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»

Чередніченко С.С.

(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

Чередніченко Святослав Сергійович

(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 11 травня 2013 р.
2. Вихідні дані для роботи:

* Порівняння реалізації механізму моніторів у мовах і  
  бібліотеках паралельного програмування;
* математична задача ;
* структури ПКС СП та ПКС ЛП;
* бібліотеки програмування: OpenMP, MPI.

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

* Порівняння реалізації механізму моніторів у мовах і  
  бібліотеках паралельного програмування;
* розробка і тестування програми ПРГ1 для ПКС СП;
* розробка і тестування програми ПРГ2 для 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 | 20.03.2013 |
|  | Розробка паралельного алгоритму рішення задачі | 01.04.2013 |
|  | Розробка алгоритмів процесів | 06.04.2013 |
|  | Розробка схем взаємодії процесів | 13.04.2013 |
|  | Розробка програм | 20.04.2013 |
|  | Тестування програм | 30.04.2013 |
|  | Оформлення КР | 10.05.2013 |
|  | Захист КР | 18.05.2013 |

Студент

(підпис) (прізвище та ініціали)

Керівник роботи

(підпис) (прізвище та ініціали)

**ПОЯСНЮВАЛЬНА ЗАПИСКА**

ЗМІСТ

1. Порівняння реалізації механізму моніторів у мовах і  
   бібліотеках паралельного програмування 6
   1. Загальна інформація про монітор 6
   2. Монітори в Java 10
   3. Монітори в С# і VisualBasic 12
   4. Монітори в Ada 15
   5. Висновки до розділу 1 20
2. Розробка програми для ПОС з СП за допомогою

бібліотеки OpenMP 21

* 1. Розробка паралельного мат. алгоритму. 23
  2. Розробка алгоритму процесів. 23
  3. Розроблення структурної схеми взаємодії задач 26
  4. Розробка програми. 26
  5. Тестування програми. 28
  6. Висновки до розділу 2. 31

1. Розробка програми для ПОС з ЛП за допомогою

бібліотеки MPІ 32

* 1. Розробка паралельного мат. алгоритму. 34
  2. Розробка алгоритму процесів. 34
  3. Розроблення структурної схеми взаємодії задач 36
  4. Розробка програми. 38
  5. Тестування програми. 39
  6. Висновки до розділу 3. 43

ОСНОВНІ РЕЗУЛЬТАТИ І ВИСНОВКИ ДО РОБОТИ 44

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 45

ДОДАТКИ 46

1. **Порівняння реалізації механізму моніторів у мовах і  
   бібліотеках паралельного програмування**
   1. **Загальна інформація про монітор**

Монітор - в мовах програмування, високорівневий механізм взаємодії та синхронізації процесів, що забезпечує доступ до неподільні ресурсів. Підхід до синхронізації двох або більше комп'ютерних задач, що використовують спільний ресурс, апаратуру або набір змінних.

При багатозадачності, що грунтується на моніторах, компілятор або інтерпретатор прозоро для програміста вставляє код блокування-розблокування в відповідним чином оформлені процедури, прибираючи необхідність програмісту явно звертатися до примітивів синхронізації.

**Історія**

Пер Брінч Хансен був першим, хто описав і реалізував монітори, засновуючи їх на ідеях Хоара. Згодом Хоар розробив теоретичну основу і показав її еквівалентність семафору (використовуючи початкову семантику). Вперше втілений у мові Concurrent Pascal і використаний для структурування межпроцессної взаємодії в операційній системі Solo.

Мови програмування, що підтримують монітори:

* Ада
* C # (та інші мови, що використовують. NET Framework)
* Concurrent Pascal
* D
* Delphi (тип TMonitor)
* Java (за допомогою ключового слова synchronized або біблотекі java.util.concurrent)
* Mesa
* Модула-3
* Ruby
* Squeak Smalltalk
* uC + +

***Принципи, які покладені в основу та способи реалізації***

Монітор складається з:

* набору процедур, взаємодіючих із загальним ресурсом
* мьютекса
* змінних, пов'язаних з цим ресурсом
* інваріанта, який визначає умови, що дозволяють уникнути стан гонки

Процедура монітора захоплює мьютекс перед початком роботи і тримає його або до виходу з процедури, або до моменту очікування умови (див. нижче). Якщо кожна процедура гарантує, що перед звільненням мьютекса інваріант істина, то ніяка задача не може отримати ресурс в стані, що веде до гонки.

Щоб уникати стану активного очікування, процеси повинні сигналізувати один одному про очікувані події. Монітори забезпечують цю можливість за допомогою умовних змінних. Коли процедура монітора вимагає для подальшої роботи виконання певної умови, вона чекає пов'язану з ним умовну змінну. Під час очікування вона тимчасово відпускає мьютекс і вибуває із списку що виконуються . Будь-який процес, який в подальшому призводить до виконання цієї умови, використовує умовну змінну для оповіщення чекаючего її процесу. Сповіщений процес захоплює мьютекс знову і може продовжувати роботу.

