Національний технічний університет України  
«Київський політехнічний інститут»

Кафедра обчислювальної техніки

(повна назва кафедри, циклової комісії)

Члени комісії

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

Студента 3 курсу групи ІО-01

напряму підготовки 050102

«Комп’ютерна інженерія»

Кравченко І.В.

(прізвище та ініціали)

Керівник доцент Корочкін О.В.

Національна оцінка

Кількість балів:

Оцінка: ECTS

**Курсова робота**

з дисципліни «Паралельні та розподілені обчислення»

(назва дисципліни)

на тему: «Розробка програмного забезпечення   
для паралельних комп’ютерних систем»

Київ – 2013 рік

Національний технічний університет України  
«Київський політехнічний інститут»

Факультет (інститут) інформатики та обчислювальної техніки

(повна назва)

Кафедра обчислювальної техніки

(повна назва)

Освітньо-кваліфікаційний рівень бакалавр

Напрям підготовки 6.050102 «Комп’ютерна інженерія»

(шифр і назва)

**Завдання**

на курсову роботу студенту

Кравченко Іван Володимирович

(прізвище, ім’я, по батькові)

1. Тема роботи «Розробка програмного забезпечення для паралельних комп’ютерних систем».
2. Керівник роботи к. т. н., доцент Корочкін Олександр Володимирович.

(вчене звання, науковий ступінь, прізвище та ініціали)

1. Строк подання студентом роботи 14 травня 2013 р.
2. Вихідні дані для роботи:

* Огляд засобів роботи процесами в мові Python
* математична задача A=sort(a\*B+C\*(MX\*MD))
* структури паралельної комп’ютерної системи з спільною пам’яттю (ПКС СП) та паралельної комп’ютерної системи з локальною пам’яттю (ПКС ЛП);
* Мови програмування та бібліотеки:С#, MPI

1. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):

* Огляд засобів роботи процесами в мові Python
* розробка і тестування програми ПРГ1 для ПКС СП;
* розробка і тестування програми ПРГ2 для ПКС ЛП.

СП

1

2

3

4

А,В,С а,МХ МD

Рисунок 1 – Структура ПКС СП

5

1

3

2

4

Рисунок 2 – Структура ПКС ЛП

1. Перелік графічного матеріалу:

* структурна схема ПКС СП;
* структурна схема ПКС ЛП;
* схеми алгоритмів процесів і головної програми для ПРГ1;
* схеми алгоритмів процесів і головної програми для ПРГ2.

1. Дата видачі завдання

КАЛЕНДАРНИЙ ПЛАН

|  |  |  |
| --- | --- | --- |
| № з/п | Назва етапів виконання КР | Строк виконання етапів КР |
|  | Виконання огляду для розділу 1 | 25.03.2013 |
|  | Розробка паралельного алгоритму рішення задачі | 06.04.2013 |
|  | Розробка алгоритмів процесів | 11.05.2013 |
|  | Розробка схем взаємодії процесів | 11.05.2013 |
|  | Розробка програм | 12.05.2013 |
|  | Тестування програм | 13.05.2013 |
|  | Оформлення КР | 08.05.2013 |
|  | Захист КР | 23.05.2013 |

Студент

(підпис) (прізвище та ініціали)

Керівник роботи

(підпис) (прізвище та ініціали)

**ПОЯСНЮВАЛЬНА  
ЗАПИСКА**

ЗМІСТ

ВСТУП……………………………………………………………………………….8

РОЗДІЛ 1. Огляд засобів роботи процесами в мові Python ………………….... 9

* 1. Вступ………………………………………………………………………...9
  2. Опис процесів і потоків в Python………………………………………….9
  3. Потоки управління………………………………………………………….12
     1. Функції модуля threading ……………………………………………13
     2. Клас Thread……………………………………………………………14
     3. Замки………………………………………………………………….15
     4. Тупікова ситуація……………………………………………………16
     5. Семафори……………………………………………………………..18
     6. Події……………………………………………………………………19
     7. Умови………………………………………………………………….19
  4. Висновки до розділу 1……………………………………………………21

РОЗДІЛ 2. РОЗРОБКА ПРОГРАМИ ПРГ1 ДЛЯ ПКС ОП……………………22

2.1 Розробка паралельного математичного алгоритму…………………22

2.2 Розробка алгоритмів процесів………………………………………….23

2.3 Розробка схеми взаємодії процесів…………………………………….24

2.4 Розробка програми ПРГ1………………………………………………. 27

2.5 Тестування програми ПРГ1 …………………………………………. 27

2.6 Висновки до розділу 2 …………………………………………………29

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМИ ПРГ2 ДЛЯ ПКС ЛП ………………….31

3.1 Розробка паралельного математичного алгоритму…………………31

3.2 Розробка алгоритмів процесів……………………………………….32

3.3 Розробка схеми взаємодії процесів………………………………….33

3.4 Розробка програми ПРГ2……………………………………………33

3.5 Тестування програми ПРГ2…………………………………………35

3.6 Висновки до розділу 3……………………………………………….37

ОСНОВНІ РЕЗУЛЬТАТИ І ВИСНОВКИ ДО РОБОТИ……………………….38

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ……………………………………… 40

ДОДАТКИ……………………………………………………............................. 41

# ВСТУП

Курсова робота по дисципліні «Паралельні і розподілені обчислення» складається з трьох розділів.

В першому розділі описані способи розпаралелення програми на мові Phyton.

Другий та третій розділи присвячені розробці програми для обчислення математичної задачі в паралельній комп’ютерній системі зі спільною та локальною пам’яттю відповідно. Програмне забезпечення для комп’ютерної системи зі спільною пам’яттю розроблено на мові C# з використанням семафорів, мютексів, замків та моніторів, а для комп’ютерної системи з локальною пам’яттю – розроблено на мові С++ з використанням бібліотеки MPI. Проведено тестування отриманих програмних продуктів і зроблено висновки по їх ефективності.

