НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_Кафедра обчислювальної техніки\_\_\_\_\_\_\_\_\_\_\_\_

(повна назва кафедри, циклової комісії)

КУРСОВА РОБОТА

з дисципліни «Паралельні та розподілені обчислення»

(назва дисципліни)

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

Студента (ки) 3 курсу ІО-21 групи

напряму підготовки 050102 «Комп’ютерна інженерія»

Кузьменка В. З.

(прізвище та ініціали)

Керівник доцент Корочкін О.В.

Національна оцінка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_\_\_

Оцінка: ECTS \_\_\_\_\_\_\_\_\_\_\_

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

Київ- 2015 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)*

*З А В Д А Н Н Я*

НА КУРСОВУ РОБОТУ СТУДЕНТУ

Кузьменко Володимир Зіновійович

*(прізвище, ім’я, по батькові)*

1. Тема роботи «Розробка програмного забезпечення для паралельних

комп’ютерних систем»

керівник роботи Корочкін Олександр Володимирович к.т.н., доцент

( прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

2. Строк подання студентом роботи 18 травня 2015 р.

3. Вхідні дані до роботи

* огляд шести ядерних процесорів фірми AMD
* математична задача A = B∙(MO∙MK)∙α + min(Z)∙E∙MR
* структури паралельної комп’ютерної системи з спільною пам’яттю (ПКС СП) та паралельної комп’ютерної системи з локальною пам’яттю (ПКС ЛП);
* бібліотека програмування: OpenMP, Open MPI.

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

* огляд шести ядерних процесорів фірми AMD
* розробка і тестування програми ПРГ1 для ПКС ОП



Рисунок 1.1 – Структура ПКС СП

* розробка і тестування програми ПРГ2 для ПКС ЛП



Рисунок 1.2 – Структура ПКС ЛП

5. Перелік графічного матеріалу

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

7. Дата видачі завдання \_\_\_\_\_\_02.02.2015\_\_\_\_\_\_\_

*КАЛЕНДАРНИЙ ПЛАН*

|  |  |  |
| --- | --- | --- |
| №  з/п | Назва етапів виконання КР | Строк виконання етапів КР |
| 1 | Виконання Розділу 1 | 23.02.2015 |
| 2 | Виконання Розділу 2 | 23.03.2015 |
| 3 | Виконання Розділу 3 | 23.04.2015 |
| 4 | Тестування програм ПРГ1 та ПРГ2 | 10.05.2015 |
| 7 | Оформлення КР | 17.05.2015 |
| 8 | Захист КР | 18.05.2015 |

Студент \_\_\_\_\_\_\_\_\_ Кузьменко В. З.

( підпис ) (прізвище та ініціали)

Керівник роботи \_\_\_\_\_\_\_\_\_\_\_\_\_ Корочкін О.В.

( підпис ) (прізвище та ініціали)

ЗМІСТ

ВСТУП…………………………………………………………………………… 5

РОЗДІЛ 1. ОГЛЯД ШЕСТИ ЯДЕРНИХ ПРОЦЕСОРІВ КОМПАНІЇ AM….... 6

* 1. Історія розвитку багатоядерних процесорів компанії AMD …...…. 6
  2. Огляд шестиядерних процесорів AMD …………………….……. 11
  3. Висновки до розділу 1…………………………………….………… 24

РОЗДІЛ 2. РОЗРОБКА ПРОГРАМИ ПРГ1 ДЛЯ ПКС ОП…………………25

* 1. Розробка паралельного математичного алгоритму…………………25
  2. Розробка алгоритмів процесів……………………………………… 27
  3. Розробка схеми взаємодії процесів…………………………………28
  4. Розробка програми ПРГ1…………………………………………… 29
  5. Тестування програми ПРГ1 …………………………………………30
  6. Висновки до розділу 2 ………………………………………………..31

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

* 1. Розробка паралельного математичного алгоритму………………
  2. Розробка алгоритмів процесів……………………………………
  3. Розробка схеми взаємодії процесів………………………… …
  4. Розробка програми ПРГ2………………………………………
  5. Тестування програми ПРГ2………………………………………
  6. Висновки до розділу 3………………………………………………

ОСНОВНІ РЕЗУЛЬТАТИ І ВИСНОВКИ ДО РОБОТИ………………………

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ…………………………………….

ДОДАТКИ…………………………………………………….............................

ВСТУП

Для розв’язання багатьох задач необхідна висока продуктивність та висока швидкість передачі інформації по каналах зв’язку, великі об’єми оперативної і постійної пам’яті, які не можуть забезпечити типові обчислювальні засоби. Одним з шляхів забезпечення таких вимог є організація паралельних та розподілених обчислень і відповідних технічних засобів їх реалізації.

В першому розділі «Огляд шестиядерних процесорів компанії AMD», загальна характеристика процесорів AMD, їх архітектура, та наведені специфікації шести ядерних процесорів.

Другий та третій розділи присвячені розробці програми для обчислення математичної задачі в паралельній комп’ютерній системі зі спільною та локальною пам’яттю відповідно.

Для розробки програми обчислення математичної задачі в паралельній комп’ютерній системі зі спільною пам’яттю використана бібліотека OpenMP.

Лістинги та алгоритми розроблених програм наведено у додатках.

**РОЗДІЛ 1. ОГЛЯД ШЕСТИТИЯДЕРНИХ ПРОЦЕСОРІВ КОМПАНІЇ AMD**

Advanced Micro Devices, Inc. (AMD) — компанія виробник інтегрованої електроніки [2]. Це другий найбільший постачальник x86сумісних процесорів і великий постачальник флеш-пам'яті. Через жорстку цінову конкуренцію з багаторічним суперником Intel фінансові показники не мають стабільності: благополучні періоди чергуються з періодами збитків. При цьому AMD, не економлячи, фінансує свої наукові дослідження (до 20% від обсягу продажу) та розширює свої виробничі потужності. У AMD одні з найсучасніших у галузі виробничі потужності, розташовані в США, Південно-Східній Азії. Що стосується азіатських виробництв AMD, то вони створені в Японії в рамках спільного з Fujitsu підприємства з виробництва модулів флеш-пам'яті на основі 0,35-мікронних технологій. Крім того, AMD має складальні і тестові майданчики у Сінгапурі і Таїланді. Стратегічними партнерами компанії AMD у виробництві персональних комп'ютерів є такі загальновідомі компанії, як Acer, Fujitsu/ICL, Hewlett-Packard і IBM. Їх супроводжують 3Com, Bay Networks, Cabletron і Cisco у мережевих продуктах і Alcatel, AT&T, Ericsson, NEC,Siemens і Sony на ринку телекомунікаційних систем.

**1.1. Історія розвитку багатоядерних процесорів компанії AMD**

**Процесор Athlon 64 X2**

Першим двохядерним мікропроцесором компанії AMD для персональних комп'ютерів став Athlon 64 X2. Цей процесор містить два ядра Athlon 64, розміщених на одному кристалі. Ядра мають у своєму розпорядженні загальний двоканальний контролер пам'яті, північний міст і додаткову логіку управління.

Початкові версії засновані на Athlon 64 степінга E і, залежно від моделі, мають 512 або 1024 КБ кешу другого рівня, а також 128 КБ кешу першого рівня на кожне ядро. Техпроцес виробництва зберігся з попередніх версій і становить 65 нм.

Athlon 64 X2 підтримує набір інструкцій SSE3 (які раніше підтримувалися тільки процесорами компанії Intel), що дозволило запускати з максимальною продуктивністю код, оптимізований для процесорів Intel.[1] Ці поліпшення не унікальні для Athlon 64 X2 і також є в релізах процесорів Athlon 64, побудованих на ядрах Venice, San Diego і Newark.

Процесори Phenom

У 2007 році компанія випустила серію процесорів Phenom для персональних комп'ютерів. AMD Phenom — багатоядерний центральний процесор від компанії AMD. Має: два, три або чотири ядра (Phenom X2, Phenom X3, Phenom X4 відповідно).[2] Базується на архітектурі К10. Трьохядерні версії (кодова назва Toliman) Phenom відносяться до серії 8000, а чотирьохядерні (кодова назва Agena) до AMD Phenom X4 9000. В цій серії добавлено спільний кеш третього рівня, який дозволяє швидко обмінюватися інформацією між ядрами.

**Процесори Phenom** II

В 2010 році було представлено серію процесорів Phenom II, які виготовлені за 45 нм технологічним процесом, і базуються на AMD K10 мікроархітектурі, в свою чергу є наступниками процесорної лінійки Phenom.

Головною особливістю нової архітектури є розширення модельного ряду новими шестиядерними процесорами. У процесорах Phenom II збільшили втричі загальний розмір кеш-пам'яті L3 від 2 МБ (як в лінії Phenom) до 6 МБ, завдяки цьому приріст продуктивності збільшився до 30%. Ще одна відмінність від попередніх Phenom це те, що технологія Cool'n'Quiet тепер застосовується для процесора в цілому, а не для кожного окремого ядра. Це було зроблено з метою уникнення неправильної обробки потокових обчислень у Windows Vista, яка могла переводити однопотокові обрахунки на інші ядра, які працювали в режимі холостого ходу на частоті, зменшеній в два рази, в результаті продуктивність процесу також зменшувалася вдвічі.

Нові шестиядерні та чотирьохядерні процесори AMD Phenom II сумісні з материнськими платами з Socket AM3 і AM2+, хоча доведеться оновити BIOS.

**Архітектура Bulldozer**

В 2011 році AMD перейшла на нову архітектуру з кодовою назвою Bulldozer. Процесори покоління AMD K11 виготовляються за 32-нм технологією і призначені для серверів і високопродуктивних ПК.

Процесори Bulldozer, за запевненням представників AMD, мають повністю нову архітектуру в порівнянні з попередніми поколіннями AMD K8 і AMD K10.

Тому компанія разом з процесорами презентувала новий сокет та чіпсет. Нові процесори сумісні з материнськими платами із Socket AM3+.

Основною особливістю цієї архітектури є те, що вона модульна. Модуль складається з двох фізичних ядер зі спільною кеш-пам’яттю другого рівня. У нових процесорах для персональних комп’ютерів може бути 2, 3, або 4 модуля, тобто 4, 6, або 8 ядер відповідно. З цього зрозуміло, що AMD випустила перший 8-ядерний процесор для ринку ПК, оскільки в процесорів Intel максимальна кількість ядер рівна шести. Процесори Bulldozer вперше підтримують нові інструкції x86 (SSE4.1, SSE4.2, CVT16, AVX і XOP, в тому числі 4-операндний модуль FMAC). Кожне ядро має 128-бітний модуль FPU з підтримкою FMA, при чому ці модулі можуть об’єднуватися в один загальний 256-бітний FPU між двома ядрами. Цю конструкцію супроводжують два модуля обчислень над цілими числами (по одному на кожне ядро) з 4-ма лініями зв’язку і можливістю спільної вибірки та декодування. Таким чином, один модуль з двома ядрами еквівалентний двохядерному процесору при операціях з цілими числами та одноядерному процесору при роботі над числами з плаваючою крапкою. Також кожен модуль має 2 МБ кешу другого рівня.

А кеш третього рівня загальний для всіх модулів. Його розмір залежить від кількості модулів і може займати 4, 6, або 8 МБ.