Постулюються три основні властивості монітора:

1. Структури даних, що входять в монітор, можуть бути доступні тільки для процедур, що входять в цей монітор (таким чином, монітор являє собою певний аналог об'єкта в об'єктно-орієнтованих мовах і реалізує інкапсуляцію даних)
2. Процес «входить» у монітор шляхом виклику однієї з його процедур
3. У будь-який момент часу всередині монітора може перебувати не більше одного процесу. Якщо процес намагається потрапити в монітор, в якому вже знаходиться інший процес, він блокується. Таким чином, щоб захистити роздільні структури даних, їх досить помістити всередину монітора разом з процедурами, що представляють критичні секції для їх обробки.

Так, як монітор являє собою конструкцію мови програмування тоді, компілятору відомо про те, що вхідні в нього процедури і дані мають особливу семантику, тому перша умова може перевірятися ще на етапі компіляції. Крім того, код для процедур монітора теж може генеруватися особливим чином, щоб задовольнялася третя умова. Оскільки організація взаємного виключення в даному випадку покладається на компілятор, кількість програмних помилок, пов'язаних з організацією взаємного виключення, зводиться до мінімуму.

*Семантика Хоара та Mesa*

У ранніх реалізаціях монітора (відомих як семантика Хоара), оповіщення умовної змінної негайно активізує очікуючий процес і відновлює блокування, тим самим гарантується, що умова все ще істинна.

Реалізація цієї поведінки складна і дуже надлишкова. Також вона не сумісна з витісняючою багатозадачністю, коли процес може бути перерваний у довільний момент. З цих причин дослідники розробили безліч інших семантик для умовних змінних.  
У самих сучасних реалізаціях (відомих як семантика Mesa), сповіщення не перериває працюючий процес, а просто переводить деякі чекають процеси в стан готовності.

Сповіщаючий процес продовжує тримати блокування до тих пір, поки не вийде з процедури монітора. Побічні ефекти цього підходу в тому, що сповіщаючий процес не зобов'язаний дотримуватись інваріанту перед оприлюдненням, а чекаючий процес - повинен повторно перевірити умову,виконання якої він чекає. Зокрема, якщо процедура монітора включає вираз if test then wait (cv), інший процес може увійти в монітор після моменту сповіщення і змінити значення test до того, як чекаючий процес відновить роботу. Вираз потрібно переписати так: while test do wait (cv), щоб умова була пере-перевірено після очікування.

Реалізації також надають операцію «notifyAll», або «broadcast», яка сповіщає всі процеси, що чекають дану умову. Ця операція корисна, наприклад, коли декілька процесів чекають доступності різних обсягів пам'яті. Звільнення пам'яті дозволить продовжити роботу якогось із них, але планувальник не може знати, якого саме.

* 1. **Монітори в Java**

У кожного об'єкта в Java є свій власний неявний монітор. Коли метод типу synchronized викликається для об'єкта, відбувається звернення до монітора об'єкта щоб визначити, чи виконує будь-який інший потік в даний момент метод типу synchronized для даного об'екта. Якщо ні, то данний потік отримує дозвіл увійти в монітор.Вход в монітор називається також блокуванням (locking) монітора. Якщо при цьому інший потік вже увійшов в монітор, то данний потік повинен чекати до тих пір, поки інший потік не покине монітор. Таким чином монітор Java вводить почерговість у паралельну обробку. Цей спосіб називається також перетворенням в послідовну форму (serialization).

Оголошення методу synchronized не має на увазі, що тільки один потік може одночасно виконувати цей метод, як у випадку критичного ділянки (critical sections). Мається на увазі, що в будь-який момент часу тільки один потік може викликати цей метод (або будь-який інший метод типу synchronized) для конкретного об'єкта. Таким чином, монітори Java пов'язані з об'єктами, але не з блоками коду. Два потоки можуть паралельно виконувати один і той же метод типу synchronized за умови, що цей метод викликаний для різних об'єктів.  
Монітори не є об'єктами мови Java, у них немає атрибутів або методів. Доступ до моніторів можливий на рівні власного коду JVM.

У Java є два способи синхронізації потоків.

1. Створення синхронізуючого методу всередині класу.  
Використовується при наявності методу або групи методів, що обробляють внутрішній стан об'єкта в многопоточной сітуаціі.Для організації послідовного доступу потоків до методу оголошення методу випереджається ключовим словом synchronized.

class Callme{

  synchronized void call(String msg){

  . . . . . . . . . . . . . . . . . .

  . . . . . . . . . . . . . . . . . .

  }

2. Створення синхронізуючого блоку. Використовується для організації доступу до об'єктів класу, не розробленого для багатопотокового доступу (наприклад до масивів) або створеного іншим програмістом (немає доступу до вихідного коду). Необхідно помістити виклики методів в синхронізуючий блок шляхом використання оператора synchronized, що має наступний синтаксис:

        synchronized (object){

// операторы,которые необходимо синхронизировать

        }

Оператор synchronized корисний при безпосередній зміні загальних змінних об'єкта.

   void call(SomeClassobj){

   synchronized(obj){

     obj.variable=5;

     }

   }

* 1. **Монітори в С# і VisualBasic**

Об'єкти Монітор в С # надають можливість синхронізувати доступ до області коду за допомогою отримання і звільнення блокування на заданому об'єкті за допомогою методів Monitor. Enter, Monitor.TryEnter і Monitor.Exit. Методи Monitor.Wait, Monitor.Pulse і Monitor.PulseAll можна використовувати, як тільки на області коду встановлено ​​блокування. Метод Wait знімає блокування, якщо воно встановлено​​, і очікує оповіщення. При отриманні сповіщення метод Wait повертається і знову отримує блокування. Pulse і PulseAll сигналізують про перехід до наступного потоку в черзі очікування.

