НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

КИЇВСЬКИЙ ПОЛІТЕХНИЧНИЙ ІНСТИТУТ

ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Кафедра обчислювальної техніки

К У Р С О В А Р О Б О Т А

*з дисципліни:* «Паралельні та розподілені обчислення»

*на тему:* «Розробка програмного забезпечення для

паралельних комп’ютерних систем»

Виконав: Воробйов В.В.

Група: ІО- 83

Перевірив: \_\_\_\_\_\_\_\_\_\_\_\_\_ Корочкін О. В.

Захищено з оцінкою\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_2011 р.

Київ - 2011 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

КИЇВСЬКИЙ ПОЛІТЕХНИЧНИЙ ІНСТИТУТ

ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Кафедра обчислювальної техніки

***ЗАВДАННЯ НА КУРСОВУ РОБОТУ***

*з дисципліни:* «Паралельні та розподілені обчислення»

*на тему:* «Розробка програмного забезпечення для

паралельних комп’ютерних систем»

Дата отримання 31.03.2011р.

Термін виконання 19.05.2011р.

Керівник роботи доц.. Корочкін О.В.

ЗАВДАННЯ

1.Огляд трансп’ютерних систем та мови программування Оккам.

2.Розробка програмного забезпечення для масштабованих паралельних комп’ютерних систем. Мови та бібліотеки паралельного програмування Java і МРІ. Засоби організації взаємодії процесів: захищений модуль, механізм посилки повідомлень. Математична задача: А = В + С \* (МО \* МХ – МЕ).

3. Тестування програмного забезпечення в багатоядерної комп’ютерної системі

ПЕРЕЛІК ГРАФИЧНОЇ ДОКУМЕНТАЦІЇ

1. Структурна схема ПКС ОП

2. Структурна схема ПКС ЛП

3. Схема алгоритму основної програми для ПКС ОП

4. Схеми алгоритмів процесів для ПКС с ОП

5. Схема алгоритму основної програми для ПКС ЛП

6. Схеми алгоритмів процесів для ПКС ЛП

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Жуков І., Корочікн О. Паралельні та розподілені обчислення. – Київ,

Корнійчук, 2005. – 226 с.

2. Корочкин А.В. Ада 95: Введение в программирование. - Киев; Свит, .

1998. - 260 с.

ПІДПИС КЕРІВНИКА РОБОТИ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Техническое задание на курсовую работу**

1. Область застосування

1. Розроблюване програмне забезпечення може бути застосовано в галузі паралельних математичних обчислень для масштабованих комп'ютерних систем з загальної та локальної пам'яттю.

2. Підстави для розробки

Підставою для розробки служить ТЗ на курсову роботу.

3. Мета курсової роботи

Метою даної роботи є закріплення отриманих знань з дисципліни "Паралельні та розподілені обчислення ", а також отримання навичок і досвіду в розробці програмного забезпечення для комп'ютерних систем з різною структурною організацією.

4. Призначення роботи

Розробка програмного забезпечення (ПЗ) для ПКС з спільною та локальною пам'ятью. Проведення досліджень ефективності ПЗ.

5. Вихідні дані та виконання роботи

5.1. Завдання для розділу 1: Огляд трансп’ютерних систем та мови программування Оккам.

5.2 Завдання для розділів 2 і 3: Математична задача: А = В + С \* (МО \* МХ - МО). Розглядаються вектора і матриці розмірності N.

5.3. Структура паралельної комп'ютерної системи з спільною пам'яттю (рисунок. 1).

5. 4. Структура паралельної комп'ютерної системи з локальною пам'яттю (рисунок. 2).



Рисунок.1. Структура ПКС ОП



Рис.2. Структура ПКС ЛП

5.5. Мови та бібліотеки програмування та засоби організації взаємодії процесів: мова Java і механізм моніторів для розділу 2, Java і МРІ - для розділу 3.

