КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет кібернетики та комп'ютерних наук

Звіт до лабораторної роботи №1

з дисципліни “Розподілене та паралельне програмування”

на тему “MPI”

Студентки 3 курсу

групи ТТП-32

спеціальності 122 "Комп’ютерні науки"

Дейко Вероніки

Київ –2025

Тема лабораторної роботи: Odd-Even sort.

Завдання: розробити та реалізувати паралельний алгоритм парно-непарного сортування для масиву цілих чисел з використанням бібліотеки MPI (Message Passing Interface).

Реалізація. Паралельний алгоритм парно-непарного сортування було реалізовано мовою програмування C++ з використанням бібліотеки MPI.

Основні етапи реалізації:

1. **Генерація випадкового масиву**: На головному процесі (ранг 0) генерується випадковий масив цілих чисел заданого розміру.
2. **Розподіл даних:** З використанням функції MPI_Scatter, масив рівномірно розподіляється між усіма запущеними MPI-процесами. Кожен процес отримує свою локальну частину масиву.
3. **Локальне сортування**: Кожен процес сортує свою локальну частину масиву.
4. **Фази парно-непарного обміну**: Виконується серія ітерацій (фаз), де на кожній фазі сусідні процеси обмінюються частинами своїх відсортованих масивів. Залежно від парності фази та рангу процесу, відбувається обмін з лівим або правим сусідом. Після обміну кожен процес зливає отриману частину зі своєю та залишає у себе або меншу, або більшу половину злитого масиву. Для обміну даними використовується функція MPI_Sendrecv.
5. **Збір результатів**: Після завершення всіх фаз сортування, відсортовані локальні частини масивів збираються на головному процесі за допомогою функції MPI_Gather.
6. **Вимірювання часу**: За допомогою функцій MPI_Wtime() вимірюється час виконання паралельної частини алгоритму.

Основні методи, що реалізують алгоритм:

- *distributeData* - відповідає за розподіл початкового масиву між усіма процесами MPI.
- *sortLocalArray* - виконує локальне сортування частини масиву, яка належить кожному окремому процесу.
- *exchangeAndMerge* - реалізує логіку обміну та злиття відсортованих частин масивів між сусідніми процесами.
- *performOddEvenPhase* - відповідає за виконання однієї фази парно-непарного сортування.
- *performOddEvenSort* - організовує виконання всіх фаз парно-непарного сортування.
- *gatherResults* - відповідає за збір відсортованих локальних масивів з усіх процесів назад на головний процес.

Виконання програми.

mpiexec -n <p> ParallelProgramming_Odd-Even_Sort.exe <s>, де p - це бажана кількість MPI-процесів, а s - розмір масиву для сортування.

Аналіз результатів.

Кількість потоків	Результат 10^6	Результат 10^8
1	0.0522675	6.05833
2	0.0365201	4.44145
4	0.0331564	4.31909
8	0.0288161	4.91453
10	0.0275051	5.314

Розмір масиву 10^6:

1 потік: Час виконання становить 0.0522675 секунди. Це послідовне виконання алгоритму на одному процесорі.

2 потоки: Час виконання зменшується до 0.0365201 секунди. Це свідчить про певне прискорення завдяки паралелізації на двох процесорах. Частина роботи розподіляється між двома потоками, що дозволяє виконати сортування швидше.

4 потоки: Час виконання продовжує зменшуватися до 0.0331564 секунди. Подальше збільшення кількості потоків дає додаткове прискорення, хоча і менше, ніж при переході від одного до двох потоків.

8 потоків: Час виконання знову зменшується до 0.0288161 секунди. Паралелізація на восьми потоках все ще є вигідною.

10 потоків: Час виконання є найменшим серед усіх варіантів для цього розміру масиву і становить 0.0275051 секунди. Це вказує на те, що для масиву розміром 10^6 оптимальна кількість потоків у даному випадку становить 10.

Розмір масиву 10^8:

1 потік: Час виконання значно більший і становить 6.05833 секунди. Це очікувано, оскільки обробка в 100 разів більшого масиву займає набагато більше часу при послідовному виконанні.

2 потоки: Час виконання зменшується до 4.44145 секунди. Паралелізація на двох потоках дає значне прискорення.

4 потоки: Час виконання продовжує зменшуватися до 4.31909 секунди. Прискорення при переході від двох до чотирьох потоків менше, ніж від одного до двох.

8 потоків: Час виконання зростає до 4.91453 секунди. Це несподіваний результат. Після зменшення часу при збільшенні кількості потоків до 4, подальше збільшення до 8 призводить до його зростання. Це може бути пов'язано з накладними витратами на комунікацію між потоками, які стають більш значущими для великого масиву при більшій кількості потоків.