Інструкції SyncLock в Visual Basic і lock в C # використовують метод Monitor.Enter для установки блокування і метод Monitor.Exit для її зняття. Перевага використання інструкцій мови полягає в тому, що весь вміст інструкції lock або SyncLock включається в інструкцію Try. Інструкція Try забезпечується блоком Finally, що гарантує зняття блокування.

Monitor блокує об'єкти (зі показниковим типом), а не типи значень. Оскільки методам Enter і Exit можна передати тип значення, він упаковується окремо для кожного виклику. Оскільки при кожному виклику створюється окремий об'єкт, метод Enter ніколи не блокується, і код, імовірно захищається цим методом, насправді не є синхронізованим. Крім того об'єкт, переданий методу Exit, відрізняється від об'єкта, переданого методу Enter, тому Monitor створює виняток SynchronizationLockException з повідомленням "Метод синхронізації об'єкта викликаний з несинхронізованого блоку коду". У наступному прикладі показані ці несправності.

**С#**

try

{

int x = 1;

// виклик Enter() створює generic синхронізуючий об'єкт для значення

// x кожен раз коли виконується код, тому Enter ніколи не заблокований.

Monitor.Enter(x);

try

{

// код який потребує захисту монітором

}

finally

{

// виклик Exit() призведе до помилки

// синхронізуючий об'єкт створений для х при виклику Exit() буде відрізнятись

// від об'єкту використаного для Enter().

// буде згенеровано помилку SynchronizationLockException.

Monitor.Exit(x);

}

}

catch (SynchronizationLockException SyncEx)

{

Console.WriteLine("A SynchronizationLockException occurred. Message:");

Console.WriteLine(SyncEx.Message);

}

**VB**

Try

Dim x As Integer = 1

' виклик Enter() створює generic синхронізуючий об'єкт для значення

' x кожен раз коли виконується код, тому Enter ніколи не заблокований.

Monitor.Enter(x)

Try

' код який потребує захисту монітором

Finally

' виклик Exit() призведе до помилки

' синхронізуючий об'єкт створений для х при виклику Exit() буде відрізнятись

' від об'єкту використаного для Enter().

' буде згенеровано помилку SynchronizationLockException.

Monitor.Exit(x)

End Try

Catch SyncEx As SynchronizationLockException

Console.WriteLine("A SynchronizationLockException occurred. Message:")

Console.WriteLine(SyncEx.Message)

End Try

Необхідно згадати про відмінності у використанні об'єктів Monitor і WaitHandle. Об'єкти Monitor є повністю керованими і стерпним. Крім того, вони більш ефективні відносно вимог до ресурсів операційної системи. Об'єкти WaitHandle представляють об'єкти очікування операційної системи і використовуються при синхронізації керованого і некерованого коду, вони також надають деякі додаткові можливості операційної системи, наприклад можливість очікування одночасно великої кількості об'єктів.

*Відмінність від Java*

В обох мовах є можливість створити синхронно виконуваний блок коду; в Java це робиться за допомогою оператора synchronized (), в C # - оператором lock (). У Java є також можливість оголошувати синхронні методи, використовуючи модифікатор synchronized в заголовку опису методу. Такі методи при виконанні блокують свій об'єкт-господар (таким чином, з синхронізованих методів класу, для одного і того ж екземпляра, одночасно може виконуватися тільки один, інші будуть чекати). Аналогічна можливість в. NET реалізується за допомогою атрибута реалізації методу MethodImplAttribute MethodImplOptions.Synhronized, але, на відміну від Java, ця можливість формально не є частиною мови C #.

В обох мовах доступні також ідентичні засоби синхронізації, засновані на відправку і очікуванні сигналу від одного потоку до іншого (іншим). У Java це методи notify (), notifyAll () і wait (), в C # - методи Pulse (), PulseAll (), Wait () (трійки методів функціонально попарно аналогічні). Різниця полягає лише в тому, що в Java ці методи (і, відповідно, функціональність монітора) реалізується в класі Object, тому для синхронізації не потрібно ніяких додаткових бібліотек, а в C # ці методи реалізовані як статичні в окремому бібліотечному класі Monitor.

* 1. **Монітори в Ada**

Засобом, який дозволяє організувати взаємновиключаючий доступ до даних з різних задач, що виконуються одночасно, є захищені модулі, які були введені стандартом Ada95. Характерною особливістю захищених модулів є забезпечення ними синхронізованого доступ до приватних даних, проте в протилежність задачам, які є активними сутностями, захищені модулі - пасивні.

Захищені модулі (типи і об'єкти) Ади інкапсулюють дані і дозволяють здійснювати доступ до них за допомогою захищених підпрограм або захищених входів. Стандарт мови гарантує, що в результаті виконання коду таких підпрограм і входів зміна вмісту даних буде здійснюватися в режимі взаємного виключення без необхідності створення додаткового завдання.