У КР необхідно:

Розробити алгоритм розв'язання заданого математичної задачі і досліджувати паралельний властивості завдання в рамках концепції необмеженого паралелізму для N і Р, де N - розмірність матриць, Р - кількість процесорів.

Виконати розробку програми вирішення заданого математичного виразу для заданих структур ПКС на заданих язиках (біблотеках) програмування. Описати структуру взаємодії завдань.

Виконати дослідження ефективності ПЗ завдання в залежності від значень Р і N. При цьому підрахувати часи виконання математичної задачі на різній кількості процессоров.Определіть значення коефіцієнтів прискорення (Кп) та ефективності (Ке).

6. Вимоги до складу і параметрів технічних засобів

Тестування ПЗ виконується з використання багатоядерної ПКС

7. Вимоги до інформаційної та програмної сумісності

Програма повинна працювати під управлінням ОС Windows.

8. Спеціальні вимоги

Тестування виконується на 4-х ядерної системі

9. Етапи і стадії розробки

- Узгодження технічного завдання

- Виконання курсової роботи.

- Захист курсової роботи.

ЗМІСТ

[РОЗДІЛ 1. ОГЛЯД ТРАНСП’ЮТЕРІВ ТА МОВИ ПРОГРАМУВАННЯ ОККАМ 5](#_Toc293439611)

[1.1 Класифікація паралельних обчислювальних систем 5](#_Toc293439612)

[1.2 Архітектура трансп’ютерів 10](#_Toc293439617)

[1.3 Архітектура трансп’ютера Т800 10](#_Toc293439619)

[1.4 Процеси в трансп’ютерах та апаратний планувальник 16](#_Toc293439622)

[1.5 Комунікаційні можливості трансп’ютерів 17](#_Toc293439624)

[1.6 Системи на основі трансп’ютерів 18](#_Toc293439625)

[1.7 Огляд мови Оккам 21](#_Toc293439631)

[Висновки до Розділу 1 23](#_Toc293439634)

[РОЗДІЛ 2. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ПАРАЛЕЛЬНИХ КОМП’ЮТЕРНИХ СИСТЕМ 25](#_Toc293439635)

[2.1 Розробка паралельного математичного алгоритму 25](#_Toc293439636)

[2.2 Аналіз завдання 25](#_Toc293439637)

[2.3 Розробка ПЗ для системи з спільною пам’яттю 27](#_Toc293439639)

[2.4 Розробка ПЗ для системи з локальною пам’яттю 29](#_Toc293439643)

[Висновки до Розділу 2 31](#_Toc293439647)

[РОЗДІЛ 3. ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 32](#_Toc293439648)

[3.2 Тестування ПЗ для системи з спільною пам’яттю 32](#_Toc293439649)

[3.3 Тестування ПЗ для системи з локальною пам’яттю 34](#_Toc293439650)

[Висновки до Розділу 3 36](#_Toc293439651)

[Висновки 37](#_Toc293439652)

[Література 39](#_Toc293439653)

[Додатки 41](#_Toc293439654)

# РОЗДІЛ 1. ОГЛЯД ТРАНСП’ЮТЕРІВ ТА МОВИ ПРОГРАМУВАННЯ ОККАМ

Розглядаються мікропроцесорні пристрої – трансп'ютери, якi є прикладом реалізації MIMD архітектури, тобто системи, в якій кожен елемент, що має свій процессор, асинхронно і незалежно виконують різні команди над різними частинами даних. Трансп'ютер можна з'єднувати між собою через лінки, вибудовуючи різні структури комп'ютерних систем з локальною пам'яттю. Для програмування трансп'ютерних систем використовують мову Оккам, розроблений спеціально для таких цілей. Мова Оккам розроблена на основі моделі відправки повідомлень.