У новій архітектурі введена підтримка нової версії технології AMD Direct Connect і чотирьох каналів HyperTransport 3.1 на кожен процесор. Можливість роботи з пам'яттю DDR3 і технологією розширення пам'яті AMD G3MX дозволить збільшити пропускну здатність процесора. Також покращено керування живленням.

Нові процесори отримають підтримку технології Turbo Core 2, яка дозволяє збільшити номінальну частоту всього процесора на 500МГц, або половину ядер на 1ГГц, і помітно підвищити продуктивність аналогічно технології Intel Turbo Boost.

**Архітектура Piledriver**

В 2012 році компанія переходить на нову мікроархітектуру Piledriver. Piledriver – мікроархітектура , розроблена AMD в якості наступника Bulldozer. Piledriver використовує той же модульний дизайн. Одними з головних відмінностей в оновленій модульній архітектурі є поліпшені модуль передбачення розгалужень і планування використання модулів цілих чисел і чисел з плаваючою крапкою, поряд з переходом на новий тип тригерів з поліпшеними показниками енергоспоживання . На практиці це призвело до зросту частоти на 8-10 % та збільшення продуктивності приблизно на 15 % з аналогічним енергоспоживанням.

Спочатку в продажу вийшли процесори типу AMD Accelerated Processing Unit (APU) з кодовою назвою Trinity та серія мобільних продуктів.Пізніше також розповсюдилися процесори FX-серії. Відмінністю APU від серії FX є наявність вбудованого графічного ядра.

Його потужності вистачає для роботи з мультимедіа, а розташування центрального і графічного процесорів на одному кристалі дешевше, ніж два елементи окремо. Графічний процесор займає близько половини площі кристалу, що значно більше, ніж в аналогів від Intel. У ГП відсутня внутрішня пам’ять, тому він резервує частину оперативної пам’яті для власних потреб. Через відведення великої частини площі кристалу на ГП, довелося змінити архітектурі центрального процесору, а саме: відмовитися від використання кешу третього рівня ( в серії FX розподіл кешу залишився без змін).

З появою нової архітектури компанія AMD також націлилася на ринок мобільних комп’ютерів в низькому та середньому ціновому діапазоні. Тому були презентовані одно- та двомодульні APU з малим TDP (17-35 Вт).

Для встановлення процесорів FX-серії використовуються старий сокет AM3+, але для серії APU презентовано новий Socket FM2.

Архітектура Steamroller

На початку 2014 року компанія анонсувала нову архітектуру Steamroller. В Streamroller відсутня FX-серія, але APU продовжують використовувати модульні структуру, як і їх попередники, одночасно спрямовані на досягнення вищого рівня паралелізму.

Відмінністю нових модулів є розділені декодери інструкцій для кожного ядра в модулі. Також на 25% збільшено шину розсилки на ядро, оновлені планувальники інструкцій, більші і гнучкіші кеші (кеш другого рівня може динамічно змінювати розмір), додана черга мікрооперацій, на кристалі розміщено більше регістрових ресурсів, покращений контролер пам’яті.

Згідно з оцінками з AMD, ці поліпшення збільшать кількість виконуваних інструкцій за такт до 30 % в порівнянні з ядром першого покоління Bulldozer при збереженні високих тактових частот Piledriver, але із зменшеним енергоспоживання. В середньому нові процесори швидші за попередників на 9 % в однопоточних програмах та на 18% в багатопоточних.

Архітектура Excavator

В 2015 році запланований перехід на архітектуру Excavator. Зараз ця мікроархітектура на стадії розробки. Excavator APU будуть випускатися під кодовою назвою Carrizo. Очікується, підтримка нових інструкцій, таких як AVX2, BMI2 і RDRAND. Також очікується оновлення контролерів пам’яті для підтримки пам’яті стандартів DDR3 і DDR4.

**1.2. Огляд шестиядерних процесорів AMD**

**Огляд шестиядерного процесора AMD Phenom II X6 1055T**

Таблиця 1.1 Специфікація AMD Phenom II X6 1055T:

|  |  |
| --- | --- |
| Модель | AMD Phenom II X6 1055T |
| Маркування | HDT55TFBK6DGR |
| Процесорний роз’єм | Socket AM3, AM2+ |
| Тактова частота, МГц | 2800 |
| Множник | 14 |
| Частота шини HT, МГц | 2000 |
| Об'єм кеш-пам'яті L1, КБ | 128 x 6 |
| Об'єм кеш-пам'яті L2, КБ | 512 х 6 |
| Об'єм кеш-пам'яті L3, КБ | 6144 |
| Ядро | Thuban |
| Кількість ядер | 6 |
| Напруга живлення, В | 1,125 – 1,40 |
| Тепловий пакет, Вт | 125 |
| Тактова частота в режимі AMD Turbo Core, МГц | до 3300 |
| Критична температура, °C | 62 |
| Техпроцес, нм | 45 |
| Підтримка технологій | AMD Turbo Core Cool’n’Quiet 3.0  Coolcore Technology  Dual Dynamic Power Management  Enhanced Virus Protection  Virtualization Technology  Core C1 and C1E states  Package S0, S1, S3, S4 and S5 states |
| Вбудований контролер пам'яті | |
| Типи пам'яті | DDR2-800/1066  DDR3-800/1066/1333/1600 |
| Число каналів пам'яті | 2 |
| Максимальний об'єм пам'яті, ГБ | 16 |
| Максимальна пропускна здатність, ГБ/c | 21,3 |
| Підтримка ECC | немає |

AMD Phenom II X6 на ядрі Theban від AMD Phenom II X4 на ядрі Deneb, таких як поява ще двох обчислювальних ядер і реалізація технології Turbo Core, у вбудованого контролера пам'яті новинки офіційно з'явилася підтримка DDR3-1600. Використання більш швидшої оперативної пам'яті повинно трохи зменшити можливі затримки внаслідок збільшення кількості виконавчих блоків без розширення кеш-пам'яті третього рівня.

Маркування HDT55TFBK6DGR, яке можна розшифрувати так:

* HD – процесор AMD архітектури K10,5 для робочих станцій;
* T – процесор з фіксованим множником;
* 55T – модельним номер, що ідентифікує сам процесор та вказує на підтримку технології Turbo Core;
* FB – тепловий пакет процесора до 125 Вт при напрузі живлення до 1,4 В;
* K – впакований процесор у корпус 938 pin OµPGA (Socket AM3);
* 6 – загальна кількість активних ядер і відповідно об'єм кеш-пам'яті L2 6x512 КБ;
* DGR - ядро Thuban степпінгу E0.

Розподіл кеш-пам'яті процесора AMD Phenom II X6 залишився точно таким же, як і для інших двох-, трьох- і чотирьохядерних моделей з повним об'ємом кеш-пам'яті третього рівня. Так,  AMD Phenom II X6 1055T має у своєму розпорядженні 128 КБ кеш-пам'яті першого рівня з дволінійною асоціативністю окремо для даних і інструкцій на кожне ядро, 512 КБ кеш-пам'яті другого рівня з шістнадцятилінійною асоціативністю також на кожне ядро та загальні 6 МБ кеш-пам'яті третього рівня з 48 ліній асоціативності[5].

Огляд шестиядерного процесора AMD Phenom II X6 1075T

Подібно іншим процесорам модель AMD Phenom II X6 1075T підтримує технологію Turbo Core. Для того щоб дізнатися багато інформації про можливості процесора, достатньо лише глянути на його маркування на теплорозподільній кришці[9]. Маркування HDT75TFBK6DGR означає наступне:

* HD – процесор сімейства AMD Phenom архітектури K10,5 для робочих станцій;
* T75T – модельним номер, що ідентифікує сам процесор та вказує на підтримку технології Turbo Core (процесор з заблокованим множником);
* FB – тепловий пакет процесора 125 Вт;
* K – процесор упакований в корпус 938 pin OµPGA (Socket AM3);
* 6 – загальна кількість активних ядер;
* D – об'єм кеш-пам'яті L2 6x512 КБ і L3 6 МБ;
* GR – ядро степпінгу PH-E0.

Що ж стосується тильної сторони процесора, то перед нами з'являється знайоме 938-контактне пакування для роз’єму Socket AM3, який зворотно сумісний з Socket AM2+.

Таблиця 1.2 Специфікація AMD Phenom II X6 1075T

|  |  |
| --- | --- |
| Модель | AMD Phenom II X6 1075Т |
| Маркування | HDT75TFBK6DGR |
| Процесорний роз’єм | Socket AM3 (AM2+) |
| Тактова частота, МГц | 3000 |
| Тактова частота в режимі Turbo Core (спрацьовує для 3-х і менше ядер), МГц | 3500 |
| Множник | 15 |
| Частота шини HT, МГц | 2000 |
| Об'єм кеш-пам'яті L1, КБ | 128 x 6 |
| Об'єм кеш-пам'яті L2, КБ | 512 х 6 |
| Об'єм кеш-пам'яті L3, КБ | 6144 |
| Ядро | Thuban |
| Кількість ядер | 6 |
| Напруга живлення, В | 1,15 – 1,475 (базова частота)  1,25 – 1,475 (Turbo Core mode)  1,00 – 1,225 (простій) |
| Напруга північного мосту, В | 1,05 – 1,175 |
| Тепловий пакет, Вт | 95 |
| Критична температура, °C | 55 – 62 |
| Техпроцес, нм | 45 |
| Підтримка технологій | Cool’n’Quiet 3.0 CoolCore Technology Dual Dynamic Power Management Enhanced Virus Protection Virtualization Technology Core C1 and C1E states Package S0, S1, S3, S4 and S5 states |
| Вбудований контролер пам'ятей | |
| Типи пам'яті | DDR2-667/800/1066  DDR3-800/1066/1333 |
| Число каналів пам'яті | 2 |
| Максимальний об'єм пам'ятей, ГБ | 16 |
| Максимальна пропускна здатність, ГБ/c | 21,3 |
| Підтримка ECC | є |

Принципової відмінності моделі процесора AMD Phenom II X6 1075T від своїх «побратимів» практично ніякої немає, за винятком того, що множник заблокований, на відміну від старших моделей. Сам же множник зафіксований на 15, що, власне, і забезпечує частоту процесора 3,0 ГГц[4].

Огляд шестиядерного процесора AMD Phenom II X6 1100T

Процесор AMD Phenom II X6 1100T – вершина лінійки шестиядерних процесорів AMD на ядрі Thuban. Даний процесор має три характерні риси: це найвища в лінійці шестиядерних процесорів AMD номінальна тактова частота, та статус самого потужного продукту.

На теплорозподільну кришку процесора нанесене спеціальне маркування – HDE00ZFBK6DGR, яка розшифровується таким чином:

* HD – процесор сімейства AMD Phenom архітектури K10,5 для робочих станцій;
* E00 – модельним номер, що ідентифікує сам процесор та вказує підтримку технології Turbo Core;
* Z – процесор з вільним множником;
* FB – тепловий пакет процесора 125 Вт;
* K –процесор упаковано в корпус 938 pin OµPGA (Socket AM3);
* 6 – загальна кількість активних ядер;
* D – об'єм кеш-пам'яті L2 6x512 КБ і L3 6 МБ;
* GR – ядро Thuban степпінга E0.