Лістинги та алгоритми розроблених програм наведено у додатках.

**РОЗДІЛ 1. Огляд засобів роботи процесами в мові Python**

**1.Вступ**

Python - високорівнева мова програмування загального призначення з акцентом на продуктивність розробника і розуміння коду. Синтаксис Пітона мінімальний. У той же час стандартна бібліотека включає великий обсяг корисних функцій. Розробка мови Python була розпочата наприкінці 1980-х років співробітником голландського інституту CWI Гвідо ван Россум. Для розподіленої ОС Amoeba вимагався розширюваний скриптова мова, і Гвідо почав писати Python на дозвіллі, запозичивши деякі напрацювання для мови ABC (Гвідо брав участь у розробці цієї мови, орієнтованого на навчання програмуванню). У лютому 1991 року Гвідо опублікував вихідний текст в ньюсгруппе alt.sources. З самого початку Python проектувався як об'єктно-орієнтована мова.

**1.2.Процеси і потоки в Python**

Запущена на Python програма називається процесом. Процес при запуску містить один єдиний потік, який зазвичай називають головним потоком процесу. Потік послідовно виконує інструкції програмного коду.

Процес

Об’єм займає мої пам’яті

Список відкритих файлів

Головний потік

Програмний лічильник,який посилається на виконувану інструкцію

Стек викликів , який використовується для збереження локальних змінних функцій

Рис.1.1 Процес з 1 потоком

 Процес містить інформацію про обсяг займаної оперативної пам'яті, списку відкритих файлів, програмний лічильник посилається на виконувану інструкцію, стек викликів використовуваний для зберігання локальних змінних функції.

Процес

Не демон

Дочірній процес

Не демон

Дочірній процес

Демон

Головний потік

Головний потік

Головний потік

Рис.1.2 Типи процесів

Програма може створювати нові процеси, звані - дочірніми. Дочірні процеси є незалежними і ізольованими один від одного. У процесу існує прапор, що говорить про те, чи є процес демонічним. Демонічний процес автоматично завершується разом з процесом створив його, крім того демонічні процеси не можуть створювати дочірніх. Для обміну інформацією між процесами потрібно використовувати механізми взаємодії процесів IPC (Inter-Process Communication). Одним з таких механізмів є - обмін повідомленнями. Повідомлення - це простий буфер двійкових байтів передаються через канали введення-виведення, по засобом операцій send () і recv (). Також існує й інші механізми, наприклад відображувані області пам'яті (модуль mmap). Цей метод заснований на використанні поділюваних областей пам'яті, доступних усім процесам.

Процес

Виконуючі інструкції

Потік 2

Потік 1

Стек викликів

Виконання інструкції

Стек визовів

Рис.1.3 Процес з 2 потоками

Замість запуску нових процесів, можна створити нові потоки в поточному процесі. Такі потоки будуть виконувати власні послідовності інструкцій, мати свій стек виклику функцій, а також мати доступ до даних і системних ресурсів виділеним процесу.

Створювати нові потоки доцільно, коли необхідно реалізувати одночасне виконання декількох завдань і є значний обсяг даних, які повинні бути доступні всім потокам. При паралельному виконанні завдань, виникає проблема доступу до спільно використовуваних даних. Спроба змінити деякі дані одночасно з декількох потоків може призвести до їх пошкодження і порушення цілісності стану програми. Для вирішення цієї проблеми необхідно виділити критичний ділянку програмного коду, в якому здійснюється доступ до даних і забезпечити його виконання під захистом взаємовиключних блокувань. В Python існує глобальна блокування інтерпретатора Global Interpreter Lock (GIL). Через доступу до змінних інтерпретатора, GIL змушена блокувати потоки і фактично одночасно буде виконуватися тільки один потік. Лише при блокуючих операціях, таких як введення-виведення у інших потоків є шанс перехопити естафету на виконання. Багатопоточність краще всього використовувати при написанні логіки з безліччю операції вводу-виводу. Програмний код з численними обчисленнями не має сенс переносити на багатопоточність, це плачевно позначиться на продуктивності програми. Великий обсяг обчислень в програмному коді найкраще реалізувати в кілька процесів, забезпечивши їх взаємодія за допомогою механізму обміну повідомлень.

**1.3.Потоки управління**

У сучасній операційній системі, навіть не виконуючи нічого особливого, можуть одночасно працювати кілька процесів (processes). Наприклад, при запуску програми запускається новий процес. Функції для управління процесами можна знайти в стандартному модулі os мови Python.

Потоки управління (threads) утворюються і працюють в рамках одного процесу. У однопоточном додатку (програмі, яка не використовує додаткових потоків) є тільки один потік управління. Говорячи спрощено, при запуску програми цей потік послідовно виконує надибуємо у програмі оператори, прямуючи по одній з альтернативних гілок оператора вибору, проходить через тіло циклу потрібне число раз, вибирається до місця обробки винятки при порушенні виключення. У будь-який момент часу інтерпретатор Python знає, яку команду виконати наступною. Після виконання команди стає відомо, якій команді передати управління. Ця ниточка неперервна в ході виконання програми і обривається тільки по її завершенні.

Тепер можна уявити собі, що в деякій точці програми ниточка роздвоюється, і кожен потік йде своїм шляхом. Кожен з утворених потоків може надалі ще кілька разів роздвоюватися. (При цьому один з потоків завжди залишається головним, і його завершення означає завершення всієї програми.) У кожен момент часу інтерпретатор знає, яку команду який потік повинен виконати, і приділяє кванти часу кожному потоку. Таке, здавалося б, незначне ускладнення механізму виконання програми насправді потребує якісних змін в програмі - адже діяльність потоків повинна бути узгоджена. Не можна допускати, щоб потоки одночасно змінювали один і той же об'єкт, результат такої зміни, швидше за все, порушить цілісність об'єкта.