## Класифікація паралельних обчислювальних систем

### Класифікація по Флінну

Самою ранньою і найбільш відомою є класифікація архітектур обчислювальних систем, запропонована в 1966 році М. Флінном. (23) Класифікація базується на кількості потоків команд і даних. Флінн классификує обчислювальні системи таким чином, що потоків команд і даних може бути один або безліч. На основі цього Флінн виділяє чотири класи архітектур: SISD, MISD, SIMD, MIMD. (6, 17) Класифікація за Фліну зображено в таблиці 1.1. Конвеєрні однопроцесорні комп'ютери класифікуються як SISD системи. Паралельні комп'ютери відносяться до SIMD або MIMD архитектурам. Якщо в системі тільки один процесор або всі процесори обробляють однакові інструкції, тоді така паралельно система класифікується як SIMD.

Таблиця 1.1 Классифікація паралельних обчислюваних систем по Флінну

|  |  |  |
| --- | --- | --- |
|  | Одиночный поток команд  (Single Instruction) | Множество потоков команд  (Multiple Instruction) |
| Одиночный поток данных  (Single Data) | SISD | MISD |
| Множество потоков данных  (Multiple Data) | SIMD | MIMD |

**SISD** (англ. Single Instruction, Single Data) - архітектура комп'ютера, в якій один процесор виконує один потік команд, оперуючи одним потоком даних. На рис. 1.1 показана загальна структура архітектури SISD. Комп'ютери з такою архітектурою це звичайні, «традиційні » фон Неймановскі послідовні комп'ютери, в яких у кожен момент часу виконується лише одна операція над одним елементом даних.

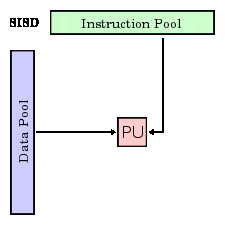


Рис. 1.1. SISD архітектура

**SIMD** (англ. single instruction, multiple data) - архітектура обчислювальних систем, що використовують принцип комп'ютерних обчислень, який дозволяє забезпечити паралелізм на рівні даних. Архітектура SIMD, вперше застосована в процесорах Pentium з технологією MMX (22) для операцій з цілими числами, відкрила шлях до розробки програмних аудіозасобів що реалізують 3D-звук, додатки по обробці двомірної графіки, стандартні засоби відтворення відео, традиційні додатки ділового призначення та інше. Одним з переваг цієї архітектури вважається те, що в цьому випадку більш ефективно реалізована логіка обчислень. До половини логічних інструкцій звичайного процесора пов'язане з управлінням виконанням машинних команд, а решта їх частина належить до роботи із внутрішньою пам'яттю процесора та виконання арифметичних операцій. У SIMD комп'ютері управління виконується контролером, а «арифметика» віддана процесорним елементам.

**MISD** (англ. Multiple Instruction stream, Single Data stream) - тип архітектури паралельних обчислень, де кілька функціональних модулів (два або більше) виконують різні операції над одними даними. На рис. 1.2 показаний загальний вид даної архітектури. Відмовостійкі комп'ютери, що виконують одні й ті ж команди надлишково з метою виявлення помилок, як випливає з визначення, належать до цього типу. Було створено небагато ЕОМ з MISD-архітектурою, оскільки MIMD і SIMD найчастіше є більш придатними для загальних методик паралельних даних. Вони забезпечують кращий масштабування та використання обчислювальних ресурсів, ніж архітектура MISD. Один з небагатьох реалізацій - систолічний масив процесорів, в якому процесори перебувають у вузлах регулярної решітки, роль ребер якій грають міжпроцесорних з'єднання. Всі процесорні елементи управляються загальним тактовим генератором. У кожному циклі роботи кожен процесорний елемент отримує дані від своїх сусідів, виконує одну команду і передає результат сусідам.

