Спільні ресурси: a, s, MR.

**2.3. Аналіз задачі з точки зору концепції необмеженого паралелізму (КНП)**

Для оцінки необхідного часу обчислень використаємо теорему Мунро-Петерсена, яка для комп’ютерної системи з необмеженим числом процесорів формулюється наступним чином: якщо виконується обчислення скалярної величини, яке потребує *m* бінарних операцій, то необхідний час обчислень tp:

tp ≥ [log2(m + 1)].

Для обчислення а = max(Z) необхідно виконати N бінарних операцій порівняння. Тому час виконання буде:

tp1 ≥ [log2(N + 1)].

Для обчислення одного елементу добутку матриць a \* MR \* MK необхідно виконати N + 1 множень та N – 1 операцій додавання. Час виконання:

tp2 ≥ [log2(2N + 1)].

Для обчислення одного елементу матриці (B \* C) \* MO, необхідно виконати  N  + 1 операцій множення та N – 1 операцій додавання. Тому час виконання буде:

tp3 ≥ [log2(2N + 1)].

Так як другий та третій етап незалежні, то вони можуть виконуватись паралельно. Тому сумарний час їх виконання буде рівний максимальному з двох:

tp2,3 ≥ max(tp2, tp3) = tp3 = tp2 = [log2(2N + 1)].

Для обчислення одного елементу суми матриць (B \* C) \* MO + a \* MR \* MK необхідно виконати одну операцію додавання. Час виконання:

tp4 ≥ [log2(2)] = 1.

Сумарний час виконання всіх етапів обчислень буде виражатись наступною формулою:

tp ≥ tp1 + tp2,3 + tp4 = [log2(N + 1)] + [log2(2N + 1)] + 1.

**2.4. Розробка алгоритмів процесів**

Так як розроблюване програмне забезпечення має бути масштабованим, тобто має працювати на системі з будь-якою кількістю процесорів, то зручним варіантом реалізації є написання єдиного алгоритму для всіх задач.

|  |  |
| --- | --- |
| **Крок алгоритму** | **ТС, КД** |
| 1. Якщо *tid = 0*, ввести MA, B, MR |  |
| 2. Якщо *tid = 0*, сигнал задачам *1…Р-1* про завершення *вводу 1*. | S*i,1* i = |
| 3. Якщо *tid = Р-1*, вести C, MO, Z, MK. |  |
| 4. Якщо *tid = Р-1*, сигнал задачам *0…Р-2* про завершення *вводу 2*. | S*i,2* i = |
| 5. Якщо *tid != 0*, чекати сигналу про завершення *вводу 1* від задачі *0*. | W*0,1* |
| 6. Якщо *tid != Р-1*, чекати сигналу про завершення *вводу 2* від задачі *Р-1* | W*P-1,2* |
| 7. Обчислення a*i* = max(Z*H*), si = BH \* CH |  |
| 8. Обчислення a = max(a, a*i*), s = s + si. | КД |
| 9. Сигнал всім задачам про завершення обчислення. | S*i,3* i = |
| 10. Чекати сигналів про завершення обчислення від всіх задач. | W*i3* i = |
| 11. Копія МR*i* = MR, s*i* = s, a*i* = a. | КД |
| 12. Обчислення MAH = si \* MOH + ai \* (MRi \* MKH). |  |
| 13. Якщо *tid != 0*, сигнал задачі *0* про завершення обчислень. | S*0,4* |
| 14. Якщо *tid = 0*, чекати сигналів про завершення обчислення від задач *1…Р-1* | W*i4* i = |
| 15. Якщо *tid = 0*, вивести *МА*. |  |

**2.5. Розробка схеми взаємодії процесів**

Під час виконання даного етапу була розроблена структура класу-монітора TaskControl. Він використовується для синхронізації паралельних потоків. На даному етапі визначався набір захищених елементів, що будуть знаходитись у моніторі, а також множина захищених операцій. Набір захищених елементів визначається множиною спільних ресурсів (див. паралельний математичний алгоритм) та множиною змінних, що використовуються в якості умов. Семантика захищених операцій обиралася виходячи з завдання мiнiмiзацiї кількості захищених операцій.