Таблиця 1.3 Специфікація AMD Phenom II X6 1100T:

|  |  |
| --- | --- |
| Модель | AMD Phenom II X6 1100T |
| Маркування | HDE00ZFBK6DGR |
| Процесорний роз’єм | Socket AM3, AM2+ |
| Тактова частота, МГц | 3300 |
| Множник | 16,5 |
| Частота шини HT, МГц | 2000 |
| Об'єм кеш-пам'яті L1, КБ | 128 x 6 |
| Об'єм кеш-пам'яті L2, КБ | 512 х 6 |
| Об'єм кеш-пам'яті L3, КБ | 6144 |
| Ядро | Thuban |
| Кількість ядер | 6 |
| Напруга живлення, В | 1,470 |
| Тепловий пакет, Вт | 125 |
| Тактова частота в режимі AMD Turbo Core, МГц | до 3700 |
| Критична температура, °C | 62 |
| Техпроцес, нм | 45 |
| Підтримка технологій | AMD Turbo Core Cool’n’Quiet 3.0 CoolCore Technology Dual Dynamic Power Management Enhanced Virus Protection Virtualization Technology Core C1 and C1E states Package S0, S1, S3, S4 and S5 states |
| Вбудований контролер пам'яті | |
| Типи пам'яті | DDR2-800/1066  DDR3-800/1066/1333/1600 |
| Кількість каналів пам'яті | 2 |
| Максимальний об'єм пам'яті, ГБ | 16 |
| Максимальна пропускна здатність, ГБ/c | 21,3 |
| Підтримка ECC | є |

Шестиядерний процесор AMD відрізняється від моделі на щабель нижче - AMD Phenom II X6 1090T тільки збільшеним на половину одиниці множником, отже, збільшилася на 100 МГц тактова частота в номінальному режимі та режимі AMD Turbo Core.

При встановленні швидких модулів пам'яті DDR3-2000 1024 МБ Kingston Hyperx KHX16000D3T1K3 система автоматично розпізнає їх тільки як DDR3-1333, для використання пам'яті в більш швидшому режимі, ніж DDR3-1333, необхідно проводити її налаштування в BIOS.

Щоб оцінити обсяги енергоспоживання ми замірили потреби в енергії для всього тестового стенда, створюючи навантаження на процесор за допомогою програмі EVEREST 5.0. Для порівняння наведені результати аналогічних вимірів на деяких інших процесорах.

Таблиця 1.4 Обсяги енергоспоживання

|  |  |  |
| --- | --- | --- |
| Процесор | Режим простою, Вт | EVEREST 5.0, Вт |
| AMD Phenom II X6 1075T | 75 | 232 |
| AMD Phenom II X6 1100T | 74 | 251 |

Згідно наведеній таблиці, процесор з шістьма ядрами споживає найбільшу кількість енергії.

Процесор AMD Phenom II X6 1100T вінчає вершину лінійки шестиядерних процесорів на ядрі Thuban. У плані функціональних можливостей даний процесор не відрізняється від молодших моделей і цікавий своїми характеристиками тільки при порівнянні з молодшими шести ядерними процесорами, що працюють на номінальних частотах[6].

Огляд шестиядерного процесора AMD FX-6100 на базі архітектури Bulldozer

Традиційне маркування на процесорній кришки повідомляє власнику досить багато інформації. У цьому випадку вона наступна – FD6100WMW6KGU:

* F – процесор належить до сімейства AMD FX;
* D – сфера застосування даного процесора – робочі станції;
* 6100 – модельним номер;
* WM – тепловий пакет процесора 95 W;
* W – упакований процесор у корпус 938 pin Socket AM3+;
* 6 – загальна кількість активних ядер;
* K – об'єм кеш-пам'яті L2 2 МБ на кожний модуль і 8 МБ кеш-пам'яті L3;
* GU - ядро процесора степпінга OR-B2.

На звороті є знайомі нам 938 контактів, однак у цьому випадку роз’єм для встановлення CPU використовує тільки Socket AM3+, що забезпечує підтримку лише  DDR3.

Кеш-пам'ять новинки розподіляється таким чином. Кеш-пам'ять 1 рівня: по 16 КБ на кожне з 6 ядер виділяється для даних з чотирма каналами асоціативності, при цьому для інструкції є 64 КБна кожний 2-процесорний модуль з 2 каналами асоціативності. Кеш-пам'ять 2 рівня: по 2 МБ на кожний модуль процесора, яких як ви пам'ятаєте 3, з 16 каналами асоціативності. Кеш-пам'ять 3 рівня загальна для всього процесора і становить 8 МБ з 64 каналами асоціативності. Процесори даного сімейства оснащені технологією Turbo Core 2.0, яка дозволяє підвищувати частоту процесора при вирішенні ресурсномістких завдань[8].

Таблиця 1.5 Специфікація AMD FX-6100

|  |  |
| --- | --- |
| Модель | AMD FX-6100 |
| Маркування | FD6100WMW6KGU |
| Процесорний роз’єм | Socket AM3+ |
| Тактова частота (номінальна), МГц | 3300 |
| Максимальна тактова частота з TC 2.0), МГц - для 6 ядер - для 3 ядер | 3600 3900 |
| Множник (номінал) | 16,5 |
| Частота шини HT, МГц | 2200 |
| Об'єм кеш-пам'яті L1, КБ | 3 x 64 (інструкції) 6 x 16 (дані) |
| Об'єм кеш-пам'яті L2, МБ | 3 х 2 |
| Об'єм кеш-пам'яті L3, МБ | 8 |
| Ядро | Zambezi |
| Кількість ядер | 6 |
| Напруга живлення, В | --- |
| Тепловий пакет, Вт | 95 |
| Критична температура, °C | 70 |
| Техпроцес, нм | 32 |
| Підтримка технологій | Multiple low-power states Enhanced Virus Protection Advanced Power Management Virtualization Technology Hardware Thermal Control Core C0, C1, C1E, C6, CC6, states Package S0, S1, S3, S4 and S5 states AMD Turbo CORE technology 2.0 |
| Вбудований контролер пам'яті | |
| Типи пам'яті | DDR3-1066/1333/1600/1866 |
| Кількість каналів пам'яті | 2 |
| Максимальний об'єм пам'яті, ГБ | 16 |
| Максимальна пропускна здатність, ГБ/c | 21,3 |
| Підтримка ECC | є |

Таблиця 1.6 Аналіз енергоефективності

|  |  |  |
| --- | --- | --- |
| Найменування | У простої, Вт | EVEREST 5.0, Вт |
| AMD FX-6100 | 72 | 144 |
| AMD Phenom II X6 1100T | 74 | 251 |
| AMD Phenom II X6 1050T | 74 | 168 |
| AMD Phenom II X6 1075T | 75 | 249 |

Огляд шестиядерного процесора FX-6200

Модель AMD FX-6200, анонсована наприкінці лютого 2012 року разом з AMD FX-4170. Даний ЦП виготовлений по 32 нм техпроцесу і складається з трьох обчислювальних модулів, кожний з яких містить по два ядра, що разом дає нам шість ядер[7]. Процесор також відрізняється підвищеною до 3,8 ГГц, але термопакет залишився без змін - 125 Вт.

Корпус процесора такий же, як і у інших представників сімейства AMD FX. Та і від більш старих Phenom II для Socket AM3 мало чим відрізняється. Масивна металева кришка захищає кремнієву основу процесора від пошкоджень при встановленні системи охолодження і одночасно забезпечує рівномірний розподіл тепла, запобігаючи перегріву окремих ділянок кристала. Маркування на кришці дає користувачу максимум інформації про процесор[8]. Для цієї моделі вона наступна - FD6200FRW6KGU:

* F – процесор належить до сімейства AMD FX;
* D – сфера застосування даного процесора – робочі станції;
* 6200 – модельним номер;
* FR – тепловий пакет процесора 125 W;
* W – впакований процесор у корпус 938-pin Socket AM3+;
* 6 – загальна кількість активних ядер;
* K – об’єм кеш-пам'яті L2 2 МБ на кожний модуль і 8 МБ кеш-пам'яті L3;
* GU - ядро процесора степінга OR-B2

Дана модель є найпотужнішим шестиядерним процесором на архітектурі Bulldozer. Детально ознайомитися із її характеристиками можна в таблиці 1.7.

Таблиця 1.7 Специфікація AMD FX-6200

|  |  |
| --- | --- |
| Модель | AMD FX-6200 |
| Маркування | FD6100WMW6KGU |
| Процесорний роз’єм | Socket AM3+ |
| Тактова частота (номінальна),  МГц | 3800 |
| Максимальна тактова частота з  TC 2.0, МГц (для всіх ядер) | 4100 |
| Множник (номінал) | 16,5 |
| Частота шини HT, МГц | 2200 |
| Об'єм кеш-пам'яті L1, КБ | 3 x 64 (інструкції)  6 x 16 (дані) |
| Об'єм кеш-пам'яті L2, МБ | 3 х 2 |
| Об'єм кеш-пам'яті L3, МБ | 8 |
| Ядро | Zambezi |
| Кількість ядер | 6 |
| Напруга живлення, В | 0,75-1,40 |
| Тепловий пакет, Вт | 125 |
| Критична температура, °C | 75 |
| Техпроцес, нм | 32 |
| Підтримка технологій | Multiple low-power states  Enhanced Virus Protection  Advanced Power Management  Virtualization Technology  Hardware Thermal Control  Core C0, C1, C1E, C6, CC6, states  Package S0, S1, S3, S4 and S5 states  AMD Turbo CORE technology 2.0 |
| Вбудований контролер пам'яті | |
| Типи пам'яті | DDR3-1066/1333/1600/1866 |
| Кількість каналів пам'яті | 2 |
| Максимальний об'єм пам'яті, ГБ | 16 |
| Максимальна пропускна здатність, ГБ/c | 21,3 |
| Підтримка ECC | є |

Огляд шестиядерного процесора FX-6350

AMD FX-6350 є представником високопродуктивної лінійки компанії AMD. Він не обладнаний графічним прискорювачем, але при цьому має досить високу швидкодію та шість обчислювальних ядер. У лінійці AMD FX є два шестиядерні процесори, які основані на мікроархітектурі AMD Piledriver і входять у сімейство AMD Vishera.

Процесор AMD FX-6350 має класичний корпус, який за формою не відрізняється від інших моделей компанії AMD. Що стосується маркування CPU, то при його розшифруванні можна скласти досить детальну картину про даний пристрій. У нашому випадку FD6350FRW6KHK має таке позначення:

* F – сімейство AMD FX;
* D – сегмент робочих станцій;
* 6350 – номер моделі;
* FR – тепловий пакет 125 Вт;
* W – процесорний роз’єм Socket AM3+;
* 6 – загальна кількість активних ядер;
* K – обсяг кеш-пам'яті L2 – 2 МБ на кожний двоядерний модуль і 8 МБ загальної кеш-пам'яті L3;
* HK - ядро процесора степінга OR-C0.

Дана модель є найпотужнішим шестиядерним процесором на архітектурі Bulldozer[3]. Відмінністю від специфікацій FX-6200 є змінена частота, додаткові технології та відсутність критичної температури. Детально ознайомитися із її характеристиками можна в таблиці 1.8.