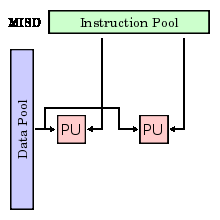


Рис. 1.2. MISD архітектура

**MIMD** (англ. Multiple Instruction stream, Multiple Data stream) - концепція архітектури комп'ютера, яка використовується для досягнення паралелізму обчислень. Як видно на рис. 1.3, машини мають кілька процесорів, які функціонують асинхронно і незалежно. MIMD-архітектури можуть бути використані в цілому ряді областей, таких як системи автоматизованого проектування, моделювання, а також комунікатор зв'язків, також такі системи активно використовуються для космічних цілей. MIMD машини можуть бути або із загальною пам'яттю, або з розподіленою пам'яттю. Ця класифікація заснована на тому як MIMD-процесори отримують доступ до пам'яті. Цей клас припускає, що в обчислювальній системі є кілька пристроїв обробки команд, об'єднаних в єдиний комплекс і працюючих кожне зі своїм потоком команд і даних, і якимось чином синхронізуються між собою. Обробка розділена на кілька потоків, кожен з власним апаратним станом процесора, в рамках єдиного певного програмного забезпеченням процесу або в межах множинних процесів. Оскільки система має кілька потоків, які очікують виконання (системні або призначені для користувача потоки), ця архітектура ефективно використовує апаратні ресурси. (11, 12) Прикладом такої архітектури є обчислювальна система побудована за допомогою трансп’ютерної технології.

### Класифікація Хокни

Р. Хокні розробив свій підхід до класифікації для більш детальної систематизації комп'ютерів, що потрапляють в клас MIMD за системою М. Флінна. Намагаючись систематизувати архітектури усередині цього класу Хокні отримав ієрархічну структуру, представлену на рис. 1.4 (16).

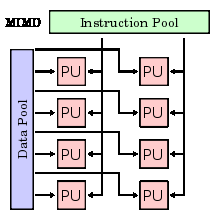


Рис.1.3. MIМD архітектура

### 

Рис. 1.4. Додаткова класифікація MIМD архітектури Р. Хокні

В класс конвеєрних архітектур (за Хокні) потрапляють машини з одним конвеєрним пристроєм обробки, що працює в режимі поділу часу для окремих потоків. Машини, в яких кожен потік обробляється своїм власним пристроєм Хокні назвав перемикаючими. В клас перемикаючих машин потрапляють машини, в яких можливий зв'язок кожного процесора з кожним, реалізований за допомогою перемикачів - машини з розподіленою пам'яттю. Якщо ж пам'ять є розділяється ресурс, машина називається із загальною пам'яттю. При розгляді машин з мережевою структурою Хокні вважав, що всі вони мають розподілену пам'ять. Подальшу класифікацію він проводив відповідно до топології мережі. (21)

### Класифікація Скіллікорна

У 1989 році була зроблена чергова спроба розширити класифікацію Флінна. Д. Скіллікорн розробив підхід придатний для опису властивостей багатопроцесорних систем і деяких нетрадиційних архітектур. Пропонується розглядати архітектуру будь-якого комп'ютера, як абстрактну структуру, що складається з чотирьох компонентів:

*процесор команд* (IP - Instruction Processor) - функціональний пристрій, що працює як інтерпретатор команд у системі, взагалі кажучи, може бути відсутнім;

*процесор даних* (DP - Data Processor) - функціональний пристрій, що працює як перетворювач даних відповідно з арифметичними операціями;

*ієрархія пам'яті* (IM - Instruction Memory, DM - Data Мemory) - запам'ятовуючий пристрій, в якому зберігаються дані та команди, що пересилаються між процесорами;

*перемикач* - абстрактний пристрій, що забезпечує зв'язок між процесорами і пам'яттю. Для опису складних паралельних обчислювальних систем Скіллікорн зафіксував чотири типи перемикачів без будь-якого зв'язку з типом пристроїв, які вони з'єднують:

1 - 1 - перемикач такого типу пов'язує пару функціональних пристроїв;

n - n - перемикач пов'язує кожен пристрій з множини пристроїв з відповідним йому пристроєм з іншої множини, тобто фіксує попарно зв'язок;

1 - n - перемикач з'єднує одне виділене пристрій з усіма функціональними пристроями з деякого набору;

n \* n - кожне функціональне влаштування одного безлічі може бути пов'язано з будь-яким пристроєм іншої множини, і навпаки. Класифікація Д. Скіллікорна будується на основі наступних восьми характеристик:

• кількість процесорів команд IP;

• число запам'ятовуючих пристроїв команд IM;

• тип перемикача між IP і IM;

• кількість процесорів даних DP;

• число запам'ятовуючих пристроїв даних DM;

• тип перемикача між DP і DM;

• тип перемикача між IP і DP;

• тип перемикача між DP і DP.