Одним із класичних засобів узгодження потоків є об'єкти, звані семафорами. Семафори не допускають виконання деякої ділянки коду декількома потоками одночасно. Найпростіший семафор - замок (lock) або mutex (від англійського mutually exclusive, взаємовиключний). Для того щоб потік міг продовжити виконання коду, він повинен спочатку захопити замок. Після захоплення замку потік виконує певну ділянку коду і потім звільняє замок, щоб інший потік міг його отримати і пройти далі до виконання охоронюваного замком ділянці програми. Потік, зіткнувшись із зайнятим іншим потоком замком, звичайно чекає його звільнення.

Підтримка багатопоточності в мові Python доступна через використання ряду модулів. У стандартному модулі threading визначені потрібні для розробки многопоточної (multithreading) програми класи: кілька видів семафорів (класи замків Lock, RLock і клас Semaphore) та інші механізми взаємодії між потоками (класи Event і Condition), клас Timer для запуску функції по закінченні деякого часу . Модуль Queue реалізує чергу, якою можуть користуватися відразу кілька потоків. Для створення і управління (низькорівневого) потоками в стандартному модулі thread визначено клас Thread.

**1.3.1. Функції модуля threading**

У модулі threading, який тут використовується, є функції, що дозволяють отримати інформацію про потоки:

• activeCount () Повертає кількість активних в даний момент екземплярів класу Thread. Фактично, це len (threading.enumerate ()).

• currentThread () Повертає поточний об'єкт-потік, тобто відповідний потоку управління, який викликав цю функцію. Якщо потік не був створений через модуль threading, буде повернуто об'єкт-потік з скороченою функціональністю (dummy thread object).

• enumerate () Повертає список активних потоків. Що завершилися і ще не розпочаті потоки не входять в список.

**1.3.2. Клас Thread**

Екземпляри класу threading.Thread представляють потоки Python-програми. Задати дії, які будуть виконуватися в потоці, можна двома способами: передати конструктору класу виконуваний об'єкт і аргументи до нього або шляхом успадкування отримати новий клас з перевизначення методів run(). Конструктор класу threading.Thread має такі аргументи:

Thread (group, target, name, args, kwargs)

Тут group - група потоків (поки що не використовується, повинен бути рівний None), target - об'єкт, який буде викликаний в методі run (), name - ім'я потоку, args і kwargs - послідовність і словник позиційних і іменованих параметрів (відповідно) для виклику заданого в параметрі target об'єкта

Життям потоків можна управляти викликом методів:

• start () Дає потоку життя.

• run () Цей метод являє дії, які повинні бути виконані в потоці.

• join ([timeout]) Потік, який викликає цей метод, призупиняється, очікуючи завершення потоку, чий метод викликаний. Параметр timeout (число з плаваючою точкою) дозволяє вказати час очікування (у секундах), після закінчення якого був загальмований потік продовжує свою роботу незалежно від завершення потоку, чий метод join був викликаний. Викликати join () деякого потоку можна багато разів. Потік не може викликати метод join () самого себе. Також не можна очікувати завершення ще не запущеного потоку. Слово "join" в перекладі з англійської означає "приєднати", тобто, метод, який викликав join (), бажає, щоб потік по завершенні приєднався до викликає метод потоку.

• getName () Повертає ім'я потоку. Для головного потоку це "MainThread".

• setName (name) Привласнює потоку ім'я name.

• isAlive () Повертає істину, якщо потік працює (метод run () вже викликаний, але ще не завершився).

• isDaemon () Повертає істину, якщо потік має ознаку демона. Програма на Python завершується по завершенні всіх потоків, які не є демонами. Головний потік демоном не є.

• setDaemon (daemonic) Встановлює ознака daemonic того, що потік є демоном. Початкове значення цієї ознаки запозичується у потоку, що запустив даний. Ознака можна змінювати тільки для потоків, які ще не запущені.

У модулі Thread поки що не реалізовані можливості, властиві потокам в Java (визначення груп потоків, припинення та переривання потоків ззовні, пріоритети та деякі інші речі), проте вони, швидше за все, будуть створені в недалекому майбутньому.

**1.3.3. Замки**

Найпростіший замок може бути реалізований на основі класу Lock модуля threading. Замок має два стани: він може бути або відкритий, або замкнений. В останньому випадку їм володіє деякий потік. Об'єкт класу Lock має наступні методи:

• acquire ([blocking = True]) Робить запит на замикання замка. Якщо параметр blocking не зазначено або є істиною, то потік буде чекати звільнення замку. Якщо параметр не був заданий, метод не поверне значення. Якщо blocking було задано і правдивий, метод поверне True (після успішного оволодіння замком). Якщо блокування не потрібний (тобто заданий blocking = False), метод поверне True, якщо замок не був замкнений і їм успішно опанував даний потік. В іншому випадку буде повернуто False.

• release () Запит на відмикання замка.

• locked () Повертає поточний стан замку (True - замкнений, False - відкритий). Слід мати на увазі, що навіть якщо стан замку тільки що перевірено, це не означає, що він збереже цей стан до наступної команди.

Є ще один варіант замку - threading.RLock, який відрізняється від threading.Lock тим, що деякий потік може запитувати його запирання багато разів. Відмикання такого замка повинне відбуватися стільки ж разів, скільки було замикання. Це може бути корисно, наприклад, всередині рекурсивних функцій.

Замки дозволяють обмежувати вхід в деяку область програми одним потоком. Замки можуть бути потрібні для забезпечення цілісності структури даних. Наприклад, якщо для коректної роботи програми потрібне додавання певного елемента відразу в кілька списків або словників, такі операції в багатопотоковому додатку слід обставити замками. Навколо атомарних операцій над вбудованими типами (операцій, які не викликають виконання якогось іншого коду на Python) замки ставити необов'язково. Наприклад, метод append () (вбудованого) списку є атомарної операцією, а той же метод, реалізований користувальницьким класом, може вимагати блокувань. Відсутність замку в критичній частині програми, що працює над загальними для двох і більше потоків ресурсами, може призвести до випадкових, важковизначимих помилок.