10 потоків: Час виконання продовжує зростати до 5.314 секунди. Це підтверджує, що для масиву такого розміру подальше збільшення кількості потоків після певного значення призводить до погіршення продуктивності.

Порівняння результатів для різних розмірів масиву:

- Для меншого масиву (10^6) збільшення кількості потоків послідовно призводить до зменшення часу виконання в дослідженому діапазоні (до 10 потоків).
- Для більшого масиву (10^8) спостерігається інша картина. Збільшення кількості потоків спочатку зменшує час виконання, але потім призводить до його зростання. Це демонструє важливість врахування розміру даних при виборі оптимальної кількості потоків для паралельної програми. Для великих обсягів даних накладні витрати на паралелізацію можуть стати суттєвим фактором, що обмежує масштабованість.

```
C:\Users\deyko\source\repos\ParallelProgramming_Odd-Even_Sort\x64\Release>mpiexec -n 1 P
.exe 1000000
Size: 1000000
Process 0 on DESKTOP-VLOMFU1
Time: 0.0522675

C:\Users\deyko\source\repos\ParallelProgramming_Odd-Even_Sort\x64\Release>mpiexec -n 2 P
.exe 1000000
Size: 1000000
Process 0 on DESKTOP-VLOMFU1
Process 1 on DESKTOP-VLOMFU1
Time: 0.0365201
```

```
C:\Users\deyko\source\repos\ParallelProgramming_Odd-Even_Sort\x64\Release>mpiexec -n 4 P
.exe 1000000
Process 3 on DESKTOP-VLOMFU1
Process 2 on DESKTOP-VLOMFU1
Size: 1000000
Process 1 on DESKTOP-VLOMFU1
Process 0 on DESKTOP-VLOMFU1
Time: 0.0331564
```

```
C:\Users\deyko\source\repos\ParallelProgramming_Odd-Even_Sort\x64\Release>mpiexec -n 8 Pa
.exe 1000000
Process 1 on DESKTOP-VLOMFU1
Process 3 on DESKTOP-VLOMFU1
Process 6 on DESKTOP-VLOMFU1
Size: 1000000
Process 0 on DESKTOP-VLOMFU1
Process 7 on DESKTOP-VLOMFU1
Process 4 on DESKTOP-VLOMFU1
Process 5 on DESKTOP-VLOMFU1
Process 2 on DESKTOP-VLOMFU1
Time: 0.0288161

C:\Users\deyko\source\repos\ParallelProgramming_Odd-Even_Sort\x64\Release>mpiexec -n 10 P
t.exe 1000000
Process 6 on DESKTOP-VLOMFU1
Size: 1000000
Process 9 on DESKTOP-VLOMFU1
Process 8 on DESKTOP-VLOMFU1
Process 0 on DESKTOP-VLOMFU1
Process 1 on DESKTOP-VLOMFU1
Process 4 on DESKTOP-VLOMFU1
Process 7 on DESKTOP-VLOMFU1
Process 2 on DESKTOP-VLOMFU1
Process 5 on DESKTOP-VLOMFU1
Process 3 on DESKTOP-VLOMFU1
Time: 0.0275051
```

Спостерігається чітка тенденція до зменшення часу виконання зі збільшенням кількості потоків. Це очікувано, оскільки паралельне виконання дозволяє розділити роботу між кількома процесорами, що *теоретично* має призводити до швидшого завершення обчислень.

Результати демонструють, що використання більшої кількості потоків призводить до скорочення часу виконання, що є головною метою паралельного програмування.

Інколи при більшій кількості потоків збільшується і час виконання, замість очікуваного зменшення. Це може відбуватися через те, що потоки часто повинні обмінюватися даними для виконання завдання. Зі збільшенням кількості потоків зростає і кількість необхідних комунікацій. Час, витрачений на передачу даних між потоками, може стати значним і переважити виграш від паралельних обчислень. У моєму випадку з алгоритмом парно-непарного злиття, кожен процес обмінюється даними зі своїм сусідом на кожній фазі. З більшою кількістю процесів зростає кількість таких обмінів.

Висновок. Результати показують, що ефективність паралелізації залежить як від кількості потоків, так і від розміру оброблюваних даних. Для менших даних збільшення кількості потоків може бути вигідним, тоді як для великих даних існує оптимальна кількість потоків, після якої подальше збільшення може призвести до погіршення продуктивності через зростання накладних витрат

Гітхаб:
https://github.com/DeikoVeronika/ParallelProgramming_Odd-Even_Sort.git