Використовуючи введені характеристики і припускаючи, що розгляд кількісних характеристик можна обмежити лише трьома можливими варіантами значень: 0, 1 і n (тобто більше одиниці), можна отримати різні класи архітектур. (16)

## Архітектура трансп’ютерів

Трансп’ютер (1985 р.) - це універсальний мікропроцесор з власною внутрішньою пам'яттю (кешем) досить великого обсягу, RISC-архітектури, що має в своєму складі незалежні двонаправлені коммунікаціоннне канали (лінки), здатні виконувати обмін даними з периферійними пристроями одночасно з обчисленнями в центральному процесорі .

Прилад спроектований таким чином, щоб максимально полегшити побудову паралельних обчислювальних систем. При з'єднанні транспьютерних елементів між собою потрібно мінімальне число додаткових інтегральних схем. Зв'язок між транспьютерами здійснюється шляхом безпосереднього з'єднання лінка одного приладу з лінком іншого. Це дозволяє створювати мережі з різними топологіями з великим числом елементів. Трансп’ютер мають високе відношення продуктивності до вартості. Правильне сучасне і досить широке тлумачення терміну "трансп’ютер" таке: це - універсальний мікропроцесор з вбудованими міжпроцесорних інтерфейсами. Трансп’ютер є апаратною підтримкою мови програмування Оккам. (1, 7)

### Загальна характеристика архітектури

Трансп’ютерна архітектура визначає сімейство програмованих компонентів. У самому простому вигляді трансп’ютер це пристрій який містить в собі процесор, чіп пам'яті і канали зв'язку для з'єднання між сусідніми трансп’ютерами.Также трансп’ютери можуть мати спеціальні інтерфейси і схеми адаптації до периферійних пристроїв.

Основне призначення транспьютеров це створення високопродуктивних паралельних систем. Транспьютер може бути запрограмований на будь-якій мові високого рівня, спеціально призначеного для забезпечення того, щоб скомпільовані програм були ефективні. Щоб отримати максимальну вигоду від трансп’ютерних систем, вся система може бути запрограмована за допомогою мови Оккам. Це дає всі переваги мови високого рівня, максимальну ефективність програмування і можливість використовувати особливості трансп’ютеров. (20) Внутрішня архітектура функціональних блоків транспьютера зображена на рис. 1.5. (3)

### Архітектура трансп’ютера Т800

Узагальнена структурна схема трансп’ютера Т800 наведена на рис. 1.6. На цьому ж рисунку показано і ВЗУ ємністю до 4 Гбайт, що входить в єдиний адресний простір. Залежно від моделі, трансп’ютер складається з 32