**1.3.4 Тупікова ситуація (deadlock)**

Замки застосовуються для управління доступом до ресурсу, який можна використовувати спільно. У програмі таких ресурсів може бути кілька. При роботі з замками важливо добре продумати,чи не зайде виконання програми в тупік (deadlock) через те, що двом потокам потрібні одні і ті ж ресурси, але ні той, ні інший не зможуть їх отримати, так як вони вже отримали замки . Така ситуація проілюстрована в наступному прикладі:

import threading, time

resource = {'A': threading.Lock (), 'B': threading.Lock ()}

def proc (n, rs):

for r in rs:

print "Процес% s запитує ресурс% s"% (n, r)

resource [r]. acquire ()

print "Процес% s отримав ресурс% s"% (n, r)

time.sleep (1)

print "Процес% s виконується"% n

for r in rs:

resource [r]. release ()

print "Процес% s закінчив виконання"% n

p1 = threading.Thread (target = proc, name = "t1", args = ["1", "AB"])

p2 = threading.Thread (target = proc, name = "t2", args = ["2", "BA"])

p1.start ()

p2.start ()

p1.join ()

p2.join ()

У цьому прикладі два потоки (t1 і t2) запрошують замки до одних і тих же ресурсів (A і B), але в різному порядку, від чого виходить, що ні в того, ні в іншого не вистачає ресурсів для подальшої роботи, і вони обидва безнадійно повисають, очікуючи звільнення потрібного ресурсу. Завдяки операторам print можна побачити послідовність подій:

Процес 1 запрошувати ресурс A

Процес 1 отримав ресурс A

Процес 2 запрошувати ресурс B

Процес 2 отримав ресурс B

Процес 1 запрошувати ресурс B

Процес 2 запрошувати ресурс A

Існують методики, що дозволяють уникнути подібних тупиків.Можна порадити наступні прийоми:

• побудувати логіку додатка так, щоб ніколи не запитувати замки до двох ресурсам відразу. Можливо, доведеться визначити складовою ресурс. Зокрема, до даного прикладу можна було б визначити замок "AB" для вказівки ексклюзивного доступу до ресурсів A і B.

• суворо упорядкувати всі ресурси (наприклад, по ціні) і завжди запитувати їх у певному порядку (скажімо, починаючи з більш дорогих ресурсів). При цьому перед замовленням деякого ресурсу потік повинен відмовитися від заблокованих їм більш дешевих ресурсів.

**1.3.5. Семафори**

Семафори (їх іноді називають семафорами Дійкстра (Dijkstra) на ім'я їх винахідника) є більш загальним механізмом синхронізації потоків, ніж замки. Семафори можуть допустити в критичну область програми відразу декілька потоків. Семафор має лічильник запитів, який зменшується з кожним викликом методу

acquire () і збільшується при кожному виклику release (). Лічильник не може стати менше нуля, тому в такому стані потокам доводиться чекати, як і у випадку з замками, поки значення лічильника не збільшиться.

Конструктор класу threading.Semaphore приймає в якості (необов'язкового) аргументу початковий стан лічильника (за замовчуванням воно дорівнює 1, що відповідає замку класу Lock). Методи acquire () і release () діють аналогічно описаним вище однойменним методам у замків.

Семафор може застосовуватися для охорони обмеженого ресурсу. Наприклад, з його допомогою можна вести пул з'єднань з базою даних. Приклад такого використання семафора (запозичений з документації до Python) даний нижче:

from threading import BoundedSemaphore

maxconnections = 5

# Підготовка семафора

pool\_sema = BoundedSemaphore (value = maxconnections)

# Усередині потоку:

pool\_sema.acquire ()

conn = connectdb ()

# ... використання з'єднання ...

conn.close ()

pool\_sema.release ()

Таким чином, застосовується не більше п'яти з'єднань з базою даних. У прикладі використаний клас threading.BoundedSemaphore. Примірники цього класу відрізняються від екземплярів класу threading.Semaphore тим, що не дають зробити release () більше, ніж зроблений acquire ().

**1.3.6.Події**

Ще одним способом комунікації між об'єктами є події. Екземпляри класу threading.Event можуть бути використані для передачі інформації про настання деякої події від одного потоку одному або декільком іншим потокам. Об'єкти-події мають внутрішній прапор, який може знаходитися в установленому чи скинутому стані. При своєму створенні прапор події знаходиться в скинутому стані. Якщо прапор у встановленому стані, очікування не відбувається: потік, що викликав метод wait () для очікування події, просто продовжує свою роботу. Нижче наведені методи екземплярів класу threading.Event:

• set () Встановлює внутрішній прапор, що сигналізує про настання події. Всі чекають даної події потоки виходять зі стану очікування.

• clear () Скидає прапор. Всі події, які викликають метод wait () цього об'єкта-події, будуть перебувати в стані очікування до тих пір, поки прапор скинутий, або після закінчення заданого таймауту.

• isSet () Повертає стан прапора.

• wait ([timeout]) Перекладає потік в стан очікування, якщо прапор скинутий, і відразу повертається, якщо прапорець встановлений. Аргумент timeout задає таймаут в секундах, після закінчення якого очікування припиняється, навіть якщо подія не наступила.

**1.3.7. Умови**

Більш складним механізмом комунікації між потоками є механізм умов. Умови представляються у вигляді екземплярів класу threading.Condition і, подібно тільки що розглянутим подіям, сповіщають потоки про зміну деякого стану. Конструктор класу threading.Condition приймає необов'язковий параметр, задаючий замок класу threading.Lock або threading.RLock. За замовчуванням створюється новий екземпляр замку класу threading.RLock. Методи об'єкта-умови описані нижче:

• acquire (...) Запитує замок. Фактично викликається однойменний метод належить об'єкту-умові об'єкта-замку.