Захищений модуль може бути описаний як захищений тип або як одиночний захищений об'єкт, що аналогічно одиночній задачі. В останньому випадку передбачається, що захищений об'єкт має відповідний анонімний тип. Слід враховувати, що захищений тип є лімітованим і, таким чином, не володіє визначеними операціями присвоювання або порівняння.  
  
Подібно задачі або пакету, захищений модуль має специфікацію і тіло. Специфікація описує протокол доступу до захищеного модулю (інтерфейс), і може містити специфікації процедур, функцій і входів захищеного модуля. Тіло описує деталі реалізації протоколу доступу до даних захищеного модуля і, як правило, містить тіла захищених підпрограм і входів. Подібно задачам і записам, захищений модуль може мати дискримінант дискретного або вказівного типу.

В якості простої ілюстрації запропоновано приклад наступного одиничного захищеного об'єкта:

-- специфікація захищеного об'єкта

**protected** Variable **is**

**function** Read **return** Item;

**procedure** Write(New\_Value : Item);

**private**

Data : Item;

**end** Variable;

-- тіло захищеного об'єкта

**protected body** Variable **is**

**function** Read **return** Item **is**

**begin**

**return** Data;

**end**;

**procedure** Write (New\_Value : Item) **is**

**begin**

Data := New\_Value;

**end**;

**end** Variable;

Захищений об'єкт Variable надає керований доступ до приватної змінної Data типу Item. Функція Read дозволяє читати, а процедура Write - оновлювати поточне значення змінної Data.

Захищаються дані і дані про стан об'єкта повинні бути поміщені в приватну частину специфікації. Сенс цього полягає в тому, що приватна частина не доступна клієнтові безпосередньо, а наявність всієї інформації в інтерфейсі захищеного об'єкта необхідно компілятору для ефективного розподілу пам'яті.

Захищені процедури передбачають взаємно виключаючий доступ до даних захищеного модуля по читанню і / або запису. Захищені функції надають одночасний доступ до даних захищеного модуля тільки з читання, що передбачає одночасне виконання безлічі викликів функцій. Однак виклики захищених процедур і захищених функцій залишаються взаємно виключаючими. Порядок, в якому різні завдання очікують виконання захищених процедур і захищених функцій, стандартом не визначається. Однак підтримка вимог додатка D (Annex D) стандарту Ada95, в якому зазначені вимоги для систем реального часу, дозволяє зробити деякі припущення про можливий порядок виконання підпрограм.

Для звернення до захищених підпрограм використовується традиційна точкова нотація:

X := Variable.Read;

. . .

Variable.Write (New\_Value => Y);

Усередині тіла захищеного об'єкта допускається кілька підпрограм, при цьому реалізація Ада-системи буде гарантовано здійснювати виклики підпрограм за принципом взаємного виключення (подібно монітору).

Входи захищених об'єктів мають атрибути, призначення яких подібно до призначеннь атрибутів для входів завдань:  
  
E'Caller - Повертає значення типу Task\_ID, яке ідентифікує оброблювану в поточний момент задачу, що звернулися на вхід захищеного об'єкта E. Використання цього атрибута допустимо тільки всередині тіла входу (для захищеного об'єкта).  
E'Count - Повертає значення типу Universal\_Integer, що показує число звернень на вході E, які перебувають у черзі.  
  
Тут, мається на увазі, що E - це ім'я будь-якого входу захищеного об'єкта. Так само як і у випадку задач, при використанні цих засобів слід враховувати, що після закінчення деякого часу ніщо не гарантує активність задачі або її присутність в області видимості.

Захищені входи і бар'єри  
  
За аналогією зі входами задач, захищений модуль може мати захищені входи. Дії, що виконуються при виклику захищеного входу, передбачаються в його тілі. Захищені входи подібні захищеним процедурам тому, що вони гарантують взаємно виключний доступ до даних захищеного модуля з читання та / або запису. Однак всередині тіла захищеного модуля захищені входи охороняються логічним виразом, яке називають бар'єром, а результат обчислення цього логічного виразу повинен мати визначений логічний тип Standard.Boolean.

Якщо при виклику захищеного входу значення бар'єру є False, то виконання задачі що викликає захищений вхід припиняється до тих пір, поки значення бар'єру не стане рівним True і всередині захищеного модуля будуть відсутні активні задачі (задачі, які виконують тіло якогось захищеного входу або який-небудь захищеної підпрограми). Отже, виклик захищеного входу може бути використаний для реалізації умовної синхронізації.

Можна помітити, що існує сувора паралель між інструкцією прийняття з охоронною умовою з тіла задачі і тілом захищеного входу з бар'єрною умовою для захищеного модуля. При цьому витрати часу на перевірку бар'єрної умови для захищеного входу значно менше, ніж витрати часу на перевірку охороннї умови інструкції прийняття.