Таблиця 1.8 СпецифікаціяAMD FX-6350

|  |  |
| --- | --- |
| Модель | AMD FX-6350 |
| Маркування | FD6100WMW6KGU |
| Процесорний роз’єм | Socket AM3+ |
| Тактова частота (номінальна),  МГц | 3900 |
| Максимальна тактова частота з  TC 2.0, МГц (для всіх ядер) | 4200 |
| Множник (номінал) | 16,5 |
| Частота шини HT, МГц | 2200 |
| Об'єм кеш-пам'яті L1, КБ | 3 x 64 (інструкції)  6 x 16 (дані) |
| Об'єм кеш-пам'яті L2, МБ | 3 х 2 |
| Об'єм кеш-пам'яті L3, МБ | 8 |
| Ядро | Zambezi |
| Кількість ядер | 6 |
| Напруга живлення, В | 0,888-1,404 |
| Тепловий пакет, Вт | 125 |
| Критична температура, °C | - |
| Техпроцес, нм | 32 |
| Підтримка технологій | Multiple low-power states  Enhanced Virus Protection  Advanced Power Management  Virtualization Technology  Hardware Thermal Control  Core C0, C1, C1E, C6, CC6, states  Package S0, S1, S3, S4 and S5 states  AMD Turbo CORE technology 2.0 |
| Вбудований контролер пам'яті | |
| Типи пам'яті | DDR3-1066/1333/1600/1866 |
| Кількість каналів пам'яті | 2 |
| Максимальний об'єм пам'яті, ГБ | 16 |
| Максимальна пропускна здатність, ГБ/c | 21,3 |
| Підтримка ECC | є |

1.3 Висновки до розділу 1

1. Копанія AMD за останні дев’ять років випустила 6 поколінь багатоядерних процесорів. Починаючи з двохядерних і закінчуючи восьмиядерними процесорами для ринку персональних комп’ютерів.
2. З точки зору архітектури ці покоління можна поділити на 2 частини. В першу входять Athlon 64 X2, Phenom та Phenom II. В другу – Bulldozer, Piledriver, Srteamroller. Останні відрізняються від перших наявністю модульної архітектури. Тобто, процесор поділяється не на ядра, а на модулі, кожен з яких містить по два ядра.
3. Процесори з архітектурами Phenom та Phenom II відрізняються від Athlon 64 X2 тим, що у них є кеш-пам’ять третього рівня. Ця пам’ять спільна для всіх ядер і дозволяє швидко обмінюватися даними.
4. Процесори з архітектурами Piledriver і Srteamroller, на відміну від попередніх поколінь, мають серії процесорів з вбудованим графічним прискорювачем. Огляд шестиядерних процесорів показав, що з кожним поколінням зростає частота процесора і на архітектурі Srteamroller перетнула межу в 4ГГц. В

**РОЗДІЛ 2. РОЗРОБКА ПРОГРАМИ ПРГ 1 ДЛЯ ПКС СП**

У даному розділі розроблюється програма ПРГ1 для системи зі спільною пам’яттю, що відповідає технічному завданню, представленому на малюнку нижче.



**2.1 Розробка паралельного математичного алгоритму.**

Паралельний математичний алгоритм відповідно до рекомендованої методичної літератури [10] можна подати у вигляді наступних двох етапів:

1. *minZi = min(ZH), i = ;*

*minZ = min(minZ, minZi), i = ;*

1. *MAH = MO∙MKH;*
2. *AH = α∙B∙MAH + minZ∙E∙MRH,*

де:

* H = N/P;
* AH – H рядків вектора А;
* MKH – H рядків матриці MK;
* MRH – H рядків матриці MR;
* MAH – H рядків матриці MA

Спільні ресурси: α, B, E, MO

**2.2 Розробка алгоритмів процесів**

Оскільки розроблюване програмне забезпечення є масштабованим і працює на системі із кількістю процесорів P≥2, то зручним варіантом реалізації є написання єдиного алгоритму для всіх задач.

|  |  |
| --- | --- |
| Задачі Т(0) – Т(P-1) | |
| Крок алгоритму | КД, Бар’єри |
| 1. Якщо tid = 0 Ввести α, B, MK, MR. |  |
| 2. Якщо tid = P-1 ввести Z, E, MO. |  |
| 3. **Бар’єр** для усіх задач. Синхронізація по вводу. | Бар’єр |
| 4. Обчислення *minZi = min(Z), i =* |  |
| 5. Обчислення *minZ = min(minZ, minZi), i =* | КД |
| 6. **Бар’єр** для усіх задач. Синхронізація обчислень minZ | Бар’єр |
| 7. **Копіювати** MOi=MO, *minZi = minZ, αi = α, Bi = B, Ei = E* | КД |
| 8. Обчислення MAH = MOi∙MKH |  |
| 9. **Бар’єр** для усіх задач. Синхронізація обчислень MA | Бар’єр |
| 10. Обчислення *AH = αi∙Bi∙MAH + minZi∙Ei∙MRH, i =* |  |
| 11. **Бар’єр** для усіх задач. Синхронізація обчислень A | Бар’єр |
| 12. Якщо tid = 0 вивести результат AH. | ­ |

**2.3 Розробка схеми взаємодії процесів**

На основі алгоритму для всіх задач, наведеному в попередньому розділі, було розроблена структурна схема взаємодії задач (рис. 2.1). Вона дозволяє наочно контролювати бар’єри та критичні ділянки. Крім того, на структурній схемі уводяться також замок, та критична секція, що будуть використовуватись в програмі.

Для демонстрації взаємодії між задачами вибрано три задачі: T(0), T(i), T(P-1) (i = ). Задачі T(0) і T(P-1) вводять дані, тому з ними взаємодіють всі інші (синхронізація по вводу). Задачі T(i) (i = ) виконують лише обчислення і синхронізуються зі всіма іншими задачами по обчисленню. Задача T(0) виводить результат обчислень.



Рис. 2.1 Структурна схема взаємодії задач

**2.4 Розробка програми ПРГ1**

Програма написана на мові C++ з використанням бібліотеки OpenMP та складається з трьох модулів: основного pro2\_course\_work\_prg1.cpp, та допоміжного operations.cpp і файлу заголовків operations.h.

Основний модуль pro2\_course\_work\_prg1.cpp містить одну функцію:

* main – точка входу в програму. Формує ідентифікатори tid задач, запускає задачі, а також вимірює час виконання elapsedTime програми ПРГ1.

В цьому модулі об’явлена така змінна: P – налаштування кількості процесорів.

Файл заголовків operations.h. представляє заголовки службових функцій (таких як ввід/вивід, копіювання), та типи вектора і матриці. У ньому також об’явлена змінна N що визначає розмірність матриць і векторів.

Допоміжний модуль operations.cpp реалізовує функції об’явлені у файлі заголовків operations.h.

Лістинг програми ПРГ1 наведено у додатку Г.

**2.5 Тестування ПРГ1**

Для тестування використовувалась паралельна обчислювальна система з наступними апаратними характеристиками:

* процесор (буде)
* оперативна пам’ять (буде)

В якості програмного забезпечення виступали:

* операційна система: (буде)
* середовище розробки і компіляції C++ програми: (буде)

Для вимірювання часу виконання програми використовувався високоточний таймер з бібліотеки OpenMP.

Коефіцієнт прискорення  показує скорочення часу виконання паралельної програми в паралельній системі з ** процесорами ** в порівнянні з часом виконання послідовної програми в однопроцесорній системі :



Коефіцієнт ефективності  застосування комп’ютерної системи показує ступінь використання ** процесорів системи:



Результати тестування і проведених досліджень ефективності розробленої програми наведено в таблицях Таблиця 2.1‑Таблиця 2.3.

Таблиця . – Час виконання програми ПРГ1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| N | Т1, мс | Т2, мс | Т3, мс | Т4, мс | Т5, мс | Т6, мс |
| 900 | 3606 | 1808 | 1328 | 1045 | 851 | 786 |
| 1800 | 29169 | 14467 | 10407 | 7952 | 6383 | 5530 |
| 2400 | 68163 | 34222 | 24637 | 18724 | 14979 | 12548 |

На основі даних із таблиці Таблиця 2.1 виконано розрахунок значень коефіцієнтів прискорення, які наведені в таблиці Таблиця 2.2.

Таблиця . – Коефіцієнти прискорення для програми ПРГ1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| N | Кількість процесорів (P) | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 900 | 1,00 | 1,99 | 2,72 | 3,45 | 4,24 | 4,59 |
| 1800 | 1,00 | 2,02 | 2,80 | 3,67 | 4,57 | 5,27 |
| 2400 | 1,00 | 1,99 | 2,77 | 3,64 | 4,55 | 5,43 |

Коефіцієнти ефективності (таблиця Таблиця 2.3) обчислено за даними таблиці Таблиця 2.2.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| N | Кількість процесорів (P) | | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 900 | 100,00% | 99,74% | 90,52% | 86,31% | 84,75% | 76,46% |
| 1800 | 100,00% | 100,81% | 93,43% | 91,70% | 91,40% | 87,92% |
| 2400 | 100,00% | 99,59% | 92,22% | 91,01% | 91,01% | 90,53% |

Таблиця . – Коефіцієнти ефективності для програми ПРГ1

Використовуючи таблиці Таблиця 2.2‑Таблиця 2.3 побудовано графіки зміни коефіцієнтів прискорення і ефективності в залежності від *N* і .

Рис. 2.2 – Графік зміни коефіцієнту прискорення програми ПРГ1 в залежності від кількості ядер при N=900

Рис. 2.3 – Графік зміни коефіцієнту прискорення програми ПРГ1 в залежності від кількості ядер при N=1800

Рис..4 – Графік зміни коефіцієнту прискорення програми ПРГ1 в залежності від кількості ядер при N=2400

Рис..5 – Графік зміни коефіцієнту ефективності програми ПРГ1 в залежності від кількості ядер при N=900

Рис.. – Графік зміни коефіцієнту ефективності програми ПРГ1 в залежності від кількості ядер при N=1800

Рис.. – Графік зміни коефіцієнту ефективності програми ПРГ1 в залежності від кількості ядер при N=2400

**2.6 Висновки до розділу 2**

Виконано розробку програми ПРГ1 для ПКС СП з використанням мови С++ і засобів синхронізації з бібліотеки OpenMP. Тестування програми показало наступне:

* тут будуть результати тестування…

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

У даному розділі розроблюється програма ПРГ1 для системи зі спільною пам’яттю, що відповідає технічному завданню, представленому на малюнку нижче.



Оскільки математична задача співпадає тою, що розглядалась в роз­ділі 2.1, то перший крок по рекомендованій методиці [25] не матиме відмінностей від уже виконаних. Тому розділ «Розробка паралельного математичного алгоритму» пропущений.

**3.1 Розробка алгоритмів процесів**

Для опису алгоритмів задач системи, розділимо всі задачі на 3 групи:

* задачі що належать до крайнього лівого стовпця (0, k);
* задачі що належать до крайнього правого стовпця (k-1, P-1);
* решта задач, що утворюють середні стовпці.

