ІП-14 Сергієнко Юрій

Варіант 28

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 121 Інженерія програмного забезпечення
Освітня програма Інженерія програмного забезпечення інформаційних систем
Навчальна дисципліна – «Технології паралельних обчислень»

Екзаменаційний білет № 28

1. Проєктування паралельних програм.
2. Синхронізовані методи.
3. З використанням колективних методів обміну повідомленнями MPI напишіть фрагмент коду, який виконує розсилку фрагментів масиву A в процеси-воркери, виконує сортування переданих масивів в процесах-воркерах та отримує в процесі-майстрі перші 3 значення відсортованих масивів або весь масив, якщо значень в ньому менше 3. У масиві A зберігаються об'єкти, для яких визначений компаратор.
4. Напишіть код для створення, запуску на виконання та завершення роботи пулу потоків, 1000 підзадач якого здійснюють без упину операції збільшення та зменшення рахунку банківського клієнта на задану суму. Операція зменшення суми рахунку затримується для виконання, якщо у клієнта недостатньо коштів. При зменшенні рахунку вдвічі від його початкового значення робота пулу завершується.

Затверджено на засіданні кафедри ІПІ
Протокол № 5 від « 8 » листопада 2023 р.

Зав. кафедри інформатики та програмної інженерії _____ Едуард ЖАРІКОВ

1

Проєктування паралельних програм

Потрібно визначити, чи задачу можливо розпаралелити. Якщо так, перейдемо далі.

Необхідно обрати спосіб розпаралелювання.

1. В першу чергу, на це впливає рід задачі: у випадку множення матриць обидва варіанти програмного паралелізму підходять, адже матрицю можна множити як одночасно по процесах системи (MPI), так і почастинно в тасках (пули). В цілому, для задач, які легко можуть бути розбити на незалежні частини для обрахування (наприклад обробка зображень з CUDA), ідеально підходить data parallelism. У випадку завдань, для яких необхідна синхронізація частин

(сортування, алгоритми пошуку шляхів графа) ефективнішим буде task parallelism.

2. Залежимо від системи: у випадку системи з багатьома процесорами, data parallelism буде найкращим варіантом, коли як у системі з багатоядерним процесором виграє task parallelism.

Вибір мови програмування

Бажано обрати мову, яка ефективно справляється з паралельними обчисленнями. Наприклад java, адже вона має вбудовані бібліотеки для розробки паралельних програм.

2

Синхронізовані методи

Синхронізовані методи використовуються для захисту спільних ресурсів від одночасного доступу кількох потоків. Це допомагає уникнути проблем, пов'язаних з memory consistency error (інакше race condition).

Викликаючи даний метод декількома потоками одночасно, ключове слово synchronized гарантує, що вхідний об’єкт може бути оброблений лише одним потоком одночасно.

Потік, який намагається викликати синхронізований метод, перевіряє, чи заблоковано об’єкт. Якщо об’єкт заблокований, потік переходить у стан Blocked і очікує на розблокування об’єкта. Планувальник потоків, отримавши сигнал про зміну стану об’єкта, активізує потоки, що чекають на його розблокування. З цих потоків тільки один зможе захопити об’єкт, коли він стане доступним.

