Bài tập CS112: Quy hoạch động

Nhóm : 2
Nguyễn Thiện Nhân 23521083
Trần Vinh Khánh 23520726

Bài 1

Code :

```python
import math
import time
import multiprocessing

# Hàm kiểm tra số nguyên tố
def is_prime(n):
    if n <= 1:
        return False
    if n == 2 or n == 3:
        return True
    if n % 2 == 0 or n % 3 == 0:
        return False
    for i in range(5, int(math.sqrt(n)) + 1, 6):
        if n % i == 0 or n % (i + 2) == 0:
            return False
    return True

# Kiểm tra số nguyên tố tuần tự
def check_prime_sequential(n):
    return is_prime(n)

# Hàm phụ trợ để kiểm tra song song
def check_prime_parallel_range(start, end, n, result, idx):
    for i in range(start, end):
        if n % i == 0:
            result[idx] = False
            return
    result[idx] = True

# Kiểm tra số nguyên tố song song
def check_prime_parallel(n):
    processes = []
    num_processes = 4  # Số tiến trình song song
    step = int(math.sqrt(n)) // num_processes
    result = multiprocessing.Array('b', num_processes)

    # Chạy song song các tiến trình
    for i in range(num_processes):
        start = i * step + 2
        end = (i + 1) * step + 2 if i != num_processes - 1 else int(math.sqrt(n)) + 1
        p = multiprocessing.Process(target=check_prime_parallel_range, args=(start, end, n,
        ↪  result, i))
        processes.append(p)
        p.start()
```

```
44
45      # Chờ tất cả tiến trình kết thúc
46      for p in processes:
47          p.join()
48
49      # Kiểm tra kết quả
50      return all(result)
51
52  # Hàm kiểm tra số nguyên tố song song vs. tuần tự
53  def compare_times(n):
54      # Kiểm tra thời gian tuần tự
55      start = time.time()
56      sequential_result = check_prime_sequential(n)
57      sequential_time = time.time() - start
58      print(f"Sequential check result for {n}: {sequential_result}")
59      print(f"Sequential time: {sequential_time:.6f} seconds")
60
61      # Kiểm tra thời gian song song
62      start = time.time()
63      parallel_result = check_prime_parallel(n)
64      parallel_time = time.time() - start
65      print(f"Parallel check result for {n}: {parallel_result}")
66      print(f"Parallel time: {parallel_time:.6f} seconds")
67
68  # Các test case
69  test_cases = [
70      1000000000000091,
71      10000000000000099,
72      100000000000000049
73  ]
74
75  # Kiểm tra các test case
76  for n in test_cases:
77      print(f"Checking number {n}...")
78      compare_times(n)
79      print("-" * 50)
```

Kết quả :

```
Checking number 1000000000000091...
Sequential check result for 1000000000000091: True
Sequential time: 5.247355 seconds
Parallel check result for 1000000000000091: True
Parallel time: 8.209198 seconds
--------------------------------------------------
Checking number 10000000000000099...
Sequential check result for 10000000000000099: True
Sequential time: 8.156910 seconds
Parallel check result for 10000000000000099: True
Parallel time: 17.901273 seconds
--------------------------------------------------
Checking number 100000000000000049...
Sequential check result for 100000000000000049: True
Sequential time: 14.557678 seconds
Parallel check result for 100000000000000049: True
Parallel time: 46.678322 seconds
--------------------------------------------------
```

Bài 2

```python
import random
import time
import concurrent.futures

# Nhân ma trận tuần tự
def matrix_multiply_sequential(A, B):
    rows_A = len(A)
    cols_A = len(A[0])
    rows_B = len(B)
    cols_B = len(B[0])

    # Khởi tạo ma trận kết quả với giá trị 0
    result = [[0] * cols_B for _ in range(rows_A)]

    # Nhân ma trận tuần tự
    for i in range(rows_A):
        for j in range(cols_B):
            for k in range(cols_A):
                result[i][j] += A[i][k] * B[k][j]

    return result

# Nhân ma trận song song
def matrix_multiply_parallel(A, B):
    rows_A = len(A)
    cols_A = len(A[0])
    rows_B = len(B)
    cols_B = len(B[0])

    # Khởi tạo ma trận kết quả với giá trị 0
    result = [[0] * cols_B for _ in range(rows_A)]

    # Hàm phụ trách tính toán mỗi phần tử của ma trận kết quả
    def multiply_row(i):
        for j in range(cols_B):
            for k in range(cols_A):
                result[i][j] += A[i][k] * B[k][j]

    # Sử dụng ThreadPoolExecutor để thực hiện song song
    with concurrent.futures.ThreadPoolExecutor() as executor:
        executor.map(multiply_row, range(rows_A))

    return result

# Kiểm tra xem hai ma trận có giống nhau không
def check_equal(matrix1, matrix2):
    for i in range(len(matrix1)):
        for j in range(len(matrix1[0])):
            if matrix1[i][j] != matrix2[i][j]:
                return False
    return True

# Sinh ma trận ngẫu nhiên
def generate_random_matrix(rows, cols):
    return [[random.randint(1, 10) for _ in range(cols)] for _ in range(rows)]

```

```python
# Đo thời gian thực hiện
def measure_time(func, *args):
    start_time = time.time()
    result = func(*args)
    end_time = time.time()
    return result, end_time - start_time

# Main function
def main():
    # Sinh ma trận A và B có kích thước 400x400
    rows, cols = 400, 400
    A = generate_random_matrix(rows, cols)
    B = generate_random_matrix(cols, rows)  # Ma trận B cần có số cột bằng số dòng của A

    # Kiểm tra nhân ma trận tuần tự
    print("Running sequential matrix multiplication...")
    result_sequential, time_sequential = measure_time(matrix_multiply_sequential, A, B)
    print(f"Time taken for sequential multiplication: {time_sequential:.4f} seconds")

    # Kiểm tra nhân ma trận song song
    print("Running parallel matrix multiplication...")
    result_parallel, time_parallel = measure_time(matrix_multiply_parallel, A, B)
    print(f"Time taken for parallel multiplication: {time_parallel:.4f} seconds")

    # Kiểm tra kết quả
    if check_equal(result_sequential, result_parallel):
        print("Results are the same between sequential and parallel multiplication.")
    else:
        print("Results are different between sequential and parallel multiplication.")

# Chạy chương trình
if __name__ == "__main__":
    main()
```

Kết quả

```
Running sequential matrix multiplication...
Time taken for sequential multiplication: 15.0492 seconds
Running parallel matrix multiplication...
Time taken for parallel multiplication: 15.3341 seconds
Results are the same between sequential and parallel multiplication.
```