**Задача крайнього лівого стовпця (k=P/2)**

1. Якщо rank = 0, то ввести B, MK, MR, α.
2. Якщо rank = 0, то передати B, MK(k\*H), MR(k\*H), α. Задачі знизу.
3. Якщо rank = k, то прийняти B, MK(k\*H), MR(k\*H), α, від задачі зверху.
4. Передати B, MK(k\*H)-H, MR(k\*H)-H, α задачі справа.
5. Прийняти MO, E, ZH від задачі справа
6. Обчислити localMinZ = min(ZH)
7. Якщо rank = k, то передати localMinZ задачі зверху
8. Якщо rank = 0, то прийняти minZ від задачі знизу
9. Якщо rank = 0, то обчислити localMinZ = min(minZ, localMinZ)
10. Якщо rank = 0, то передати localMinZ задачі з права
11. Якщо rank = 0, то Отримати minZ від задачі з ліва.
12. Якщо rank = 0, то Передати minZ задачі з низу
13. Якщо rank = k, то отримати minZ від задачі зверху
14. Обчислити MAH = B(MO MKH)α + minZ∙E∙MRH
15. Отримати MAH\*k-H від задачі справа
16. Якщо rank = k, то передати MAH\*k задачі зверху
17. Якщо rank = 0, то Отримати MAH\*k від задачі знизу
18. Якщо rank = 0, то Вивести MA

**Задача крайнього правого стовпця**

1. Якщо rank = P/2-1, то ввести MO, Z, E.
2. Якщо rank = P/2-1, то передати MO, Z(k\*H), E задачі знизу.
3. Передати MO, Z(k\*H)-H, E задачі зліва.
4. Якщо rank = P-1, то прийняти MO, Z(k\*H), E від задачі зверху.
5. Прийняти B, MKH, MRH, α від задачі з ліва
6. Обчислити localMinZ = min (ZH)
7. Якщо rank = P-1, то передати localMinZ задачі зверху
8. Якщо rank = k-1, то прийняти minZ від задачі знизу
9. Якщо rank = k-1, то обчислити localMinZ = min(minZ, localMinZ)
10. Якщо rank = k-1, то передати localMinZ задачі з права
11. Якщо rank = k-1, то Отримати minZ від задачі з ліва.
12. Якщо rank = k-1, то Передати minZ задачі з низу
13. Якщо rank = P-1, то отримати minZ від задачі зверху
14. Обчислити MAH = B(MO MKH)α + minZ∙E∙MRH
15. Передати MAH\*k-H від задачі зліва

**Задача середніх стовпців**

1. Прийняти B, MK(MK.size())\*H MR, α від задачі зліва.
2. Передати B, MK(MK.size()-1)\*H, MR, α задачі справа.
3. Прийняти MO, Z(Z.size())\*H, E від задачі справа.
4. Передати MO, Z(Z.size()-1)\*H, E задачі зліва.
5. Якщо rank >= k, то передати localMinZ задачі зверху
6. Якщо rank < k, то прийняти minZ від задачі знизу
7. Обчислити minZ = (localMinZ, minZ)
8. Якщо rank = >(k-1)/2 отримати minZ від задачі з cправа
9. Якщо rank < (k-1)/2 отримати minZ від задачі зліва
10. Обчислити minZ = (localMinZ, minZ)
11. Якщо rank = >(k-1)/2, то передати minZ задачі зліва
12. Якщо rank < (k-1)/2, то передати minZ задачі справа
13. Якщо rank = (k-1)/2, то Отримати minZleft від задачі зліва
14. Якщо rank = (k-1)/2, то Отримати minZright від задачі справа
15. Якщо rank = (k-1)/2, то обчислити minZ = (minZ, minZleft, minZright)
16. Якщо rank = (k-1)/2, то передати minZ задачі зліва і справа
17. Якщо rank = >(k-1)/2 отримати minZ від задачі з зліва
18. Якщо rank < (k-1)/2 отримати minZ від задачі зліва
19. Копіювати localMinZ := minZ
20. Якщо rank = >(k-1)/2, то передати minZ задачі справа
21. Якщо rank < (k-1)/2, то передати minZ задачі зліва
22. Обчислити MAH = B(MO MKH)α + minZ∙E∙MRH
23. Отримати MA (MA.size()-1)\*H від задачі справа
24. Передати MA (MA.size())\*H задачі зліва

**3.2 Розробка схеми взаємодії процесів**

Схема взаємодії задач приведена на рисунку 3.1



**3.3 Розробка програми ПРГ2**

Програма написана на мові Java з використанням бібліотеки Open MPI та складається з 7 класів.

Класи Vector та Matrix інкапсулють поняття вектора та матриці відповідно, та реалізую методи що дозволяють змінювати стан даних абстракцій.

Клас MessageBox інкапсулює повідомлення яке відправляється у програмі. За допомогою методів, реалізованих у цьому класі можна створювати повідомлення яке місить одночасно декілька матриць, векторів чи чисел. Це повідомлення далі процесори надсилають одне одному для обміну данними.

Клас CalculateUtils містить набір статичних методів що інкапсулють математичні операції з векторами та матрицями. Також тут знаходяться методи для вводу та виводу данних.

Клас MessageKeys містить набір констант які використовуються для пересилки повідомлень між задачами.

Клас TaskPool інкапсулює задачі, він містить 3 клієнтських метода що відповідають 3 типам задач, алгоритми яких були розроблені у розділі 3.1.

Також клас містить клієнтський метод який виконує операцію, коли у нас лише одна задача.

Клас Executor є виконавчим, від запускає задачі на виконання, і обчислює їх час роботи.

**3.4 Тестування ПРГ2**

Для тестування ПРГ2 використовувалось те ж саме апаратне і програмне забезпечення, що і для програми ПРГ1.

Для обчислення часу роботи програми використовувалась функція MPI.COMM\_WORLD.Reduce(), що знаходить максимальне значення часу роботи програми.

Результати тестування і проведених досліджень ефективності розробленої програми наведено в таблицях Таблиця 3.1‑Таблиця 3.3.

Таблиця 3.1 – Час виконання програми ПРГ2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | Т1, мс | Т2, мс | Т4, мс | Т6, мс |
| 900 | 12870 | 6833 | 5039 | 3791 |
| 1800 | 161148 | 81432 | 49346 | 35163 |
| 2400 | 405016 | 207980 | 124052 | 91338 |

На основі даних із таблиці Таблиця 3.1 виконано розрахунок значень коефіцієнтів прискорення, які наведені в таблиці Таблиця 3.2.

Таблиця 3.2 – Коефіцієнти прискорення для програми ПРГ2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | Кількість процесорів (P) | | | |
| 1 | 2 | 4 | 6 |
| 900 | 1,00 | 1,88 | 2,55 | 3,39 |
| 1800 | 1,00 | 1,98 | 3,27 | 4,58 |
| 2400 | 1,00 | 1,95 | 3,26 | 4,43 |

Коефіцієнти ефективності (таблиця Таблиця 3.3) обчислено за даними таблиці Таблиця 3.2.

Таблиця 3.3 – Коефіцієнти ефективності для програми ПРГ2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | Кількість процесорів | | | |
| 1 | 2 | 4 | 6 |
| 900 | 100,00% | 94,18% | 63,85% | 56,58% |
| 1800 | 100,00% | 98,95% | 81,64% | 76,38% |
| 2400 | 100,00% | 97,37% | 81,62% | 73,90% |

Використовуючи таблиці Таблиця 3.2‑Таблиця 3.3 побудовано графіки зміни коефіцієнтів прискорення і ефективності в залежності від *N* і .

Рисунок 3.2 – Графік зміни коефіцієнту прискорення програми ПРГ2 в залежності від кількості ядер при N=900

Рисунок 3.3 – Графік зміни коефіцієнту прискорення програми ПРГ2 в залежності від кількості ядер при N=1800

Рисунок 3.4 – Графік зміни коефіцієнту прискорення програми ПРГ2 в залежності від кількості ядер при N=2400

Рисунок 3.5 – Графік зміни коефіцієнту ефективності програми ПРГ2 в залежності від кількості ядер при N=900

Рисунок 3.6 – Графік зміни коефіцієнту ефективності програми ПРГ2 в залежності від кількості ядер при N=1800

Рисунок 3.7 – Графік зміни коефіцієнту ефективності програми ПРГ2 в залежності від кількості ядер при N=2400

## Висновки до розділу 3

Виконано розробку програми ПРГ2 для ПКС ЛП з використанням мови С++ і бібліотеки MPI. Тестування програми показало наступне:

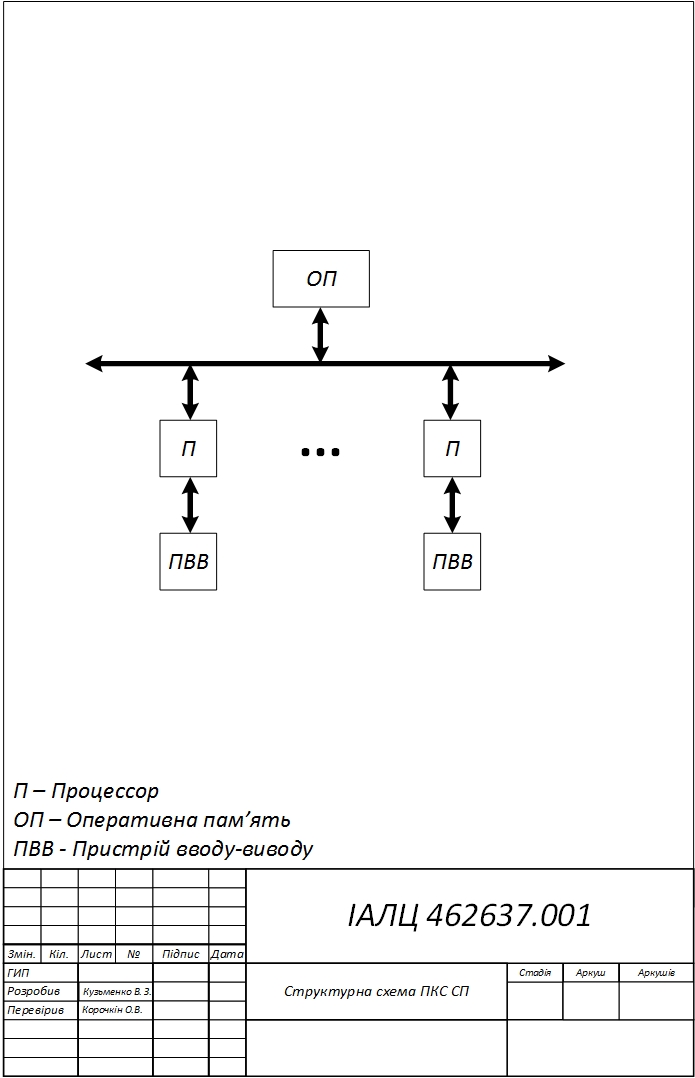
* Використання багатоядерної ПКС та програми ПРГ2 забезпечує скорочення часу обчислення заданої математичної задачі. Значення  лежать в межах від 1,98 до 5,09.
* Максимальне значення  забезпечує ПКС з  та .
* Мінімальне значення  виявлено у ПКС з .
* З ростом N  лінійно збільшується.
* Значення  змінюються від 66,4% до 99,0%.
* Найефективніше програма ПРГ2 використовує ПКС з . При цьому .
* Зі зростанням  від 1 до 5  лінійно спадає від 99,0% до 71,0% при . Далі при  зростає до 83,33%.