3

```
#include <mpi.h>
#include <iostream>
#define N 12
#define THRESHOLD 3
#define MASTER 0

void compute(int numtasks, int taskid) {
    int *a = (int *) malloc(N * sizeof(int));
    int *a_p = (int *) malloc(N * sizeof(int));
    int *result = (int *) malloc(THRESHOLD * numtasks * sizeof(int));
```

```
if (taskid == MASTER) {
    for (int i = 0; i < N; i++) {
        a[i] = N - i;
        printf("%d\t", a[i]);
    }
printf("\n");
}

int *sizes = (int *) malloc(numtasks * sizeof(int));
int *offsets = (int *) malloc(numtasks * sizeof(int));
int extra = N % numtasks;
int avgsize = N / numtasks;

int offset = 0;
for (int i = 0; i < numtasks; i++) {
    int size = (i < extra) ? avgsize + 1 : avgsize;
    sizes[i] = size;
    offsets[i] = offset;
    offset += sizes[i];
}

MPI_Scatterv(
    a, sizes, offsets, MPI_INT,
    a_p, sizes[taskid], MPI_INT,
    0, MPI_COMM_WORLD
);

for (int i = 0; i < sizes[taskid]; i++) {
    for (int j = 0; j < sizes[taskid] - 1; j++) {
        if (a_p[j] > a_p[j + 1]) {
            int temp = a_p[j];
            a_p[j] = a_p[j + 1];
            a_p[j + 1] = temp;
        }
    }
}

int endsize = sizes[taskid] >= THRESHOLD ? THRESHOLD : sizes[taskid];
int *localSorted = (int *) malloc(endsize * sizeof(int));
for (int i = 0; i < endsize; i++) {
    localSorted[i] = a_p[i];
}

for (int i = 0; i < numtasks; i++) {
    sizes[i] = endsize;
}

offsets[0] = 0;
for (int i = 1; i < numtasks; i++) {
    offsets[i] = offsets[i - 1] + sizes[i - 1];
}

MPI_Gatherv(
    localSorted, endsize, MPI_INT,
    result, sizes, offsets, MPI_INT,
    0,MPI_COMM_WORLD
);

if (taskid == MASTER) {
    for (int i = 0; i < numtasks; i++) {
```

```
            printf("Received from process: %d\n", i);
            for (int j = 0; j < THRESHOLD; j++) {
                if (result[(i * THRESHOLD) + j] < N)
                    printf("%d ", result[(i * THRESHOLD) + j]);
            }
            printf("\n");
        }
    }
    MPI_Finalize();
}

int main(int argc, char *argv[]) {
    int numtasks, taskid;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
    MPI_Comm_rank(MPI_COMM_WORLD, &taskid);
    compute(numtasks, taskid);
    return 0;
}
```

```
PS C:\Users\Yurii\Documents\jaba\kpió\exam_tpo\task_mpi> mpiexec -n 4 ./cmake-build-debug/task_mpi
12      11      10      9       8       7       6       5       4       3       2       1
Received from process: 0
10 11 12
Received from process: 1
7 8 9
Received from process: 2
4 5 6
Received from process: 3
1 2 3
PS C:\Users\Yurii\Documents\jaba\kpió\exam_tpo\task_mpi>
```

```
PS C:\Users\Yurii\Documents\jaba\kpió\exam_tpo\task_mpi> mpiexec -n 5 ./cmake-build-debug/task_mpi
12      11      10      9       8       7       6       5       4       3       2       1
Received from process: 0
10 11
Received from process: 1
7 8 9
Received from process: 2
5 6
Received from process: 3
3 4
Received from process: 4
1 2
PS C:\Users\Yurii\Documents\jaba\kpió\exam_tpo\task_mpi>
```

4

```
package org.example;

import java.util.Random;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.atomic.AtomicInteger;
```

```java
public class Main {
    public static void main(String[] args) {
        ExecutorService executors = Executors.newFixedThreadPool(8);
        int startBalance = 100000;

        AtomicInteger balance = new AtomicInteger(startBalance);
        Random random = new Random();
        for (int i = 0; i < 1000; i++) {
            executors.submit(() -> {
                while (balance.get() > startBalance / 2) {
                    if (balance.get() > 0)
                        balance.addAndGet(random.nextInt(100) - 99);

                    balance.addAndGet(random.nextInt(100));
                    System.out.println(balance);
                }
                System.out.println("Got upper peek with " + balance);
            });
        }

        executors.shutdown();

        try {
            if (!executors.awaitTermination(3, TimeUnit.SECONDS)) {
                executors.shutdownNow();
            }
        } catch (InterruptedException ex) {
            executors.shutdownNow();
        }

        System.out.println("Final balance is " + balance);
    }
}
```

```
Got lower peek with 49981
Got lower peek with 49981
Got lower peek with 49981
Got lower peek with 49981
Got lower peek with 49981
Got lower peek with 49981
Got lower peek with 49981
Final balance is 49981

Process finished with exit code 0
```