• release () Знімає замок.

• wait ([timeout]) Перекладає потік в режим очікування. Цей метод може бути викликаний тільки в тому випадку, якщо викликає його потік отримав замок. Метод знімає замок і блокує потік до появи оголошень, тобто викликів методів notify () і notifyAll () іншими потоками. Необов'язковий аргумент timeout задає таймаут очікування в секундах. При виході з очікування потік знову запитує замок і повертається з методу wait ().

• notify () Виводить з режиму очікування один з потоків, які очікують дані умови. Метод можна викликати, лише оволодівши замком, асоційованим з умовою. Документація попереджає, що в майбутніх реалізаціях модуля з цілей оптимізації цей метод буде переривати очікування відразу декількох потоків. Сам по собі метод notify () не призводить до продовження виконання чекали умови потоків, тому що цьому перешкоджає зайнятий замок. Потоки одержують управління тільки після зняття замка потоком, що викликав метод notify ().

• notifyAll () Цей метод аналогічний методу notify (), але перериває очікування всіх чекають виконання умови потоків.

Модуль thread

У порівнянні з модулем threading, модуль thread надає низькорівневий доступ до потоків. Багато функцій модуля threading, який розглядався до цього, реалізовані на базі модуля thread. Тут варто зробити деякі зауваження по застосуванню потоків взагалі. Документація по Python попереджає, що використання потоків має особливості:

• Виключення KeyboardInterrupt (переривання від клавіатури) може бути отримано будь-яким з потоків, якщо в поставці Python немає модуля signal (для обробки сигналів).

• Не всі вбудовані функції, блоковані очікуванням введення, дозволяють іншим потокам працювати. Щоправда, основні функції начебто time.sleep (), select.select (), метод read () файлових об'єктів не блокують інші потоки.

• Неможливо перервати метод acquire (), так як виключення KeyboardInterrupt збуджується тільки після повернення з цього методу.

• Небажано, щоб головний потік завершувався раніше за інших потоків, оскільки не будуть виконані необхідні деструктори і навіть частини finally в операторах try-finally. Це пов'язано з тим, що майже всі операційні системи завершують додаток, у якого завершився головний потік.

* 1. **Висновки до розділу 1**

Навички паралельного програмування необхідні будь професійному програмістові. Одним з варіантів організації (псевдо) паралельного програмування є багатопотокове програмування. У звичайній (однопоточному) програмі діє всього один потік управління, а в багатопоточній одночасно можуть працювати декілька потоків.

Паралельне програмування вимагає ретельного відпрацювання взаємодії між потоками управління. Деякі ділянки коду необхідно огороджувати від одночасного використання двома різними потоками, щоб не порушити цілісність змінюваних структур даних або логіку роботи із зовнішніми ресурсами. Для огорожі ділянок коду використовуються замки і семафори.

Стандартна бібліотека Python надає досить непоганий набір можливостей для багатопотокового програмування в модулях threading і thread, а також деякі корисні допоміжні модулі .

**Розділ 2.**

* 1. **Розробка паралельного математичного алгоритму**

Відповідно до умови рис.2.1 та заданої формули A=sort(a\*B+C\*(MX\*MD)) виконується розпаралелювання :

1)AH=a\*BH+C\*(MX\*MDH)

2)AH = Sort(АH)

3)A1,2 = mergeSort(AH,AH)

A3,4 = mergeSort(AH,AH)

4)A = mergeSort(A1,2, A3,4),

де H = N / P,

N – розмірність матриць,

P – кількість вузлів ПКС.

Структура зі спільною пам’яттю має спільні ресурси: a,C,MX

При розробці алгоритму N приймалось кратним Р.

Щоб розрахувати мінімальний час виконання програми для Р процесорів використаємо теорему Мунро-Петерсона: Tp=] log2(m+1) [

де m – кількість бінарних операцій, Tр – час розв’язання задачі на Р вузлах.

2N+1 –для множення , 2N-1-для додавання

Підставимо отримане значення в формулу:

Tp1=] log2(m+1) [ = ] log2(2N+1+2N-1+1) [ = ] log(4N+1)[

Для сортування - N порівнянь

Tp2=]log2(N)[

Tp=Tp1+Tp2 = ] log2(4N+1) [ = ] log2(4N+1)[+]log2(N)[

СП

1

2

3

4

А,В,С а,МХ МD

Рис. 2.1Структура ПКС

* 1. Розробка алгоритмів процесів

Реалізація алгоритмів представлена на табл. 2.1 та 2.2.

Табл. 2.1 Алгоритм 1 і 2 процесів

|  |  |  |  |
| --- | --- | --- | --- |
| **T1** | **ТС КД** | **T2** | **ТС КД** |
| 1. Чекати введення від T2, T3,T4 | **W2-1**  **W3-1**  **W4-1** | 1. Введення B,C |  |
| 2. Копія С1=С  а1=a, MХ1=MХ | **КД** | 2. Сигнал T1, T3, Т4 про закінчення введення | **S1,3,4-1** |
| 3. Рахування  AH=a1\*BH+ C1\* (MX1\*MDH) |  | 3. Чекати введення від T4, T3 | **W4-1**  **W3-1** |
| 4.Сортування AH = Sort(АH) |  | 4. Копія  а2=a, MХ2=MХ, C2=C | **КД** |
| 5.Чекати сигнал про завершення  Сортування від Т2 | **W2-2** | 5. Рахування  AH=a2\*BH+ C2\*(MX2\*MDH) |  |
| 6.Об’єднання A1,2= mergeSort(AH,AH) |  | 6.Сортування AH = Sort(АH) |  |
| 7.Чекати сигнал про завершення об’єднання від Т3 | **W3-2** | 7. Сигнал про завершення сортування Т1 | **S1-2** |
| 8.A = mergeSort(A1,2, A3,4) |  | 8.Чекати сигнал про завершення об’єнання від Т1 | **W1-2** |
| 9. сигнал про завершення об’єднання Т2 | **S2-1** | 9.Виведення А |  |