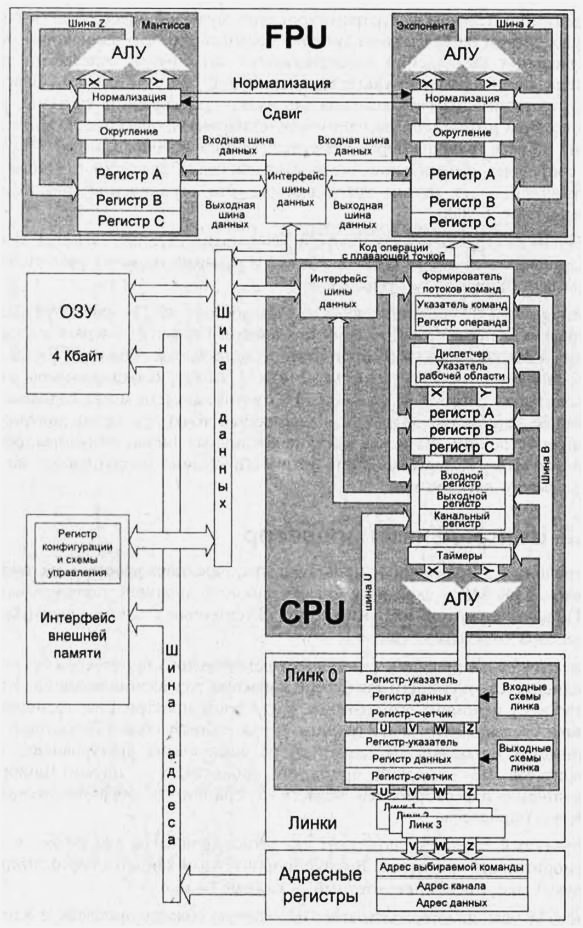


Рис. 1.5. Структура функціональних блоків транспьютера

або 16-розрядного центрального процесора, інтерфейсу зовнішньої пам'яті, 2 або 4-х двонаправлених каналів - лінків, програмованого блоку подій, таймера, внутрішнього ОЗУ ємністю 2 або 4 Кбайт, блоку режимів роботи лінків, блоку системних функцій. Деякі моделі можуть містити 64-розрядне пристрій операцій з плаваючою точкою (вбудований співпроцесор) та / або інтерфейсні схеми периферійних пристроїв, таких, як НЖМД, дисплей, мережевий адаптер. Контролери зовнішніх пристроїв зазвичай виконуються на кристалі замість двох лінків. (9)

**Регістри центрального процесора трансп’ютера**

Перерахуємо регістри ЦП трансп’ютера. Наведені нижче регістри доступні для користувача. Однак при їх використанні слід пам'ятати, що необережна зміна даних може призвести до втрати даних і збою в роботі програмного забезпечення.

А, В, С - регістри обчислювального стека (не тільки для арифметичних операцій);

О - 4-ох байтний регістр операнда;

W - покажчик робочої області процесу, в якій знаходяться його локальні дані, декларовані в описах програмного забезпечення і використовувані для проміжних результатів. Він використовується також для збереження адрес повернення і локальних змінних при виконанні викликів підпрограм, вказуючи на поточне значення стека аналогічно покажчику стека звичайного процесора;

I - покажчик наступної команди (лічильник команд);

Error - регістр помилки (однобітні прапор помилки);

HaltOnError - регістр зупинки помилково (однобітні прапор зупинки помилково);

FPtr0 - покажчик на початок черги процесів високого пріоритету;

BPtr0 - покажчик на кінець черги процесів високого пріоритету;

FPtr1 - покажчик на початок черги процесів низького пріоритету;

BPtr1 - покажчик на кінець черги процесів низького пріоритету;

Time0 - регістр стану таймера процесів високого пріоритету;

Time1 - регістр стану таймера процесів низького пріоритету;

NextTime0 - регістр таймерної черги процесів високого пріоритету;

NextTime1 - регістр таймерної черги процесів низького пріоритету;

Ще є пара регістрів, які визначають стан таймерних черг для процесів високого і низького пріоритетів (NextTime0 і NextTime1). Для трансп’ютеров с ППТ регистры FA, FB, FC образуют вычислительный стек с плавающей точкой.



Рис. 1.6. Узагальнена структурна схема транспьютеров типу Т800

**Регістри стеку**

Розглянемо регістри обчислювального стека.

Для 32-х розрядного трансп’ютера Т800 принцип функціонування регістрів А, В, С обчислювального стека може бути пояснений наступної ілюстрацією послідовного виконання 2-х операцій: c = a + b і k = c +1 на рис. 1.7.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | c=a+b |  |  |  | k=c+1 |  |
| A | b |  | c |  | 1 |  | k |  |
| B | a | 🡪 | b |  | c | 🡪 | 1 |  |
| C | - |  | a |  | b |  | c |  |
|  | **До** |  | **После** |  | **До** |  | **После** | **- выполнение операций** |

Рис. 1.7. Стан регістрів А, В, С обчислювального стека

Недоліком такого обчислювального стека є відсутність апаратної підтримки переповнення стека. У разі ППТ обчислювальний стек (регістри FA, FB, FC) побудований аналогічно, але з ЦП у ППТ передаються лише покажчики операндів, а вгорі стека буде покажчик на результат.

