

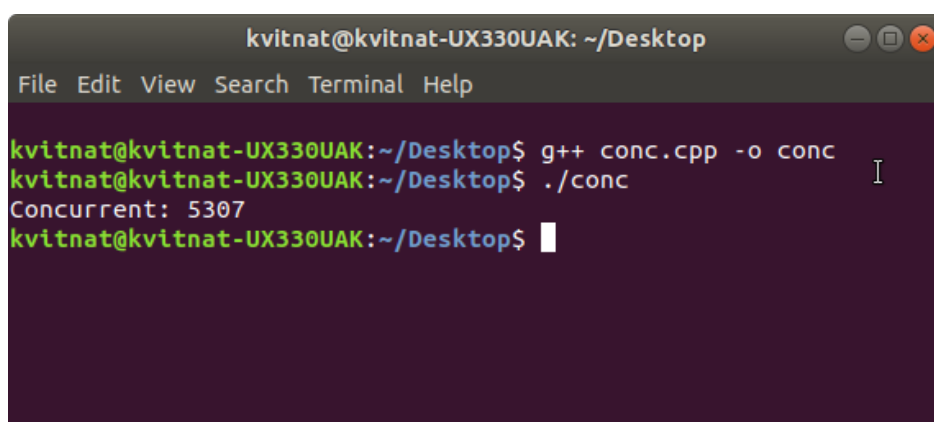
ЗВІТ

до лабораторної роботи
з паралельного програмування
студентки групи ТК-3
Сокол Наталії

В цій лабораторній роботі потрібно було вирішити одну і ту саму задачу трьома різними способами – послідовно, паралельно за допомогою технології OpenMP, та паралельно за допомогою MPI.

Обраний алгоритм – сортування підрахунком, разом зі створенням вхідних даних. Послідовна програма виглядає так:

1. Початок таймеру
2. Створення масиву розміром 100 млн та заповнення його за допомогою генератора випадкових чисел.
3. Виклик функції sort:
 - Пошук мінімального значення та максимального значення
 - Створення і заповнення нулями масиву розміром (макс – мін + 1)
 - Підрахунок скільки раз зустрічається значення в вхідному масиві.
 - Переписування відсортованого масиву.
4. Кінець таймеру
5. Вивід часу роботи на консоль.



```
kvitnat@kvitnat-UX330UAK: ~/Desktop
File Edit View Search Terminal Help

kvitnat@kvitnat-UX330UAK:~/Desktop$ g++ conc.cpp -o conc
kvitnat@kvitnat-UX330UAK:~/Desktop$ ./conc
Concurrent: 5307
kvitnat@kvitnat-UX330UAK:~/Desktop$
```

Рис. 1: Вивід послідовної програми.

Послідовний код працює за 5 секунд. Вивід часу в мілісекундах, щоб можна було порівняти між собою паралельні програми, де різниця в часі дуже маленька. Як можна побачити на рисунку 2, при цьому всі обчислення і робота з даними відбувається на одному з ядер процесора.

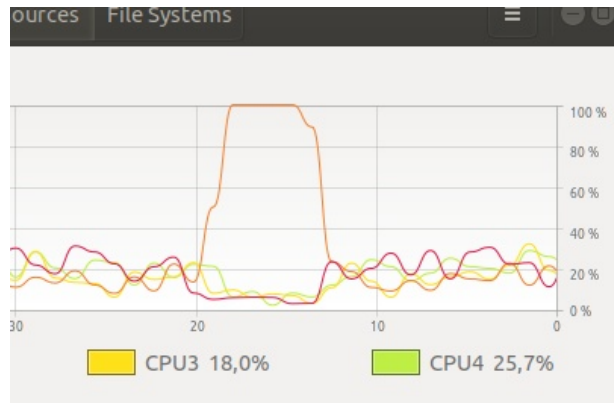


Рис. 2: Графік завантаженості ядер під час виконання послідовної програми.

OpenMP(open multi-processing)

OpenMP – це набір директив компілятора, функцій та змінних для програмування паралельних програм в системах із спільною пам'яттю. Програма, написана з використанням OpenMP має один головний потік, який час від часу породжує додаткові залежні потоки.

В цьому алгоритмі перше, що можна легко розпаралелити – це цикл, що заповнює вхідний масив випадковими числами. Ітерації циклу не залежать одна від одної, тому їх можна виконувати в різних потоках. Це робиться за допомогою директиви компілятора `#pragma omp parallel for`. По замовчуванню буде створено стільки потоків, скільки ядер є на комп'ютері, де запускається програма.

Після того, як масив було заповнено, викликається функція `sort`, яка спочатку має знайти мінімальне та максимальне значення з усього масиву. Ці дії логічно організувати, як паралельні секції, що і було зроблено.