**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

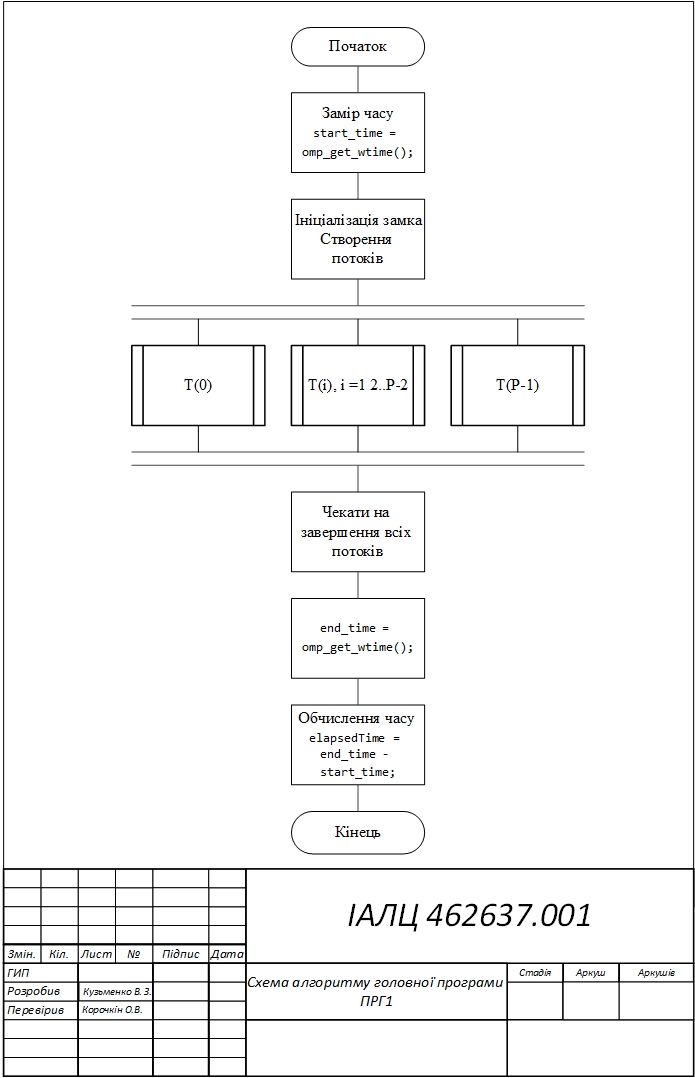
1. Знайомство з архітектурою шестиядерних процесорів AMD Phenom II X6 на ядрі Thuban [Електронний ресурс] .— Режим доступу: http://www.easycom.com.ua/data/cpu/1004272158/?lang=ukr
2. Огляд шестиядерного процесора AMD Phenom II X6 1075T[Електронний ресурс] .— Режим доступу: http://www.easycom.com.ua/cpu/amd\_phenom\_ii\_x6\_1075t/?lang=ukr
3. Офіційна презентація процесорів лінійки AMD FX [Електронний ресурс] .— Режим доступу: http://www.easycom.com.ua/ittech/ofitsialnaya\_prezentatsiya\_novyh\_vysokoproizvoditelnyh\_protsessorov\_lineyki\_amd\_fx/?lang=ukr
4. Огляд шестиядерного процесора AMD Phenom II X6 1075T[Електронний ресурс] .— Режим доступу: http://www.easycom.com.ua/cpu/amd\_phenom\_ii\_x6\_1075t/?lang=ukr
5. Огляд шестиядерного процесора AMD Phenom II X6 1055T[Електронний ресурс] .— Режим доступу: http://www.easycom.com.ua/cpu/amd\_phenom\_ii\_x6\_1055t/?lang=ukr
6. Огляд шестиядерного процесора AMD Phenom II X6 1100T[Електронний ресурс] .— Режим доступу: http://www.easycom.com.ua/cpu/amd\_phenom\_ii\_x6\_1100t/?lang=ukr
7. Огляд шестиядерного процесора AMD FX-6100 на базі архітектури Bulldozer[Електронний ресурс] .— Режим доступу: http://www.easycom.com.ua/cpu/amd\_fx-6100/?lang=ukr Last access: 27.03.2013. – Назва з екрану.
8. Процессоры AMD FX[Електронний ресурс] .— Режим доступу: http://www.amd.com/ru/products/desktop/processors/amdfx/Pages/amdfx.aspx
9. Процессоры AMD Phenom™ II [Електронний ресурс] .— Режим доступу: http://www.amd.com/ru/products/desktop/processors/phenom-ii/Pages/phenom-ii.aspx
10. Жуков І.А., Корочкін О.В. Паралельні та розподілені обчислення: Навч. посібник [Текст]. – К.: Корнійчук, 2005. – 226 с. – ISBN 996-7599-36-1.

ДОДАТКИ

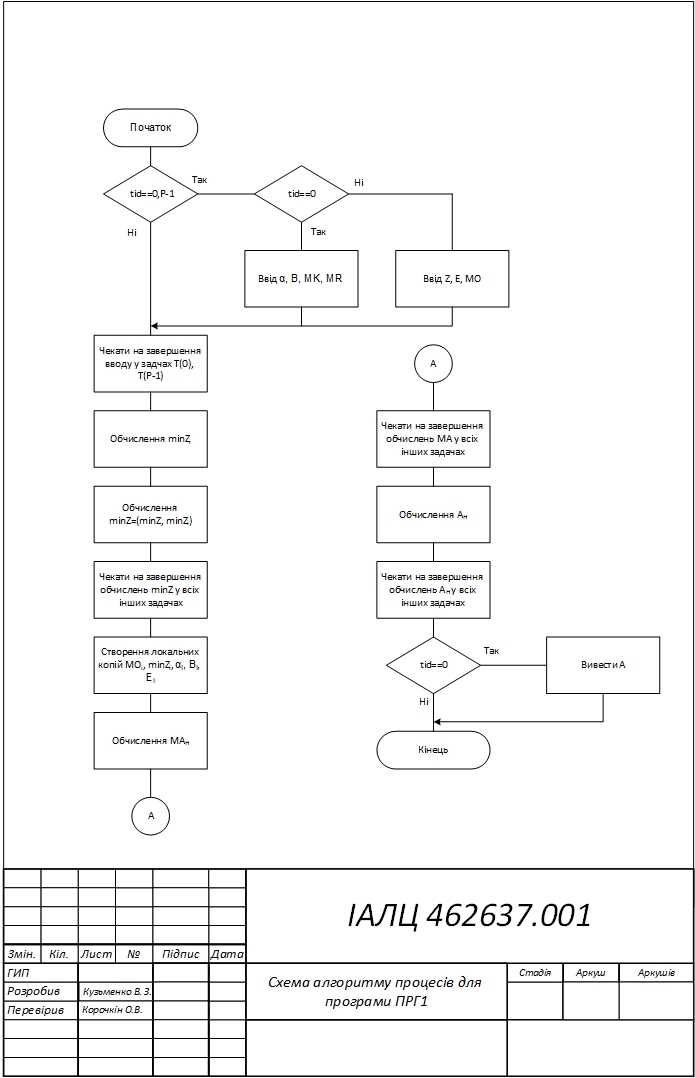
1. Структурна схема ПКС СП



1. Схема алгоритму головної програми зі вказанням паралельних ділянок для ПРГ1



1. Схема алгоритму процесів для програми ПРГ1



1. Лістинг програми ПРГ1

/\*\*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* Програмування паралельний комп'ютерних сисем

\* Курсова робота. ПРГ1. Бібліотека OpenMP

\*

\* Завдання: A = B(MO\*MK)\*a + min(Z)\*E\*MR

\*

\* Файл pro2\_course\_work\_prg1.cpp

\* Автор Кузьменко Володимир

\* Група ІО-21

\* Датаe 23.03.15

\*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

#include "stdafx.h"

#include "omp.h"

#include <windows.h>

#include <iostream>

#include "operations.h"

#include <ctime>

#include <clocale>

using namespace std;

#pragma comment(linker, "/stack:160000000")

const int P = 1;

int main()

{

vector A = new int[N], B, E, Z;

matrix MO, MK, MR, MA;

int alfa;

int minZ = 9999999999;

const int H = N / P;

omp\_lock\_t lock\_Copy;

double start\_time = omp\_get\_wtime();

setlocale(LC\_ALL, "Russian");

omp\_init\_lock(&lock\_Copy);

omp\_set\_num\_threads(P);

#pragma omp parallel

{

int tid = omp\_get\_thread\_num();

MA = initMatrix();

cout << "Задача " << tid << " стартувала"<<endl;

//switch (tid)

{

/\*1. Якщо tid = 0 Ввести α, B, MK, MR.\*/

// case 0:

alfa = 1;

B = inputVector(1);

MK = inputMatrix(1);

MR = inputMatrix(1);

// break;

/\*2. Якщо tid = P-1 ввести Z, E, MO.\*/

// case P-1:

Z = inputVector(1);

E = inputVector(1);

MO = inputMatrix(1);

// break;

}

/\*3. Бар’єр для усіх задач. Синхронізація по вводу\*/

#pragma omp barrier

/\*4. Обчислення minZi = min(Z), i = (0,P-1) \*/

int minZid = 9999999999;

for (int i = tid\*H; i < (tid+1)\*H; i++)

{

if (Z[i]< minZid)

{

minZid = Z[i];

}

}

/\*5. Обчислення minZ = min(minZ, minZi), i = (0,P-1) ̅.\*/

#pragma omp critical

{

minZ = min(minZ, minZid);

}

/\*6. Бар’єр для усіх задач. Синхронізація по обрахунку minZ\*/

#pragma omp barrier

matrix MOid;

vector Bid;

vector Eid;

int alfaId;

/\*7. Копіювати MOi=MO, minZi = minZ, αi = α, Bi = B, Ei = E\*/

omp\_set\_lock(&lock\_Copy);

MOid = copyMatrix(MO);

alfaId = alfa;

minZid = minZ;

Bid = copyVector(B);

Eid = copyVector(E);

omp\_unset\_lock(&lock\_Copy);

/\*8. Обчислення MAH = MOi∙MKH\*/

int sum;

for (int i = tid \* H; i < (tid + 1) \* H; i++) {

for (int j = 0; j < N; j++) {

sum = 0;

for (int z = 0; z < N; z++) {

sum += MOid[i][j] \* MK[j][z];

}

MA[i][j] = sum;

}

}

/\*9. Бар’єр для усіх задач. Синхронізація по обрахунку MA\*/

#pragma omp barrier

/\*10. Обчислення AH = αi∙Bi∙MAH + minZi∙Ei∙MRH, i = (0,P-1)\*/

int buf;

for (int i = tid\*H; i < (tid + 1)\*H; i++)

{

A[i] = 0;

for (int j = 0; j < N; j++)

{

A[i] += Bid[j] \* MA[j][i];

}

A[i] = A[i] \* alfaId;

buf = 0;

for (int j = 0; j < N; j++)

{

buf += Eid[j] \* MR[j][i];

}

A[i] += buf\*minZid;

}

/\*11. Бар’єр для усіх задач. Синхронізація по обчисленню A\*/

#pragma omp barrier

/\*12. Якщо tid = 0 вивести результат AH.\*/

if (tid==0)

{

output(A);

}

cout << "Задача " << tid << " завершилась" << endl;