* 1. **Висновки до розділу 1**

1. Виконано аналіз моніторів у мові Java. Показано, у кожного об'єкта в Java є свій власний неявний монітор. Це зумовлено тим, що в Java все наслідується від суперкласу Object, який містить реалізацію всіх методів необхідних монітору, тому для синхронізації не потрібно ніяких додаткових бібліотек.
2. Виконано аналіз моніторів у мовах С# та VisualBasic. В цих мовах програмування присутній статичний бібліотечний клас Монітор який містить всі необхідні методи для забезпечення синхронізації потоків і задач.
3. Виконано аналіз моніторів у мові Ада. В цій мові Монітори реалізовані у вигляді захищених модулів , які були введені стандартом Ada95. Характерною особливістю захищених модулів є забезпечення ними синхронізованого доступ до приватних даних, проте в протилежність задачам, які є активними сутностями, захищені модулі – пасивні сутності.
4. **Розробка програми для ПОС з СП за допомогою бібліотеки OpenMP**

Математична функція: .

Структура Паралельної Обчислювальної Cистеми з СП:

ОП

1

3

2

2

4

4

3

Рис. 2.1 – Структура ПОС

**Обчислення мінімального часу виконання математичної функції.**

Розмірність вхідних матриць NxN тобто матриці квадратні. Відповідно розмірність векторів N.

Основні операції які виконуються при обчисленні функції – множення матриць( відбувається за час ), множення вектора на матрицю (відбувається за час ), додавання векторів, сортування злиттям (відбувається за час ).

Обчислення мінімального часу відбувається поетапно.

* обчислюється за час
* обчислюється за час
* обчислюється за час
* обчислюється за час
* обчислюється за час

Отже для 1 процесора мінімальний час обчислення .

Для 2х процесорів мінімальний час обчислення складе .

Для 3х процесорів мінімальний час обчислення складе .

Для 4х процесорів мінімальний час обчислення складе .

Коефіцієнт прискорення:



Відповідно ,

Коефіцієнт ефективності:



Відповідно,

* 1. **Розробка паралельного мат. алгоритму.**

= sort (ZH)

YH\*2 =mergeSort(, )

A = mergeSort (YH\*2, YH\*2);

- Н рядків/стовпчиків матриці

- Н рядків/стовпчиків матриці

Спільні ресурси: ,,.

* 1. **Розробка алгоритму процесів.**

|  |  |
| --- | --- |
| Т1 | КД |
| 1. Очікування сигналу завершення вводу 2. Копія,, 3. **Чекати** на завершення обчислення в інших процесах 4. **Чекати** на завершення обчислення в інших процесах 5. Вивід даних () | Бар’єр  КД  Бар’єр  Бар’єр |

|  |  |
| --- | --- |
| Т2 | КД |
| 1. Ввід даних (C) 2. Очікування сигналу завершення вводу 3. Копія,, 4. **Чекати** на завершення обчислення в інших процесах 5. **Чекати** на завершення обчислення в інших процесах | Бар’єр  КД  Бар’єр  Бар’єр |

|  |  |
| --- | --- |
| Т3 | КД |
| 1. Ввід даних (MX, MO) 2. Очікування сигналів від завершення вводу 3. Копія,, 4. **Чекати** на завершення обчислення в інших процесах | Бар’єр  КД  Бар’єр |

|  |  |
| --- | --- |
| Т4 | КД |
| 1. Ввід даних (MZ) 2. Очікування сигналу завершення вводу 3. Копія,, 4. **Чекати** на завершення обчислення в інших процесах | Бар’єр  КД  Бар’єр |

* 1. **Розроблення структурної схеми взаємодії задач**

Структурна схема взаємодії задач зображена на рис. 2.2 (Додаток А)

Перший етап роботи програми – ввід даних. З рисунку можна побачити, що на етапі введення даних ввід відбувається лише задачах №2,№3,№4, при цьому синхронізація по вводу даних відбувається за допомогою бар'єра.

Другий етап роботи програми – копіювання спільних ресурсів всіми задачами. Синхронізація доступу до спільних ресурсів відбувається за допомогою критичної секції.

Третій етап роботи програми – обчислення математичної функції. Етап у свою чергу поділяється на обчислення часткового результату і сортування злиттям часткових результатів. Синхронізація між задачами сортування злиттям часткових результатів відбувається за допомогою бар'єрів.

Четвертий етап роботи програми – вивід результату. Виконується в задачі №2.

* 1. **Розробка програми.**

До складу програми входять процедури і функції, а також змінні, які забезпечують паралельне обчислення заданої математичної функції.

*Змінні:*

int\*\* MZ – Матриця MZ;

int\*\* MO – Матриця MO;

int\*\* MX – Матриця MX(спільний ресурс);

int\* C – вектор С (спільний ресурс);

int\* B – вектор B (спільний ресурс);

int\* A – вектор A (Результат);

*Функції задач:*

int task1() – функція, що виконує алгоритм задачі №1;

int task2() – функція, що виконує алгоритм задачі №2;

int task3() – функція, що виконує алгоритм задачі №3;

int task4() – функція, що виконує алгоритм задачі №4;

*Функції/процедури вводу/виводу і обчислення результату:*

int\* Input\_V(int n) – функція вводу вектору всі елементи якого = n;

int\*\* Input\_M(int n) – функція вводу матриці всі елементи якої = n;

void Output\_V(int\* v) – процедура виводу вектору;

int\*\* Copy\_M(int\*\* MX) – функція копіювання матриці;

int\* Copy\_V(int\* V) – функція копіювання вектору;

void SumV(int\* a, int\* b, int\* res, int m, int m1) – процедура додавання 2-х векторів;

int\*\* MulM(int\*\* a, int\*\* b, int m, int m1) – функція множення 2-х матриць;

int\* MulVM(int\* a, int\*\* b, int m, int m1) – функція множення вектору на матрицю;

static void MergeSort(int\* vector, int l, int r) – процедура сортування злиттям.