### Організація пам'яті трансп’ютера

Розглянемо використання пам'яті трансп'ютера за допомогою таблиці 1.3

Таблиця 1.3 Організація пам’яті трансп’ютера

|  |  |  |
| --- | --- | --- |
| **Регістр / сигнал** | **Адрес байта** | **Призначення / коментар** |
|  | 7FFFFFFF | Кінець зовнішньої пам'яті. / Найбільший позитивний адресу. |
| Reset | 7FFFFFFE | Запуск програми ПЗУ. / У 2-х старших байтах повинен бути записаний код команди передачі керування на програму ПЗУ. |
|  | 7FFFFFF8 | Адреса програми ПЗУ. |
|  | 7FFFFF6C | Код конфігурації пам’яті. |
|  | 00000000 | Нульова адреса. / Показує, що адресний простір позитивне і негативне. |
|  | 80001000 | Початок зовнішньої пам’яті трансп’ютера Т800. |
|  | 80000FFF | Кінець внутрьошньої накристальньої пам’яті трансп’ютера Т800. / Об’єм пам’яті - 4Кб. |
|  | 80000800 | Початок зовнішньої пам’яті трансп’ютера Т414. |
|  | 800007FF | Кінець внутрьошньої накристальньої пам’яті трансп’ютера Т800. / Об’єм пам’яті - 2Кб. |
| MemStart | 80000070 | Початок пам’яті трансп’ютера Т800, доступної користувачеві. |
| MemStart | 80000048 | Початок пам’яті трансп’ютера Т414, доступної користувачеві. |
| ERegIntSave | 80000044 | Використовується в додаткових функціях при перериванні поточного процесу низького рівня в якості додаткового регістра ЦП. Використовується також при виконанні команди переміщення блоку даних і в операціях плаваючою точки. Вміст цього регістра і наступних шести на повинні змінюватися користувачем. |
| StatusIntSaveLoc | 80000040 | Збереження регістра стану прапора помилки. |
| CRegIntSaveLoc | 8000003C | Збереження регістра стану C. |
| BRegIntSaveLog | 80000038 | Збереження регістра стану B. |
| ARegIntSaveLog | 80000034 | Збереження регістра стану A. |
| IRegIntSaveLog | 80000030 | Збереження регістра стану I. |
| WRegIntSaveLog | 8000002C | Збереження регістра стану W. |
| TPtrLoc1 | 80000028 | Регістр таймера, в якому міститься покажчик на перший процес у черзі таймера процесів низького пріоритету або ознака не використання таймера. |
| TPtrLoc0 | 80000024 | Регістр таймера, в якому міститься покажчик на перший процес у черзі таймера процесів високого пріоритету або ознака не використання таймера. |
| Event | 80000020 | Регістр каналу зовнішнього події дозволяє активізувати канал для передачі даних при виникненні зовнішнього переривання, може бути використаний для синхронізації початку виконання процесу на ряді трансп’ютеров. |
| Link3Input | 8000001C | Вхідний канал лінка 3. |
| Link2Input | 80000018 | Вхідний канал лінка 2. |
| Link1Input | 80000014 | Вхідний канал лінка 1. |
| Link0Input | 80000010 | Вхідний канал лінка 0. |
| Link3Output | 8000000C | Вихідний канал лінка 3. |
| Link2Output | 80000008 | Вихідний канал лінка 2. |
| Link1Output | 80000004 | Вихідний канал лінка 1. |
| Link0Output | 80000000 | Вихідний канал лінка 0. |
|  | 80000000 | Початок внутрішньої накрестальной пам'яті. |

