Національний технічний університет України

«Київський політехнічний інститут ім. Ігоря Сікорського»

Кафедра цифрових технологій в енергетиці

ЗВІТ   
з виконання лабораторної роботи №5  
з дисципліни «Технології паралельних обчислень в енергетичних комплексах»

«Алгоритми сортування»

Варіант 21

Виконав: студент групи ТР-12 Руденко В.І

Завдання:

Розробити паралельну реалізацію швидкого сортування в середовищі MPI.

Хід Роботи:

**Код програми Завдання**

*#include <mpi.h>*

*#include <stdio.h>*

*#include <stdlib.h>*

*#include <time.h>*

*#include <unistd.h>*

*void swap(int\* arr, int i, int j)*

*{*

*int t = arr[i];*

*arr[i] = arr[j];*

*arr[j] = t;*

*}*

*void quicksort(int\* arr, int start, int end)*

*{*

*int pivot, index;*

*if (end <= 1)*

*return;*

*pivot = arr[start + end / 2];*

*swap(arr, start, start + end / 2);*

*index = start;*

*for (int i = start + 1; i < start + end; i++) {*

*if (arr[i] < pivot) {*

*index++;*

*swap(arr, i, index);*

*}*

*}*

*swap(arr, start, index);*

*quicksort(arr, start, index - start);*

*quicksort(arr, index + 1, start + end - index - 1);*

*}*

*int\* merge(int\* arr1, int n1, int\* arr2, int n2)*

*{*

*int\* result = (int\*)malloc((n1 + n2) \* sizeof(int));*

*int i = 0;*

*int j = 0;*

*int k;*

*for (k = 0; k < n1 + n2; k++) {*

*if (i >= n1) {*

*result[k] = arr2[j];*

*j++;*

*}*

*else if (j >= n2) {*

*result[k] = arr1[i];*

*i++;*

*}*

*else if (arr1[i] < arr2[j]) {*

*result[k] = arr1[i];*

*i++;*

*}*

*else {*

*result[k] = arr2[j];*

*j++;*

*}*

*}*

*return result;*

*}*

*int main(int argc, char\* argv[])*

*{*

*int number\_of\_elements;*

*int\* data = NULL;*

*int chunk\_size, own\_chunk\_size;*

*int\* chunk;*

*FILE\* file = NULL;*

*double time\_taken;*

*MPI\_Status status;*

*int number\_of\_process, rank\_of\_process;*

*int rc = MPI\_Init(&argc, &argv);*

*if (rc != MPI\_SUCCESS) {*

*printf("Error in creating MPI program.\nTerminating...\n");*

*MPI\_Abort(MPI\_COMM\_WORLD, rc);*

*}*

*MPI\_Comm\_size(MPI\_COMM\_WORLD, &number\_of\_process);*

*MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank\_of\_process);*

*if (rank\_of\_process == 0) {*

*if (argc != 3) {*

*exit(-1);*

*}*

*file = fopen(argv[1], "r");*

*if (file == NULL) {*

*exit(-1);*

*}*

*fscanf(file, "%d", &number\_of\_elements);*

*printf("Brain Not Braning",*

*number\_of\_elements);*

*chunk\_size*

*= (number\_of\_elements % number\_of\_process == 0)*

*? (number\_of\_elements / number\_of\_process)*

*: (number\_of\_elements / number\_of\_process*

*- 1);*

*data = (int\*)malloc(number\_of\_process \* chunk\_size*

*\* sizeof(int));*

*for (int i = 0; i < number\_of\_elements; i++) {*

*fscanf(file, "%d", &data[i]);*

*}*

*for (int i = number\_of\_elements;*

*i < number\_of\_process \* chunk\_size; i++) {*

*data[i] = 0;*

*}*

*printf("\n");*

*fclose(file);*

*file = NULL;*

*}*

*MPI\_Barrier(MPI\_COMM\_WORLD);*

*time\_taken -= MPI\_Wtime();*

*MPI\_Bcast(&number\_of\_elements, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);*