* 1. **Тестування програми.**

Для тестування використовувалась паралельна обчислювальна система з наступними апаратними характеристиками:

− процесор: Intel core i5 - 2400 (4 ядер по 3,1 Ггц);

− оперативна пам'ять: DDR3 1333 МГц, 3327 МБ.

В якості програмного забезпечення виступали:

− операційна система: Microsoft Windows 7 x86 SP1 (Version 6.7.7601);

− середовище розробки і компіляції C++ програми: Microsoft Visual Studio 2010 (Version 10.0.30319.1 RTMRel).

В таблиці 2.1 приведені результати тестування програми для ПОС з СП

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | Р=1 | Р=2 | Р=3 | Р=4 |
| 800 | 2,6765 | 1,6326 | 1,3489 | 1,0431 |
| 1600 | 34,5195 | 17,9520 | 13,5391 | 9,2925 |
| 2400 | 121,2760 | 62,6027 | 47,6971 | 32,6321 |

*Таблиця 2. 1-Час обчислення функції за допомогою бібліотеки OpenMP*

В таблиці 2.2 приведені результати розрахунків коефіцієнту прискорення для ПОС з СП

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | Р=1 | Р=2 | Р=3 | Р=4 |
| 800 | - | 1,6394 | 1,9842 | 2,5659 |
| 1600 | - | 1,9229 | 2,5496 | 3,7148 |
| 2400 | - | 1,9372 | 2,5426 | 3,7165 |

*Таблиця 2. 2-* *коефіцієнт прискорення*

*Рис. 2. 1 Графіки зміни коефіцієнту прискорення Кп в залежності від кількості ядер.* *Операція*

*Рис. 2. 2 Графіки зміни коефіцієнту прискорення Кп в залежності від розмірності матриць. Операція*

В таблиці 2. 3 приведені результати розрахунків коефіцієнту ефективності для ПОС з СП

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | Р=1 | Р=2 | Р=3 | Р=4 |
| 800 | - | 81,97 | 66,14 | 64,15 |
| 1600 | - | 96,14 | 84,99 | 92,87 |
| 2400 | - | 96,86 | 84,75 | 92,91 |

*Таблиця 2. 3- коефіцієнт ефективності*

*Рис. 2. 3 Графіки зміни коефіцієнту прискорення Кп в залежності від кількості ядер. Операція*

*Рис. 2. 4 Графіки зміни коефіцієнту прискорення Кп в залежності від розмірності матриць. Операція*

* 1. **Висновки до розділу 2.**

Виконано розробку програми ПРГ1 для ПОС ОП з використанням мови С++ і засобів синхронізації з бібіліотеки OpenMP. Тестуівння програмипоказало наступне:

* використання багатоядерної ПОС та програми ПРГ1 забезпечує скорочення часу обчислення заданої математичної задачі. Значення Кп лежать в межах від 1,6394 до 3,7165;
* максимальне значення Кп = 3,7165 забезпечує ПКС з Р=4 та N=2400 ;
* мінімальне значення Кп = 1,6394 при Р = 2 та N = 800
* з ростом N Кп зростає до N=1600, при подальшому зростанні N Кп практично не змінний.
* максимальне значення Ке = 96,86 забезпечує ПКС з Р=2 та N=2400 ;
* мінімальне значення Ке = 64,15 при Р = 4 та N = 800
* з ростом N Ке зростає до N=1600, при подальшому зростанні N Кп практично не змінний.

1. **Розробка програми для ПОС з ЛП за допомогою бібліотеки MPІ**

Математична задача:

.

Структура Паралельної Обчислювальної Cистеми з ЛП:

 Рис. 3.1 – Структура ПКС з ЛП

**Обчислення мінімального часу виконання математичної функції.**

Розмірність вхідних матриць NxN тобто матриці квадратні. Відповідно розмірність векторів N.

Основні операції які виконуються при обчисленні функції – множення матриць( відбувається за час ), множення вектора на матрицю (відбувається за час ), додавання векторів, сортування злиттям (відбувається за час ).

Обчислення мінімального часу відбувається поетапно.

* обчислюється за час
* обчислюється за час
* обчислюється за час
* обчислюється за час
* обчислюється за час

Отже для 1 процесора мінімальний час обчислення .

Для 2х процесорів мінімальний час обчислення складе .

Для 3х процесорів мінімальний час обчислення складе .

Для 4х процесорів мінімальний час обчислення складе .

Коефіцієнт прискорення:



Відповідно ,

Коефіцієнт ефективності:



Відповідно,

* 1. **Розробка паралельного мат. алгоритму.**

= sort (ZH)

YH\*2 =mergeSort(, )

A = mergeSort (YH\*2, YH\*2);

- Н рядків/стовпчиків матриці

- Н рядків/стовпчиків матриці

Спільні ресурси: ,,.