Далі програма знов починає виконуватись послідовно. Цикл, що рахує кількості кожного елементу, теж можна було б розпаралелити, але потрібно було б обробляти ситуацію, коли два різні потоки намагаються змінити значення одного елементу масиву `counts`. Для цього можна використати директиву `atomic`, але такий код працює навіть повільніше, ніж послідовний. Якщо ж не використовувати `atomic`, то в посортованому масиві зустрічаються помилки.

```
kvitnat@kvitnat-UX330UAK: ~/Desktop
File Edit View Search Terminal Help

kvitnat@kvitnat-UX330UAK:~/Desktop$ g++ -fopenmp omp.cpp -o omp
kvitnat@kvitnat-UX330UAK:~/Desktop$ ./omp
OpenMP: 3828
kvitnat@kvitnat-UX330UAK:~/Desktop$
```

Рис. 4: Вивід OpenMP програми



Рис. 3: Всі 4 ядра завантажені під час виконання OpenMP програми.

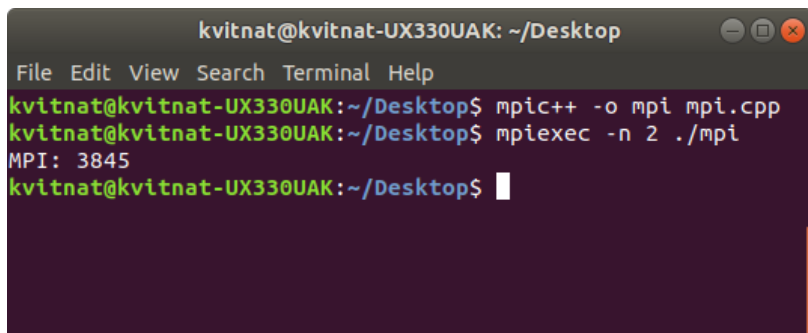
MPI(Message Passing Interface)

MPI – це інтерфейс передачі повідомлень між процесами. Розроблений переважно для систем з розподіленою пам'яттю, на відміну від OpenMP. MPI-програма має фіксовану кількість процесів, що вказується при компіляції.

Тут розпаралелюємо ті самі дії – починаємо з заповнення масиву. Залежно від кількості процесів, кожний генерує якусь частину чисел в додатковий масив `partArray`, і потім за допомогою функції `MPI_Allgather` з цих частин формується один великий масив з усіма даними.

Далі в функції сортування, пошук мінімального та максимального елементів виконується в двох різних потоках, так само, як і в OpenMP програмі. Тут процес з рангом 0 шукає мінімальне значення, а процес з рангом 1 – максимальне. Далі знайдені значення за допомогою функції `MPI_Bcast` передаються всім іншим процесам, і сортування продовжується в нульовому процесі. Після того як масив посортовано, нульовий процес в головній функції виводить час, що було витрачено на генерацію даних та сортування.

Найкращі результати MPI-програма видає на двох процесорах, так як копіювання даних теж займає час, а частина роботи розподілена тільки на два процеси.



```
kvitnat@kvitnat-UX330UAK: ~/Desktop
File Edit View Search Terminal Help
kvitnat@kvitnat-UX330UAK:~/Desktop$ mpic++ -o mpi mpi.cpp
kvitnat@kvitnat-UX330UAK:~/Desktop$ mpirun -n 2 ./mpi
MPI: 3845
kvitnat@kvitnat-UX330UAK:~/Desktop$
```

Рис. 5: Вивід MPI-програми

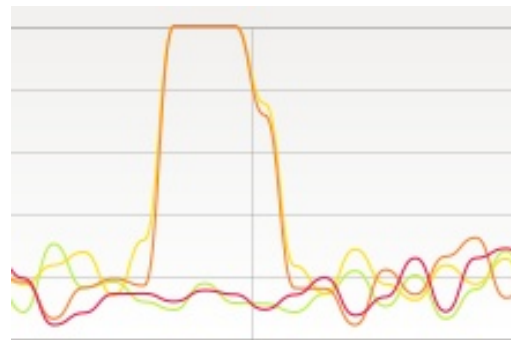


Рис. 6: Завантажені два ядра процесора під час виконання MPI-програми.

Таким чином, обидві паралельні програми працюють швидше за послідовну. На цьому прикладі OpenMP програма була ефективнішою, так як вона не витрачає зайвих ресурсів за рахунок спільної пам'яті процесів, на відміну від MPI-програми, де в кожному процесі зберігається величезний масив вхідних даних. При написанні паралельних програм потрібно враховувати фізичні особливості системи, для якої створюється програма. Технології OpenMP та MPI можливо комбінувати між собою, отримуючи найвищу ефективність від складних систем.