*chunk\_size*

*= (number\_of\_elements % number\_of\_process == 0)*

*? (number\_of\_elements / number\_of\_process)*

*: number\_of\_elements*

*/ (number\_of\_process - 1);*

*chunk = (int\*)malloc(chunk\_size \* sizeof(int));*

*MPI\_Scatter(data, chunk\_size, MPI\_INT, chunk,*

*chunk\_size, MPI\_INT, 0, MPI\_COMM\_WORLD);*

*free(data);*

*data = NULL;*

*own\_chunk\_size = (number\_of\_elements*

*>= chunk\_size \* (rank\_of\_process + 1))*

*? chunk\_size*

*: (number\_of\_elements*

*- chunk\_size \* rank\_of\_process);*

*quicksort(chunk, 0, own\_chunk\_size);*

*for (int step = 1; step < number\_of\_process;*

*step = 2 \* step) {*

*if (rank\_of\_process % (2 \* step) != 0) {*

*MPI\_Send(chunk, own\_chunk\_size, MPI\_INT,*

*rank\_of\_process - step, 0,*

*MPI\_COMM\_WORLD);*

*break;*

*}*

*if (rank\_of\_process + step < number\_of\_process) {*

*int received\_chunk\_size*

*= (number\_of\_elements*

*>= chunk\_size*

*\* (rank\_of\_process + 2 \* step))*

*? (chunk\_size \* step)*

*: (number\_of\_elements*

*- chunk\_size*

*\* (rank\_of\_process + step));*

*int\* chunk\_received;*

*chunk\_received = (int\*)malloc(*

*received\_chunk\_size \* sizeof(int));*

*MPI\_Recv(chunk\_received, received\_chunk\_size,*

*MPI\_INT, rank\_of\_process + step, 0,*

*MPI\_COMM\_WORLD, &status);*

*data = merge(chunk, own\_chunk\_size,*

*chunk\_received,*

*received\_chunk\_size);*

*free(chunk);*

*free(chunk\_received);*

*chunk = data;*

*own\_chunk\_size*

*= own\_chunk\_size + received\_chunk\_size;*

*}*

*}*

*time\_taken += MPI\_Wtime();*

*if (rank\_of\_process == 0) {*

*file = fopen(argv[2], "w");*

*if (file == NULL) {*

*exit(-1);*

*}*

*for (int i = 0; i < own\_chunk\_size; i++) {*

*fprintf(file, "%d ", chunk[i]);*

*}*

*fclose(file);*

*printf("Quick-sorted %d in %.6f secs\n",*

*number\_of\_elements, number\_of\_process,*

*time\_taken);*

*}*

*MPI\_Finalize();*

*return 0;*

*}*

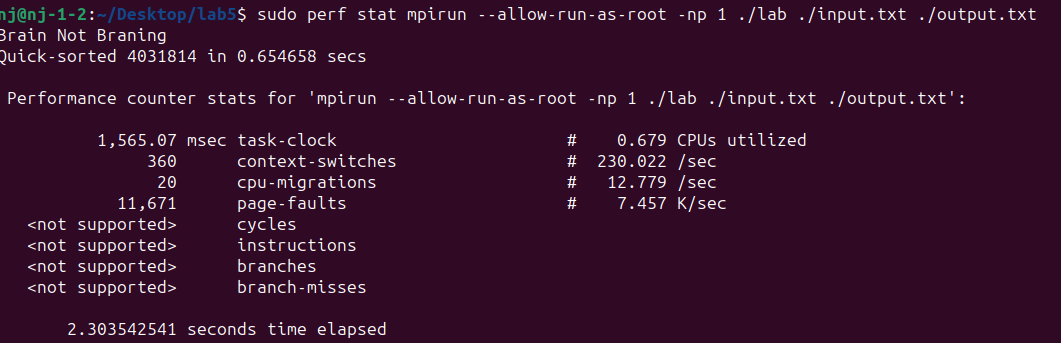


Table 2 Виконання завдання в однопоточному режимі

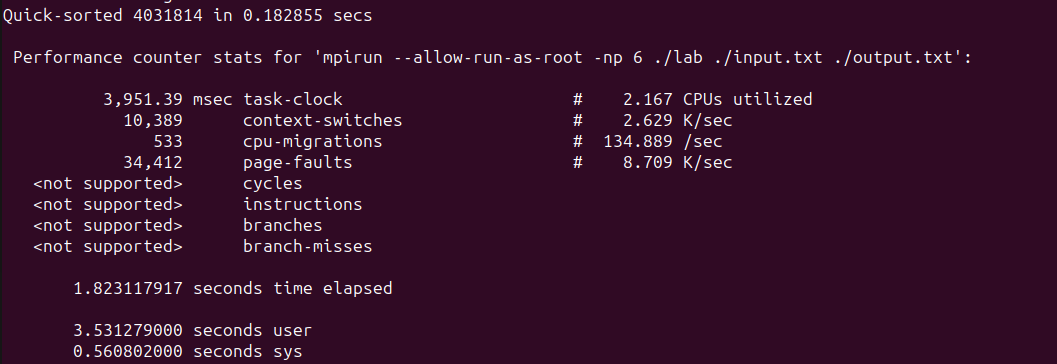


Table 4 Виконання Завдання а багатопоточному режимі

**Контрольні запитання:**

У чому полягає постановка задачі сортування даних?

Постановка задачі сортування даних передбачає розташування елементів послідовності у відповідності з певним критерієм, який може бути зростанням, спаданням або іншим порядком. Мета полягає в організації даних для полегшення їх подальшого пошуку, аналізу або обробки.

Наведіть кілька прикладів алгоритмів сортування. Яка обчислювальна складність наведених алгоритмів?

Деякі приклади алгоритмів сортування: бульбашкою (Bubble Sort), вибором (Selection Sort), вставками (Insertion Sort), злиттям (Merge Sort), швидке сортування (Quick Sort).