}

double end\_time = omp\_get\_wtime();

double elapsedTime = end\_time - start\_time;

cout << "Час роботи: " << elapsedTime << endl;

return 0;

}

/\*\*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* Програмування паралельний комп'ютерних сисем

\* Курсова робота. ПРГ1. Бібліотека OpenMP

\*

\* Завдання: A = B(MO\*MK)\*a + min(Z)\*E\*MR

\*

\* Файл operations.cpp

\* Автор Кузьменко Володимир

\* Група ІО-21

\* Датаe 23.03.15

\*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

#include "stdafx.h"

#include <windows.h>

using namespace std;

#include "operations.h"

#include <iostream>

/\*Введення вектора\*/

vector inputVector(int value){

vector result = new int[N];

for (int i = 0; i < N; i++)

{

result[i] = value;

}

return result;

}

/\*введення матриці\*/

matrix inputMatrix(int value){

matrix result = new vector[N];

for (int i = 0; i < N; i++)

{

result[i] = new int[N];

}

for (int i = 0; i < N; i++)

{

for (int j = 0; j < N; j++)

{

result[i][j] = value;

}

}

return result;

}

/\*Виведення вектора\*/

void output(vector v) {

if (N <= 20) {

for (int i = 0; i < N; i++) {

cout << v[i] << " ";

}

cout << endl;

}

}

/\*Копіювання вектора\*/

vector copyVector(vector v){

vector result = new int[N];

for (int i = 0; i < N; i++)

{

result[i] = v[i];

}

return result;

}

/\*Копіювання матриці\*/

matrix copyMatrix(matrix m){

matrix result = new vector[N];

for (int i = 0; i < N; i++)

{

result[i] = new int[N];

}

for (int i = 0; i < N; i++)

{

for (int j = 0; j < N; j++)

{

result[i][j] = m[i][j];

}

}

return result;

}

/\*Ініціалізація матриці\*/

matrix initMatrix(){

matrix result = new vector[N];

for (int i = 0; i < N; i++)

{

result[i] = new int[N];

}

return result;

}

/\*\*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\*

\* Програмування паралельний комп'ютерних сисем

\* Курсова робота. ПРГ1. Бібліотека OpenMP \*

\*

\* Завдання: A = B(MO\*MK)\*a + min(Z)\*E\*MR

\*

\* Файл operations.h

\* Автор Кузьменко Володимир

\* Група ІО-21

\* Датаe 23.03.15

\*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\*/

typedef int\* vector;

typedef int\*\* matrix;

const int N =2400;

vector inputVector(int);

matrix inputMatrix(int);

void output(vector);

void output(matrix);

vector copyVector(vector);

matrix copyMatrix(matrix);

matrix initMatrix();

**Додаток Д. Лістинг програми ПРГ2**

import mpi.\*;

/\*\*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* Програмування паралельний комп'ютерних сисем

\* Курсова робота. ПРГ2. Бібліотека MPI

\*

\* Завдання: A = B(MO\*MK)\*a + min(Z)\*E\*MR

\*

\* Файл Executor.java

\* Автор Кузьменко Володимир

\* Група ІО-21

\* Датаe 23.04.15

\*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

public class Executor {

public static int N;

public static int P;

public static int H;

public static int k;

public static void main(String[] args) throws Exception {

MPI.Init(args);

System.out.println("Task "+MPI.COMM\_WORLD.Rank()+" started");

P = Integer.parseInt(args[1]);

N = Integer.parseInt(args[3]);

H = N / P;

k = P / 2;

long start;

long []timework = new long[1];

long [] buf = new long[1];

TaskPool pool = new TaskPool();

start = System.currentTimeMillis();

if (MPI.COMM\_WORLD.Size()==1){

pool.singleThreadTask();

}else{

pool.leftTaskGroup();

pool.middleTaskGroup();

pool.rightTaskGroup();

}

buf[0] = System.currentTimeMillis() - start;

System.out.print("Task "+MPI.COMM\_WORLD.Rank()+" finished, time: ");

System.out.println(buf[0]);

System.out.println();

MPI.COMM\_WORLD.Reduce(buf,0,timework,0,1,MPI.LONG,MPI.MAX,0);

if(MPI.COMM\_WORLD.Rank()==0){

//System.out.println("Time work: "+timework[0]);

}

MPI.Finalize();

}

}

import java.io.Serializable;

import java.util.ArrayList;

/\*\*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* Програмування паралельний комп'ютерних сисем

\* Курсова робота. ПРГ2. Бібліотека MPI

\*

\* Завдання: A = B(MO\*MK)\*a + min(Z)\*E\*MR

\*

\* Файл MessageBox.java

\* Автор Кузьменко Володимир

\* Група ІО-21

\* Датаe 23.04.15

\*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

public class MessageBox implements Serializable {

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

private ArrayList<Matrix> matrixs = new ArrayList<Matrix>(3);

private ArrayList<Vector> vectors = new ArrayList<Vector>(3);

private ArrayList<Integer> values = new ArrayList<>(2);

public void addMatrix(Matrix matrix){

matrixs.add(matrix);

}

public Matrix setMatrix(int key, Matrix matrix){

return matrixs.set(key, matrix);

}

public void AddVector(Vector vector){

vectors.add(vector);

}

public Vector setVector(int key, Vector vector){

return vectors.set(key, vector);

}

public void AddValue(int value){

values.add(value);

}

public int setValue(int key, int value){

return values.set(key, value);

}

public Matrix getMatrix(int key){

return matrixs.get(key);

}

public Vector getVector(int key){

return vectors.get(key);

}

public int getValue(int key){

return values.get(key);

}

}

import mpi.MPI;

import mpi.Request;

/\*\*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* Програмування паралельний комп'ютерних сисем

\* Курсова робота. ПРГ2. Бібліотека MPI

\*

\* Завдання: A = B(MO\*MK)\*a + min(Z)\*E\*MR

\*

\* Файл TaskPool.java

\* Автор Кузьменко Володимир

\* Група ІО-21

\* Датаe 23.04.15

\*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

public class TaskPool {

private int P = MPI.COMM\_WORLD.Size();

private int H = Executor.H;

private int k = P / 2;

private int rank;

int leftRank;

int rightRank;

int minRank = (k - 1) / 2;

int endFirsRowRank = k - 1;

int middleRightRank = P - 1;

private MessageKeys KEY;

boolean notFourCore = true;

boolean notTwoCore = true;

private Vector B, E, Z, A;

private Matrix MK, MR, MO;

private int alfa;

private int localMin = Integer.MAX\_VALUE ;

private MessageBox[] box = new MessageBox[1];

private MessageBox message = new MessageBox();

public TaskPool() {

rank = MPI.COMM\_WORLD.Rank();

rightRank = rank + 1;

leftRank = rank - 1;

if (P == 4) {

minRank = 1;

endFirsRowRank = 1;

notFourCore = false;

}

if (P == 2) {

notTwoCore = false;

endFirsRowRank = 1;

minRank=1;

}

}

public void singleThreadTask(){

Vector B, E, Z, A;

Matrix MK, MR, MO;

int alfa = 1;

E = CalculateUtils.inputVector(1);

Z = CalculateUtils.inputVector(10);

Z.set(7,1);

MO = CalculateUtils.inputMatrix(1);

B = CalculateUtils.inputVector(1);

MR = CalculateUtils.inputMatrix(1);

MK = CalculateUtils.inputMatrix(1);

A = CalculateUtils.add(CalculateUtils.mult(alfa,

CalculateUtils.mult(B, CalculateUtils.mult(MK, MO))),

CalculateUtils.mult(CalculateUtils.min(Z), CalculateUtils.mult(E, MR)));

CalculateUtils.outputVector(A);

}

@SuppressWarnings("static-access")

public void leftTaskGroup() {

if (rank == 0 || (rank == k && notTwoCore)) {

if (rank == 0) {

alfa = 1;

B = CalculateUtils.inputVector(1);

MR = CalculateUtils.inputMatrix(1);

MK = CalculateUtils.inputMatrix(1);

message.AddValue(alfa);

message.AddVector(B);

if(notTwoCore){

message.addMatrix(MK.copy(MK.size() / 2, MK.size() / 2));

message.addMatrix(MR.copy(MR.size() / 2, MR.size() / 2));

MR = MR.copy(0, MK.size() / 2);

MK = MK.copy(0, MK.size()/2);

box[0] = message;

MPI.COMM\_WORLD.Isend(box, 0, 1, MPI.OBJECT, k, KEY.SEND\_RECIVE\_B\_MK\_MR\_alfa\_TO\_k);

}else{

message.addMatrix(MK.copy(H, MR.size() - H));

message.addMatrix(MR.copy(H, MR.size() - H));

box[0] = message;

}

message.setMatrix(0, MK.copy(H, MR.size() - H));

message.setMatrix(1, MR.copy(H, MR.size() - H));

MPI.COMM\_WORLD.Isend(box, 0, 1, MPI.OBJECT, rightRank, KEY.SEND\_RECIVE\_B\_MK\_MR\_alfa);

box[0] = null;

}

if (rank == k && notTwoCore) {

MPI.COMM\_WORLD.Recv(box, 0, 1, MPI.OBJECT, 0,KEY.SEND\_RECIVE\_B\_MK\_MR\_alfa\_TO\_k);

MK = box[0].getMatrix(0);

MR = box[0].getMatrix(1);

box[0].setMatrix(0, MK.copy(H, MK.size() - H));

box[0].setMatrix(1, MR.copy(H, MR.size() - H));

MPI.COMM\_WORLD.Isend(box, 0, 1, MPI.OBJECT, rightRank, KEY.SEND\_RECIVE\_B\_MK\_MR\_alfa);

alfa = box[0].getValue(0);

B = box[0].getVector(0);

box[0] = null;

}

MK = MK.copy(0, H);

MR = MR.copy(0, H);

MPI.COMM\_WORLD.Recv(box, 0, 1, MPI.OBJECT, rightRank, KEY.SEND\_RECIVE\_Z\_E\_MO);

E = box[0].getVector(0);

Z = box[0].getVector(1);

calcMinButtonTask();

MO = box[0].getMatrix(0);

box[0] = null;

Request r = MPI.COMM\_WORLD.Irecv(box, 0, 1, MPI.OBJECT, rightRank, KEY.SEND\_RECIVE\_RESULT\_A);

A = CalculateUtils.add(CalculateUtils.mult(alfa,

CalculateUtils.mult(B, CalculateUtils.mult(MK, MO))),

CalculateUtils.mult(localMin, CalculateUtils.mult(E, MR)));

r.Wait();

A.merge(box[0].getVector(0));

if (rank == k && notTwoCore) {

box[0].setVector(0, A);

MPI.COMM\_WORLD.Send(box, 0, 1, MPI.OBJECT, 0, KEY.SEND\_RECIVE\_A\_TO\_0);

}

if (rank == 0) {

if(notTwoCore){

box[0] = null;

MPI.COMM\_WORLD.Recv(box, 0, 1, MPI.OBJECT, k, KEY.SEND\_RECIVE\_A\_TO\_0);

A.merge(box[0].getVector(0));

}

CalculateUtils.outputVector(A);