* 1. **Розроблення алгоритмів роботи кожного процесу**

Задача T1

1. **Передати**  задачі T2
2. **Прийняти**  від задачі T2
3. Обчислення
4. **Прийняти** результат  від задачі T2
5. **Прийняти** результат  від задачі T2
6. Вивід даних ()

Задача T2

1. **Прийняти**  від задачі T1
2. **Передати**  задачі T1
3. **Передати**  задачі T3
4. **Передати**  задачі T3
5. Обчислення
6. **Передати** задачі T1
7. **Прийняти** результат  від задачі T3
8. **Передати**  задачі T1

Задача T3

1. **Прийняти**  від задачі T2
2. **Прийняти**  від задачі T2
3. **Передати**  задачі T4
4. **Передати**  задачі T4
5. **Передати**  задачі T5
6. **Передати**  задачі T5
7. Обчислення
8. **Прийняти** результат від задачі T4
9. **Прийняти** результат від задачі T5
10. **Передати** результат задачі T2

Задача T4

1. **Прийняти**  від задачі T3
2. **Прийняти**  від задачі T3
3. Обчислення
4. **Передати** результат задачі T3

Задача T5

1. **Прийняти**  від задачі T3
2. **Прийняти**  від задачі T3
3. Обчислення
4. **Передати** результат задачі T3
   1. **Розробка структурної схеми взаємодії задач**

Структурна схема взаємодії задач зображена на рис. 3.2 (Додаток А)

* *Задача №1*

Здійснює ввід векторів В, С, а також вивід результату А

Відсилає задачі №2 спільний ресурс - вектори В, С.

Send\_fromT2 **приймає** спільний ресурс і частини матрицьвід задачі №2.

Send\_mSortH **приймає** результат  від задачі №2.

Send\_mSort3H **приймає** результат  від задачі №2.

* *Задача №2*

Здійснює ввід матриць МО, МХ,МZ.

Send\_fromT1 **приймає** спільний ресурс  від задачі №1.

Відсилає спільний ресурс і частини матрицьзадачі №1.

Відсилає спільний ресурс і частини матриць задачі №3

Відсилає спільний ресурс  задачі T3

Відсилає результат задачі №1

Send\_fromT3 **приймає** результат  від задачі T3

Відсилає відсортований частковий результат  задачі №1

* Задача №3

Send\_fromT2 **приймає** спільний ресурс і частини матриць від задачі №2

**Передає** спільний ресурс і частини матриць від задачі №4

**Передає** спільний ресурс і частини матриць від задачі №5

Send\_fromT4 приймає результат від задачі №4

Send\_fromT4 приймає результат від задачі №5

**Передає** відсортований частковий результат задачі №2

* Задача №4

Send\_fromT3 приймає спільний ресурс і частини матриць від задачі №3

**Передає** результат задачі №3

* Задача №5

Send\_fromT3 приймає спільний ресурс і частини матриць від задачі №3

**Передає** результат задачі №3

* 1. **Розробка програми**

До складу програми входять процедури і функції, а також змінні, які забезпечують паралельне обчислення заданої математичної функції.

*Функції задач:*

int task1() – функція, що виконує алгоритм задачі №1;

int task2() – функція, що виконує алгоритм задачі №2;

int task3() – функція, що виконує алгоритм задачі №3;

int task4() – функція, що виконує алгоритм задачі №4;

int task5() – функція, що виконує алгоритм задачі №5;

*Функції/процедури вводу/виводу і обчислення результату:*

int \*alloc\_V(int size,int number) – функція вводу вектору всі елементи якого = n, розмір = size;

int \*\*alloc\_M(int rows, int cols, int number) – функція вводу матриці всі елементи якої = n, кількість рядків = rows, кількість стовпців = cols;

void Output\_V(int\* v) – процедура виводу вектору;

int\*\* Copy\_M(int\*\* MX) – функція копіювання матриці;

int\* Copy\_V(int\* V) – функція копіювання вектору;

void SumV(int\* a, int\* b, int\* res, int m, int m1) – процедура додавання 2-х векторів;

int\*\* MulM(int\*\* a, int\*\* b, int m, int m1) – функція множення 2-х матриць;

int\* MulVM(int\* a, int\*\* b, int m, int m1) – функція множення вектору на матрицю;

static void MergeSort(int\* vector, int l, int r) – процедура сортування злиттям.

* 1. **Тестування**

Для тестування використовувалась паралельна обчислювальна система з наступними апаратними характеристиками:

− процесор: Intel core i5 - 2400 (4 ядер по 3,1 Ггц);

− оперативна пам'ять: DDR3 1333 МГц, 3327 МБ.

В якості програмного забезпечення виступали:

− операційна система: Microsoft Windows 7 x86 SP1 (Version 6.7.7601);

− середовище розробки і компіляції C++ програми: Microsoft Visual Studio 2010 (Version 10.0.30319.1 RTMRel).

