в MPI?**
   MPI підтримує режими передачі даних: синхронний, асинхронний та стандартний.

10) **Як організувати неблокуючий обмін даними в MPI?**
    Для неблокуючого обміну треба використовувати функції *MPI_Isend* та *MPI_Irecv*.

11) Які колективні операції передачі даних передбачені в MPI?

MPI має колективні операції, такі як *MPI_Bcast*, *MPI_Scatter*, *MPI_Gather*, *MPI_Allgather*, *MPI_Reduce*, *MPI_Allreduce* та інші.