Табл. 2.2 Алгоритм 3 і 4 процесів

|  |  |  |  |
| --- | --- | --- | --- |
| **T3** | **ТС КД** | **T4** | **TC**  **КД** |
| 1. Введення a,MX |  | 1.Введення MD |  |
| 2. Сигнал T1, T2, Т4 про закінчення введення | **S1,2,4-1** | 2. Cигнал Т1,Т2,Т3 про закінчення введення | **S1,2,4-1** |
| 3. Чекати введення від T4, T2 | **W4-1**  **W2-1** | 3. Чекати введення від Т3,Т2 | **W2-1**  **W3-1** |
| 4. Копія  а3=a, MХ3=MХ, C3=C | **КД** | 4.Копія  а4=a, MХ4=MХ, C4=C | КД |
| 5. Рахування  AH=a3\*BH+ C3\*(MX3\*MDH) |  | 5.Рахування  AH=a4\*BH+ C4\*(MX4\*MDH) |  |
| 6. Сортування AH = Sort(АH) |  | 6.Сортування AH = Sort(АH) |  |
| 7.Чекати сигнал про завершення  Сортування від Т4 | **W4-2** | 7. Сигнал про завершення сортування Т3 | **S3-2** |
| 8.Об’єднання A1,2= mergeSort(AH,AH) |  |  |  |
| 9. сигнал про завершення об’єднання Т1 | **S1-2** |  |  |

**2.3 Розробка схеми взаємодії процесів**

Розробка схеми взаємодії процесів представлена на рис. 2.2

Дана структура складається з 4 потоків, які представлені у вигляді пронумерованих відповідно паралелограмів.

Циліндри-введення даних.

Круг –критична секція.

Стрілками вказана взаємодія між потоками. Кожна стрілка має своє ім’я, яке вказує на спосіб реалізації.

SX-У-відправлений сигнал іншому потоку,де Х-кому відправляються сигнал, У-порядковий номер

WX-У-прийом сигналу від іншого потоку, де Х-від кого приймається сигнал, У-порядковий номер.

**2.4 Розробка програми ПРГ1**

Програма складається з 1 класу-Program.

Основні змінні:

N- розмір матриць та векторів

P- кількість потоків

sem1-sem6-семафори для взаємодії між потоками

event1-event2- сигнали для взаємодії між потоками

mutex\_m-мютекс для контролю доступу до критичної ділянки

copy\_lock-замок для контролю доступу до критичної ділянки

monitor\_x-монітор для контролю доступу до критичної ділянки

В прогамі є 4 основні методи Т1, Т2, Т3, Т4, які реалізують алгоритми потоків.

Лістинг програми знаходиться в додатку Ж. **2.5. Тестування програми**

Метою проведення тестування є оцінка коефіцієнтів прискорення і

коефіцієнтів ефективності для розроблених програм при їх виконанні на

реальній паралельній обчислювальній системі. Для визначення вищевказаних

коефіцієнтів проведений ряд експериментів із різними розмінностями

операндів і різною кількістю запущених задач. Розмірність операндів була

вибрана в межах від 800 до 1200.

Таблиця 2.3. – Час виконання програми обчислення функції на мові С#

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | T1 | T2 | T3 | T4 |
| 800 | 4.976 | 2.573 | 1.939 | 1.482 |
| 1000 | 9.9846 | 5.117 | 3.8732 | 2.838 |
| 1200 | 17.436 | 8.8955 | 6.706 | 4.85627 |

На основі даних з табл.2.3 виконується розрахунок значень коефіцієнтів прискорення(ky=T1/T2, Ку= T1/T3, Ку = T1/T4), які відображаються в табл.2.4

Таблиця 2.4. – Значення коефіцієнтів прискорення

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | Кількість процесів | | | |
| 1 | 2 | 3 | 4 |
| 800 | 1 | 1,933 | 2,566 | 3,3576 |
| 1000 | 1 | 1,951 | 2,578 | 3,5179 |
| 1200 | 1 | 1,96 | 2,6 | 3,59 |

На основі даних в табл. 2.4 виконується розрахунок значень коефіцієнтів ефективності((Кэ= Ку/2 \* 100%, Ку= Ку/3\* 100%, , Ку = Ку/4\* 100%), які відображаються в табл. 2.5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | Кількість процесів | | | |
| 1 | 2 | 3 | 4 |
| 800 | 100 | 96,65 | 85,53 | 83,94 |
| 1000 | 100 | 97,55 | 85,93 | 87,95 |
| 1200 | 100 | 98 | 86,67 | 89,75 |

Таблиця 2.5. – Значення коефіцієнтів ефективності

Використовуючи табл.2.4 та 2.5 будуються графіки зміни Ку и Кэ в залежності від N і P для ПРГ1 .

Рис.2.3. – Графік залежності коефіцієнту прискорення від кількості

процесорів

Рис.2.4. – Графік залежності ефективності прискорення від кількості

процесорів

**2.6. Висновки до розділу 2**

Виконано розробку програми ПРГ1 для ПКС СП з використанням мови C#.

Аналіз отриманих результатів дозволяє зробити наступні висновки:

• Коефіцієнти ефективності в переважній більшості експериментів

лежать в межах 0,83…0.98. Такі показники вказують на досить ефективну роботу найгірша при Р=3.

паралельного алгоритму. Іншими словами, час виконання приблизно обернено

пропорційний кількості процесорів: Tp ≈ T1 / P. Це свідчить про те, що

розроблені паралельні алгоритми складені правильно. Найкраща ефективність розпарелелювання при Р=2.