В таблиці 3.1 приведені результати тестування програми для ПОС з СП

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | Р=1 | Р=2 | Р=3 | Р=4 |
| 800 | 2,6765 | 1,6326 | 1,3489 | 1,0431 |
| 1600 | 34,5195 | 17,9520 | 13,5391 | 9,2925 |
| 2400 | 121,2760 | 62,6027 | 47,6971 | 32,6321 |

*Таблиця 3.1-Час обчислення функції за допомогою бібліотеки OpenMP*

В таблиці 3.2 приведені результати розрахунків коефіцієнту прискорення для ПОС з СП

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | Р=1 | Р=2 | Р=3 | Р=4 |
| 800 | - | 1,791262 | 2,228458 | 2,822857 |
| 1600 | - | 1,905373 | 2,770495 | 3,214403 |
| 2400 | - | 1,952718 | 2,920475 | 3,158355 |

*Таблиця 3.2-* *коефіцієнт прискорення*

*Рис. 3.1 Графіки зміни коефіцієнту прискорення Кп в залежності від кількості ядер.* *Операція*

*Рис. 3.2 Графіки зміни коефіцієнту прискорення Кп в залежності від розмірності матриць. Операція*

В таблиці 3.3 приведені результати розрахунків коефіцієнту ефективності для ПОС з СП

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | Р=1 | Р=2 | Р=3 | Р=4 |
| 800 | - | 89,5631 | 74,2819 | 70,5714 |
| 1600 | - | 95,2687 | 92,3498 | 80,3601 |
| 2400 | - | 97,6359 | 97,3492 | 78,9589 |

*Таблиця 3.3- коефіцієнт ефективності*

*Рис. 3.3 Графіки зміни коефіцієнту прискорення Кп в залежності від кількості ядер. Операція*

*Рис. 3.4 Графіки зміни коефіцієнту прискорення Кп в залежності від розмірності матриць. Операція*

* 1. **Висновки до розділу 3.**

Виконано розробку програми ПРГ2 для ПОС ЛП з використанням мови С++ і засобів синхронізації з бібіліотеки OpenMP. Тестуівння програмипоказало наступне:

* використання багатоядерної ПОС та програми ПРГ2 забезпечує скорочення часу обчислення заданої математичної задачі. Значення Кп лежать в межах від 1,791262 до 3,214403;
* максимальне значення Кп = 3,214403 забезпечує ПКС з Р=4 та N=1600 ;
* мінімальне значення Кп = 1,791262 при Р = 2 та N = 800
* з ростом N Кп зростає до N=1600, при подальшому зростанні N Кп зростає повільніше, для випадку Р = 4 Кп зпадає.
* максимальне значення Ке = 97,6359 забезпечує ПКС з Р=2 та N=2400 ;
* мінімальне значення Ке = 70,5714 при Р = 4 та N = 800
* з ростом N Ке зростає до N=1600, при подальшому зростанні N Кп зростає повільніше, для випадку Р = 4 спостерігається незначне зпадання Кп.

**Основні результати і висновки по роботі.**

1. Виконано аналіз моніторів у мові Java. Показано, у кожного об'єкта в Java є свій власний неявний монітор. В мовах програмування С# та VisualBasic присутній статичний бібліотечний клас Монітор який містить всі необхідні методи для забезпечення синхронізації потоків і задач. В мові Ада Монітори реалізовані у вигляді захищених модулів , які були введені стандартом Ada95.
2. Використання багатоядерної ПОС та програми ПРГ1 забезпечує скорочення часу обчислення заданої математичної задачі. Значення Кп лежать в межах від 1,6394 до 3,7165
3. Використання багатоядерної ПОС та програми ПРГ2 забезпечує скорочення часу обчислення заданої математичної задачі. Значення Кп лежать в межах від 1,791262 до 3,214403;
4. Програми ПРГ1 і ПРГ2 показують приблизно однакові показники ефективності для випадку Р=2,3. Для випадку

Р = 4 і N=1600, 2400 ПРГ1 показує кращі показники, ніж ПРГ2. Такий результат зумовлений тим, що ПРГ2 використовує Локальну Пам'ять, тобто потрібен додатковий час для того, щоб розповсюдити вхідні дані між процесами, в той час коли ПРГ1 використовує Спільну Пам'ять, що мінімізує час отримання вхідних даних процесами.

**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

1. Використання мови Java для розробки паралельних програм <http://www.csa.ru/CSA/tutor/javathread.html>
2. Відкрита мульти-обробка

<http://en.wikipedia.org/wiki/OpenMP>

1. Гавва А. «Адское» программирование. Ada-95. Компилятор GNAT (2004)
2. Інтерфейс передавання повідомлень

http://en.wikipedia.org/wiki/Message\_Passing\_Interface

1. Монітори в C# та інших мовах, які використовують .NET Framework

<http://msdn.microsoft.com/ru-ru/library/hf5de04k.aspx?cs-save-lang=1&cs-lang=csharp#code-snippet-1>

1. Опис моніторів [http://en.wikipedia.org/wiki/Monitor\_%28synchronization%2](http://en.wikipedia.org/wiki/Monitor_%28synchronization%252)
2. Порівняння мов C# і Java <http://en.wikipedia.org/wiki/Comparison_of_C_Sharp_and_Java>