Клас TaskControl містить поля *а, MR, s* для зберігання відповідних спільних ресурсів, а також поля inputCount, calculationCount, preparationCount для організації умов виконання методів монітору. Структура класу TaskController наведена в додатку А. В класі містяться наступні методи:

– waitInput — для очікування введення даних в потоках T(0) та T(P-1);

– waitCalculation — для очікування закінчення обчислень;

– waitPreparation — для очікування закінчення пошуку максимального елемета вектора та добутку векторів;

– inputMR — для введення MО;

– getMR — для копіювання спільного ресурсу MR;

– getA — для копіювання спільного ресурсу a;

– getS — для копіювання спільного ресурсу s;

– setA — для виконання операцій над спільним ресурсом a;

– setS — для виконання операцій над спільним ресурсом s;

– signalInputDone — для сигналу про завершення вводу даних;

– signalCalculationDone — для сигналу про завершення обчислень;

– signalPreparationDone — для сигналу про завершення пошуку максимального елемента вектора та добутку векторів;

**2.6. Розробка програми ПРГ1**

Програма для системи з спільною пам’яттю написана на мові Java. Основні класи програми:

– Director — основний клас. Містить головний метод, що запускається JVM при старті програми. Головний метод формує ідентифікатори потоків, запускає потоки та вимірює час їх виконання. В основному класі знаходиться константа P, змінюючи яку можна виконати налаштування програми під конкретну комп’ютерну систему;

– TaskWorker — задачний тип, реалізує інтерфейс Runnable;

– TaskController — клас-монітор, який вирішує задачі синхронізації та взаємного виключення, а також зберігає спільні ресурси;

Повний лістинг програми наведено у додатку Д.

**2.7. Тестування програми ПРГ1**

Для тестування використовувалась паралельна обчислювальна система з наступним апаратним забезпеченням:

- процесор: Intel(R) Core(TM) i5-3337U CPU @ 1.80GHz

- оперативна пам'ять: 6 Гб DDR3;

В якості програмного забезпечення використовувались:

- операційна система: Windows 10;

- компілятор та віртуальна машина Java: Sun Java 1.8.0\_92, 64-бітна версія.

Таблиця 2.1. Час виконання програми з спільною пам’яттю(значення в мілісекундах)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | T1 | T2 | T3 | T4 |
| 1000 | 10850 | 6584 | 5036 | 4503 |
| 1500 | 50941 | 30895 | 24761 | 20804 |
| 2000 | 146102 | 85492 | 72218 | 61645 |
| 2500 | 313007 | 193823 | 155367 | 135717 |

Підрахунок коефіцієнту прискорення (КП) виконується за формулою

КП = Т1 / ТР

Таблиця 2.2. Значення КП для програми з спільною пам’яттю

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | Р | | | |
| 1 | 2 | 3 | 4 |
| 1000 | 1 | 1,64793439 | 2,1545 | 2,4095 |
| 1500 | 1 | 1,64884285 | 2,0573 | 2,4486 |
| 2000 | 1 | 1,70895522 | 2,0231 | 2,3701 |
| 2500 | 1 | 1,61491154 | 2,0146 | 2,3063 |

Підрахунок коефіцієнту ефективності (КЕ) відбувається за формулою

КЕ = КП / Р

Таблиця 2.3. Значення КЕ для програми зі спільною пам’яттю

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | Р | | | |
| 1 | 2 | 3 | 4 |
| 1000 | 1 | 0,82397 | 0,7182 | 0,6024 |
| 1500 | 1 | 0,82442 | 0,6858 | 0,6122 |
| 2000 | 1 | 0,85448 | 0,6744 | 0,5925 |
| 2500 | 1 | 0,80746 | 0,6715 | 0,5766 |

Рисунок 2.1. Графік залежності КП від кількості процесорів при N = 1000

Рисунок 2.2. Графік залежності КП від кількості процесорів при N = 1500

Рисунок 2.3. Графік залежності КП від кількості процесорів при N = 2000

Рисунок 2.4. Графік залежності КП від кількості процесорів при N = 2500

Рисунок 2.5. Графік залежності КЕ від кількості процесорів при N = 1000

Рисунок 2.6. Графік залежності КЕ від кількості процесорів при N = 1500

Рисунок 2.7. Графік залежності КЕ від кількості процесорів при N = 2000

Рисунок 2.8. Графік залежності КЕ від кількості процесорів при N = 2500

**2.8. Висновки до розділу 2**

– У даному розділі досліджені результати тестування паралельної програми для системи з загальною пам’яттю, написаної на мові Java. Тестування проводилось для 2, 3 та 4 потоків. Для розрахунку коефіцієнтів прискорення та ефективності була протестована окремо створена послідовна програма.

– Коефіцієнт прискорення приймає значення у проміжку від 1.61 до 2.44. Найвищі значення цього коефіцієнту приходяться на систему з чотирма потоками, найменше прискорення отримане для системи з двома потоками. Характер графіків коефіцієнтів прискорення однаковий для систем з 2, 3 та 4 ядрами.

– При збільшенні кількості ядер P > 2 коефіцієнт прискорення зменшується. Це в основному пов’язано з тим, що пропускна здатність пам’яті ділиться між всіма ядрами та під час запитів до пам’яті ядра простоюють. Крім того, деякий час витрачається на синхронізацію.

– Результати дослідження мають похибку, зумовлену тим, що процесори виділяються операційною системою не на монопольне використання, тобто, процесорний час може бути в будь-який час передано сторонній програмі. Чим менший час виконання програми, тим більша вірогідність виникнення досить значущої похибки.