• Коефіцієнти прискорення лежать в межах 1,933…1,96; 2,566…2,6;

3,3576…3,59 для двох, трьох і чотирьох процесорів відповідно. Залежність

коефіцієнта прискорення від кількості процесорів має лінійний характер. Це достатньо високі показники, якщо прийняти до уваги той факт, що програми

функціонували в режимі відлагодження. Таке рішення дозволило отримати

більш повну картину в ході процесу підрахунку, однак зайняло деяку частину

ресурсів ЦП, що і відзначилось на результатах.

**РОЗДІЛ 3. РОЗРОБКА ПРОГРАМИ ПРГ2 ДЛЯ ПКС ЛП**

**3.1. Розробка паралельного математичного алгоритму**

5

1

3

2

4

Рис.3.1Структура ПКС

3.1. Розробка паралельного математичного алгоритму

Відповідно до умови рис.2.1 та заданої формули A=sort(a\*B+C\*(MX\*MD)) виконується розпаралелювання :

1)AH=a\*BH+C\*(MX\*MDH)

2)AH = Sort(АH)

3)A1,2 = mergeSort(AH,AH)

A3,4 = mergeSort(AH,AH)

4)A = mergeSort(A1,2, A3,4),

де H = N / P,

N – розмірність матриць,

P – кількість вузлів ПКС.

Структура зі спільною пам’яттю має спільні ресурси: a,C,MX

При розробці алгоритму N приймалось кратним Р. Щоб розрахувати мінімальний час виконання програми для Р процесорів використаємо теорему Мунро-Петерсона: Tp=] log2(m+1) [

де m – кількість бінарних операцій, Tр – час розв’язання задачі на Р вузлах.

2N+1 –для множення , 2N-1-для додавання

Підставимо отримане значення в формулу:

Tp1=] log2(m+1) [ = ] log2(2N+1+2N-1+1) [ = ] log(4N+1)[

Для сортування - N порівнянь

Tp2=]log2(N)[

Tp=Tp1+Tp2 = ] log2(4N+1) [ = ] log2(4N+1)[+]log2(N)[

**3.2. Розробка алгоритмів процесів**

Розробка алгоритмів процесів представлена в табл.3.1.

Табл. 3.1 Алгоритм процесів

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Т1 | Т2 | Т3 | Т4 | Т5 |
| Отримати С,MX,a від Т5 | Отримати С,MX,a від Т5 | Отримати С,MX,a від Т5 | Отримати С,MX,a від Т5 | Введення B,C,a,MX,MD |
| Отримати Bh , MDh від Т5 | Отримати Bh , MDh від Т5 | Отримати Bh , MDh від Т5 | Отримати Bh , MDh від Т5 | Переслати С,MX,a процесам Т1-Т4 |
| Обчислення AH=a\*BH+ C\* (MX\*MDH)  AH = Sort(AH) | Обчислення AH=a\*BH+ C\* (MX\*MDH)  AH = Sort(AH) | Обчислення AH=a\*BH+ C\* (MX\*MDH)  AH = Sort(AH) | Обчислення AH=a\*BH+ C\* (MX\*MDH)  AH = Sort(AH) | Переслати Bh , MDh процесам Т1-Т4 |
| Переслати Ah процесору Т5 | Отримати Аh від Т5 | Переслати Ah процесору Т5 | Отримати Аh від Т5 | Отримати Ah від Т1 |
|  | A1,2 = mergeSort(AH,AH) |  | A3,4=mergeSort(AH,AH) | Переслати Аh процесору Т2 |
|  | Переслати A1,2 процесору Т5 |  | Отримати A1,2 від Т5 | Отримати Ah від Т3 |
|  |  |  | A = mergeSort(A1,2, A3,4) | Переслати Аh процесору Т4 |
|  |  |  | Переслати A процесору Т5 | Отримати A1,2 від Т2 |
|  |  |  |  | Переслати А1,2 процесору Т4 |
|  |  |  |  | Отримати Aвід Т4 |
|  |  |  |  | Вивести А. |

## Розробка схеми взаємодії процесів

Схема взаємодії задач приведена на рисунку 3.2 . Квадратом позначені процеси. Стрілками показані сигнали, які відправляються від одного процесора іншому . Зміст сигналу вказується біля стрілки.

## Розробка програми ПРГ2

Лістинг програм знаходиться в додатку З

Програма складається з 2 класів: Coursework2 та Data .

Coursework2 клас, який містить головну функцію main, в якій і зосереджена основна логіка програми: пересилання і отримання повідомлень між задачами.

Tid - оприділяє номер потоку.

N- розмір матриць та векторів

P – кількість потоків

В Data зберігається допоміжні функції.

**3.5. Тестування програми**

Метою проведення тестування є оцінка коефіцієнтів прискорення і

коефіцієнтів ефективності для розроблених програм при їх виконанні на

реальній паралельній обчислювальній системі. Для визначення вищевказаних

коефіцієнтів проведений ряд експериментів із різними розмінностями

операндів і різною кількістю запущених задач. Розмірність операндів була

вибрана в межах від 800 до 1200.