Обчислювальна складність залежить від конкретного алгоритму та властивостей вхідних даних, але в основному може бути для простих алгоритмів, таких як бульбашкове сортування, та для більш ефективних, таких як швидке сортування (Quick Sort) чи злиття (Merge Sort).

Яка операція є базовою для задачі сортування даних?

Базова операція для задачі сортування даних – це порівняння між елементами даних та їх обмін місцями для визначення відносного порядку.

У чому суть паралельного узагальнення базової операції задачі сортування даних?

Суть паралельного узагальнення базової операції сортування даних полягає у розділенні вхідних даних між різними процесами для одночасного виконання порівнянь та обміну інформацією між ними з метою швидшого сортування.

Чим є алгоритм парно-непарної перестановки?

Алгоритм парно-непарної перестановки – це метод сортування, де використовується процес рекурсивного поділу та обміну елементів між процесами MPI за допомогою парних та непарних ітерацій, щоб відсортувати дані.

У чому полягає паралельний варіант алгоритму Шелла? Які основні відмінності цього варіанта паралельного алгоритму сортування від методу парно-непарної перестановки?

Паралельний варіант алгоритму Шелла використовує той самий алгоритм, що і класичний алгоритм Шелла, але розділяє вхідні дані між різними процесами MPI для одночасного сортування різних частин послідовності. Основною відмінністю від методу парно-непарної перестановки є те, що в алгоритмі Шелла використовується специфічна послідовність кроків сортування, що може призводити до кращої продуктивності в деяких випадках.

Чим є паралельний варіант алгоритму швидкого сортування?

Паралельний варіант алгоритму швидкого сортування – це метод, який розподіляє вхідні дані між різними процесами MPI для одночасного розділення та сортування частин послідовності, що дозволяє прискорити процес сортування.

Що залежить від правильного вибору головного елемента для паралельного алгоритму швидкого сортування?

Від правильного вибору головного елемента для паралельного алгоритму швидкого сортування залежить ефективність розділення вхідних даних між процесами MPI та швидкість сортування.

Які способи вибору головного елемента можуть бути запропоновані?

Можна використовувати різні способи для вибору головного елемента в паралельному алгоритмі швидкого сортування, такі як вибір випадкового елемента, медіана серед трьох, вибір елемента з фіксованої позиції або використання статистики про вхідні дані для визначення оптимального елемента.

Для яких топологій можуть застосовуватися розглянуті алгоритми сортування?

Розглянуті алгоритми сортування можуть застосовуватися для будь-яких топологій, які дозволяють процесам обмінюватися даними між собою, таких як лінійна, кубічна, гіперкуб, кільце тощо.