}

}

}

@SuppressWarnings("static-access")

public void middleTaskGroup() {

if (rank != 0 && rank != k && rank != endFirsRowRank && rank != P - 1) {

Request [] inputRequest = new Request[2];

inputRequest[0] = MPI.COMM\_WORLD.Irecv(box, 0, 1, MPI.OBJECT, leftRank, KEY.SEND\_RECIVE\_B\_MK\_MR\_alfa);

inputRequest[0].Wait();

MK = box[0].getMatrix(0);

MR = box[0].getMatrix(1);

box[0].setMatrix(0, MK.copy(H, MK.size() - H));

box[0].setMatrix(1, MR.copy(H, MR.size() - H));

MPI.COMM\_WORLD.Isend(box, 0, 1, MPI.OBJECT, rightRank, KEY.SEND\_RECIVE\_B\_MK\_MR\_alfa);

B = box[0].getVector(0);

alfa = box[0].getValue(0);

box[0] = null;

inputRequest[1]= MPI.COMM\_WORLD.Irecv(box, 0, 1, MPI.OBJECT, rightRank, KEY.SEND\_RECIVE\_Z\_E\_MO);

inputRequest[1].Wait();

Z = box[0].getVector(1);

box[0].setVector(1, Z.copy(H, Z.size() - H));

MPI.COMM\_WORLD.Isend(box, 0, 1, MPI.OBJECT, leftRank, KEY.SEND\_RECIVE\_Z\_E\_MO);

Z = Z.copy(0, H);

calcMinButtonTask();

E = box[0].getVector(0);

MO = box[0].getMatrix(0);

MK = MK.copy(0, H);

MR = MR.copy(0, H);

Request r = MPI.COMM\_WORLD.Irecv(box, 0, 1, MPI.OBJECT, rightRank, KEY.SEND\_RECIVE\_RESULT\_A);

A = CalculateUtils.add(CalculateUtils.mult(alfa,

CalculateUtils.mult(B, CalculateUtils.mult(MK, MO))),

CalculateUtils.mult(localMin, CalculateUtils.mult(E, MR)));

r.Wait();

A.merge(box[0].getVector(0));

box[0].setVector(0, A);

MPI.COMM\_WORLD.Isend(box, 0, 1, MPI.OBJECT, leftRank, KEY.SEND\_RECIVE\_RESULT\_A);

}

}

@SuppressWarnings("static-access")

public void rightTaskGroup() {

if (rank == P - 1 || (rank == endFirsRowRank && notTwoCore)) {

if (rank == endFirsRowRank) {

E = CalculateUtils.inputVector(1);

Z = CalculateUtils.inputVector(10);

Z.set(7,1);

MO = CalculateUtils.inputMatrix(1);

message.addMatrix(MO);

message.AddVector(E);

if (notTwoCore) {

message.AddVector(Z.copy(k \* H, Z.size() / 2));

box[0] = message;

MPI.COMM\_WORLD.Isend(box, 0, 1, MPI.OBJECT, P - 1, KEY.SEND\_RECIVE\_MO\_Z\_E\_TO\_P\_1);

box[0].setVector(1, Z.copy(H, Z.size() / 2 - H));

message = box[0];

}else{

message.AddVector(Z.copy(H, Z.size()/2));

}

}

if (rank == P - 1 && notTwoCore) {

MPI.COMM\_WORLD.Recv(box, 0, 1, MPI.OBJECT, k - 1, KEY.SEND\_RECIVE\_MO\_Z\_E\_TO\_P\_1);

MO = box[0].getMatrix(0);

E = box[0].getVector(0);

Z = box[0].getVector(1);

box[0].setVector(1, Z.copy(H, Z.size() - H));

message = box[0];

}

box[0] = null;

MPI.COMM\_WORLD.Recv(box, 0, 1, MPI.OBJECT, leftRank, KEY.SEND\_RECIVE\_B\_MK\_MR\_alfa);

B = box[0].getVector(0);

MK = box[0].getMatrix(0);

MR = box[0].getMatrix(1);

alfa = box[0].getValue(0);

box[0] = message;

MPI.COMM\_WORLD.Isend(box, 0, 1, MPI.OBJECT, leftRank, KEY.SEND\_RECIVE\_Z\_E\_MO);

Z = Z.copy(0, H);

calcMinButtonTask();

A = CalculateUtils.add(CalculateUtils.mult(alfa,

CalculateUtils.mult(B, CalculateUtils.mult(MK, MO))),

CalculateUtils.mult(localMin, CalculateUtils.mult(E, MR)));

box[0] = null;

box[0] = new MessageBox();

box[0].AddVector(A);

MPI.COMM\_WORLD.Isend(box, 0, 1, MPI.OBJECT, leftRank, KEY.SEND\_RECIVE\_RESULT\_A);

}

}

private void calcMinButtonTask() {

if (rank > endFirsRowRank) {

// нижній рядок

int min[] = new int[] { CalculateUtils.min(Z) };

MPI.COMM\_WORLD.Send(min, 0, 1, MPI.INT, rank - k, 103);

min = new int[1];

MPI.COMM\_WORLD.Recv(min, 0, 1, MPI.INT, rank - k, 130);

localMin = min[0];

} else {

int min[] = new int[1];

if(notTwoCore){

MPI.COMM\_WORLD.Recv(min, 0, 1, MPI.INT, rank + k, 103);

}else{

min[0] = Integer.MAX\_VALUE;

}

localMin = Math.min(min[0], CalculateUtils.min(Z));

if (rank > minRank) {

int[] res = new int[1];

int buf = localMin;

if (rank != endFirsRowRank) {

MPI.COMM\_WORLD.Recv(res, 0, 1, MPI.INT, rank + 1, 104);

buf = Math.min(localMin, res[0]);

}

res[0] = buf;

MPI.COMM\_WORLD.Send(res, 0, 1, MPI.INT, rank - 1, 104);

} else {

if (rank != minRank) {

int [] res = new int[1];

res[0] = localMin;

if (rank != 0) {

MPI.COMM\_WORLD.Recv(res, 0, 1, MPI.INT, rank - 1, 104);

localMin = Math.min(localMin, res[0]);

res[0] = localMin;

}

MPI.COMM\_WORLD.Send(res, 0, 1, MPI.INT, rank + 1, 104);

}

}

if (rank == minRank) {

int res[] = new int[1];

if (notFourCore && notTwoCore) {

MPI.COMM\_WORLD.Recv(res, 0, 1, MPI.INT, rank + 1, 104);

this.localMin = Math.min(this.localMin, res[0]);

res = new int[1];

}

MPI.COMM\_WORLD.Recv(res, 0, 1, MPI.INT, rank - 1, 104);

this.localMin = Math.min(this.localMin, res[0]);

// знайшли мінімум

res[0] = this.localMin;

MPI.COMM\_WORLD.Send(res, 0, 1, MPI.INT, rank - 1, 122);

if (notFourCore && notTwoCore)

MPI.COMM\_WORLD.Send(res, 0, 1, MPI.INT, rank + 1, 122);

}

else if (rank > 0 && rank < minRank) {

min = new int[1];

MPI.COMM\_WORLD.Recv(min, 0, 1, MPI.INT, rank + 1, 122);

this.localMin = min[0];

MPI.COMM\_WORLD.Send(min, 0, 1, MPI.INT, rank - 1, 122);

} else if (rank != 0 && rank != endFirsRowRank) {

min = new int[1];

MPI.COMM\_WORLD.Recv(min, 0, 1, MPI.INT, rank - 1, 122);

this.localMin = min[0];

MPI.COMM\_WORLD.Send(min, 0, 1, MPI.INT, rank + 1, 122);

} else if (rank == 0) {

min = new int[1];

MPI.COMM\_WORLD.Recv(min, 0, 1, MPI.INT, rank + 1, 122);

this.localMin = min[0];

} else {

min = new int[1];

MPI.COMM\_WORLD.Recv(min, 0, 1, MPI.INT, rank - 1, 122);

localMin = min[0];

}

if(notTwoCore){

min[0] = localMin;

MPI.COMM\_WORLD.Send(min, 0, 1, MPI.INT, rank + k, 130);

}

}

}

}

import java.io.Serializable;

/\*\*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* Програмування паралельний комп'ютерних сисем

\* Курсова робота. ПРГ2. Бібліотека MPI

\*

\* Завдання: A = B(MO\*MK)\*a + min(Z)\*E\*MR

\*

\* Файл Vector.java

\* Автор Кузьменко Володимир

\* Група ІО-21

\* Датаe 23.04.15

\*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

public class Vector implements Serializable {

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

private int[] array;

public Vector(int n) {

array = new int[n];

}

public void set(int index, int value) {

array[index] = value;

}

public int get(int index) {

return array[index];

}

public int size() {

return array.length;

}

public String toString() {

String res = "";

for (int i = 0; i < array.length; i++){

res += " " + array[i];

}

res += "\n";

return res;

}

public Vector copy(int start,int size){

Vector result = new Vector(size);

for (int i = 0; i < result.size(); i++) {

result.array[i] = this.array[i+start];

}

return result;

}

public void merge(Vector vector){

int [] buf = array;

array = new int [buf.length+vector.size()];

System.arraycopy(buf, 0, array, 0,buf.length );

System.arraycopy(vector.array, 0, this.array, buf.length, vector.array.length);

}

public void reverse(){

for (int i = 0; i < array.length/2; i++) {

int buf = array[i];

array[i] = array[i +array.length/2];

array[array.length/2+i] = buf;

}

}

}

import java.io.Serializable;

/\*\*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* Програмування паралельний комп'ютерних сисем

\* Курсова робота. ПРГ2. Бібліотека MPI

\*

\* Завдання: A = B(MO\*MK)\*a + min(Z)\*E\*MR

\*

\* Файл Matrix.java

\* Автор Кузьменко Володимир

\* Група ІО-21

\* Датаe 23.04.15

\*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

public class Matrix implements Serializable {

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

private Vector[] array;

public Matrix(int n) {

array = new Vector[n];

for (int i = 0; i < array.length; i++){

array[i] = new Vector(n);

}

}

public void set(int n, int m, int val) {

array[n].set(m, val);

}

public int get(int n, int m) {

return array[n].get(m);

}

public Vector get(int index) {

return array[index];

}

public int size() {

return array.length;

}

public String toString() {

String res = "";

for (int i = 0; i < array.length; i++){

res += array[i].toString();

}

return res;

}

public Matrix copy(int start,int size){

Matrix result = new Matrix(size);

for (int i = 0; i < result.size(); i++) {

result.array[i] = array[i+start].copy(0, Executor.N);

}

return result;

}

}

/\*\*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* Програмування паралельний комп'ютерних сисем

\* Курсова робота. ПРГ2. Бібліотека MPI

\*

\* Завдання: A = B(MO\*MK)\*a + min(Z)\*E\*MR

\*

\* Файл MessageKeys.java

\* Автор Кузьменко Володимир

\* Група ІО-21

\* Датаe 23.04.15

\*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

public class MessageKeys {

static int SEND\_RECIVE\_B\_MK\_MR\_alfa = 1;

static int SEND\_RECIVE\_Z\_E\_MO = 2;

static int SEND\_RECIVE\_RESULT\_A = 3;

static int SEND\_RECIVE\_B\_MK\_MR\_alfa\_TO\_k=4;

static int SEND\_RECIVE\_MO\_Z\_E\_TO\_P\_1=5;

static int SEND\_RECIVE\_A\_TO\_0 = 6;

}