Таблиця 3.3. – Час виконання програми обчислення функції

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | T1 | T2 | T3 | T4 |
| 800 | 1,10818 | 0,590359 | 0,459 | 0,3127 |
| 1000 | 2,14776 | 1,10973 | 0,85 | 0,5884 |
| 1200 | 3,7011 | 1,9135 | 1,43698 | 1,011478 |

На основі даних з табл.3.3 виконується розрахунок значень коефіцієнтів прискорення(ky=T1/T2, Ку= T1/T3, Ку = T1/T4), які відображаються в табл.3.4

Таблиця 3.4. – Значення коефіцієнтів прискорення

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | Кількість процесів | | | |
| 1 | 2 | 3 | 4 |
| 800 | 1 | 1,877129 | 2,414 | 3,5439 |
| 1000 | 1 | 1,935398 | 2,52677 | 3,65 |
| 1200 | 1 | 1,9342 | 2,5756 | 3,659 |

На основі даних в табл. 3.4 виконується розрахунок значень коефіцієнтів ефективності((Кэ= Ку/2 \* 100%, Ку= Ку/3\* 100%, , Ку = Ку/4\* 100%), які відображаються в табл. 3.5

Таблиця 3.5. – Значення коефіцієнтів ефективності

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | Кількість процесів | | | |
| 1 | 2 | 3 | 4 |
| 800 | 100 | 93,8556 | 80,467 | 88,6 |
| 1000 | 100 | 96,77 | 84,23 | 91,25 |
| 1200 | 100 | 96,71 | 85,85 | 91,475 |

Використовуючи табл.2.2 та 2.3 будуються графіки зміни Ку

и Кэ в залежності від N і P для ПРГ1 .

Рис.3.3. – Графік залежності коефіцієнту прискорення від кількості

процесорів

Рис.3.4. – Графік залежності ефективності прискорення від кількості

процесорів

**2.6. Висновки до розділу 3**

Аналіз отриманих результатів дозволяє зробити наступні висновки:

• Коефіцієнти ефективності в переважній більшості експериментів

лежать в межах 0,80…96. Такі високі показники вказують на досить ефективну

роботу паралельного алгоритму. Найкраща ефективність розпарелелювання при Р=2, найгірша при Р=3.

• Коефіцієнти прискорення лежать в межах 1,877129…1,9342; 2,414…2,5756; 3,5439…3,659 для двох, трьох і чотирьох процесорів відповідно. Залежність коефіцієнта прискорення від кількості процесорів має лінійний характер. Це достатньо високі показники, якщо прийняти до уваги той факт, що програми функціонували в режимі відлагодження. Таке рішення дозволило отримати більш повну картину в ході процесу підрахунку, однак зайняло деяку частину ресурсів ЦП, що і відзначилось на результатах. **4. Загальний висновок**

В першому розділі було розглянуто засоби роботи процесами в мові Python.

Було розглянуто що в Python є дочірні процеси, які можуть демонічними та не демонічними. Процес може ділитися на потоки, які працюють паралельно.

Паралельне програмування вимагає ретельного відпрацювання взаємодії між потоками управління. Деякі ділянки коду необхідно огороджувати від одночасного використання двома різними потоками, щоб не порушити цілісність змінюваних структур даних або логіку роботи із зовнішніми ресурсами. Для огорожі ділянок коду використовуються замки і семафори.

Стандартна бібліотека Python надає досить непоганий набір можливостей для багатопотокового програмування в модулях threading і thread, а також деякі корисні допоміжні модулі .

Другий розділ був присвячений розробці паралельної програми для паралельної комп’ютерної системи зі спільною пам’яттю. Перш ніж написати програму, було пройдено такі етапи як розробка паралельного математичного алгоритму, розробка алгоритмів процесів та розробка схеми взаємодії процесів. Для організації взаємодії процесів, згідно ТЗ, було використано засоби мови С#. Після написання програми було проведено її тестування, в результаті чого було отримано ряд дані, по яких було складені таблиці і вирахувані коефіцієнти прискорення та коефіцієнти завантаження. По даних таблиць було побудовано різні залежності у вигляді графіків. Оцінивши дані, можна сказати, що розроблені алгоритми є правильними.

Третій розділ є аналогічним до попереднього, але в третьому розділі відрізняється структура паралельної системи і відповідно механізм взаємодії процесів. Особливістю структури паралельної системи в цьому розділі є локальна пам’ять, тому програма була написана з використанням бібліотеки МРІ на мові програмування С++, який забезпечує обмін даними між процесами. Так само як і в попередньому розділі було розроблені алгоритми процесів та схема взаємодії процесів. Програму була протестована і як результат були отримані експериментальні дані, такі як час виконання програми, коефіцієнти прискорення та коефіцієнти завантаження. У порівнянні з попередніми даними час виконання програми значно менший, більш ніж в 4 рази, що пояснюється іншим середовищем програмування та іншою структурою паралельної обчислювальної системи.

Результати обох тестів показали що найнижчий коефіцієнт ефективності при Р=3, оскільки для тестування були використані значення матриці кратне 4 і матриця розпаралелювалася на 4 процеси. А найвищий для Р=2, а не Р=4, тому що деяку частину коду не можна розпаралелити на 4 процеси, а лише на 2.

Список літератури:

Бізлі Д.М. «Python. Подробный справочник»2010-860

Жуков І. А., Корочкін А. В. «Параллельные и распределенные вычисления.» – К.: «Корнейчук», 2005. – 226 с.

Жуков И. А., Корочкин А. В. «Параллельные и распределенные вычисления. Лабораторный практикум» – К.: «Корнейчук», 2008. – 224 с.

Лутц М. «Программирование на Python» 2002.-1136с.

Чаплигін А.Н. «Учимся програмировать вместе с Питоном»2004-110

Шилтд Г.«Полний справочник по С»

Среда параллельного программирования MPI <http://rsusu1.rnd.runnet.ru/tutor/method/m2/page02.html>

Основы программирования в Message Passing Interface (MPI) <http://www.ccas.ru/mmes/educat/lab04k/01/basics.html>

MPI для начинающих. <http://docstore.mik.ua/manuals/ru/MPI_intro/index.html>

Курс программирования C# Quick Guide™ <http://codingcraft.ru/c_sharp_coding/parallel.php>

Создание потоков (Руководство по программированию на C#) <http://msdn.microsoft.com/ru-ru/library/ms173178(v=vs.90).aspx>

Программирование на Cи шарп | C# http://sipikoandukebaka.appspot.com/life-prog.ru/view\_cat.php?cat=10