Звернення до внутрішньої пам'яті трансп’ютера виконується за один цикл процесора. Адресація пам'яті здійснюється наступним чином: всі слова вирівнюються по 4-х байтовой кордоні повного слова для 32-х розрядних трансп’ютерів, і по 2-х байтовой кордоні повного слова для 26-ти розрядних трансп’ютерів. Усередині кожного слова адресація кожного байта здійснюється додаванням його селектора до адреси слова. Через контролер зовнішнього ОЗУ можна підключити до 4Гб пам'яті, або один зовнішній пристрій з паралельним інтерфейсом.

Адресний простір є наскрізним і циклічним, і при роботі з усім спектром адрес можна користуватися одними і тими ж командами трансп’ютера.

Слід пам'ятати, що циклічність адресного простору може приводити до помилок при програмуванні доступу до пам'яті трансп’ютера. При цьому можна мимоволі "перескочити" з області позитивного адресного простору в негативне і навпаки. При попаданні в негативне адресний простір може бути випадково зіпсований системна область у накрістальной пам'яті, що може з'явитися причиною збою в роботі програмного забезпечення.

### Процесор плаваючої точки

Основний процесор і ППТ можуть працювати паралельно. Потік команд можна розглядати у вигляді потоку команд послідовно виконується на одному процесорі для цілої і плаваючої арифметики. При цьому обидва процесора можуть розглядатися у вигляді оккамовскіх процесів, які обмінюються повідомленнями між собою по каналах. Механізм передачі повідомлень може бути використаний для збільшення часу одночасної роботи двох процесорів. Існує два класи команд з плаваючою арифметикою:

• команди виконуються виключно на ППТ і не змінюють стану основного процесора (наприклад fpadd)

• команди, які можуть переміщати дані між ППТ і основним процесором.

Команда плаваючої арифметики вимагає для виконання більше часу, ніж команда целочисленної арифметики, виконуваної ЦП. Тому за час виконання команди плаваючої арифметики може бути виконано декілька команд ЦП. При цьому можливі 2 ситуації: або ЦП очікує результат виконання команди в ППТ і в цьому випадку ЦП простоює, або ЦП не потрібно отримання результату виконання команди в ППТ і він виконує команду далі.

ППТ реалізований на зсувних регістрах і працює з достатньою продуктивністю, не затримуючи роботу ЦП.

## Процеси в трансп’ютерах та апаратний планувальник

Всі процеси можуть знаходитися тільки в двох станах – в активному або пасивному стані. Активним називається процес, який виконується в ЦП або перебуває в черзі активних процесів. Активний процес готовий до виконання. Процес в пасивному стані чекає або введення даного, або виведення даного, або настання деякої події по таймеру (моменту старту). Пасивний процес виводиться апаратним планувальником з розкладу черги активних процесів, не займає більше ресурсів ЦП і дає можливість іншим процесам високого або низького пріоритету виконуватися з метою їх своєчасного старту по таймеру, виконання з черги, обміну з ним даними.

### Пріоритет процесу

Планувальник процесів - це мікрокод диспетчера процесів, програмно записаний на кристалі трансп’ютера. Він дозволяє на основі 2-х пріоритетних рівнів процесів - високого і низького, а також квантування за часом процесів низького рівня, реалізовувати псевдопаралельною режим одночасної роботи великої кількості процесів на одному трансп’ютері.

Процеси бувають високого або низького пріоритету. Значення пріоритету процесу високого рівня дорівнює 0, значення пріоритету процесу низького рівня дорівнює 1. Процеси обох рівнів можуть знаходитися як в активному, так і в пасивному станах.

Процес високого пріоритету є терміновим процесом і повинен виконуватися негайно, як тільки стане готовим до виконання. При цьому він переходить в активний стан і починає виконуватися в ЦП трансп’ютера, а в разі виконання в ЦП іншого процесу з високим пріоритетом він записується в кінець черги активних процесів високого пріоритету на виконання за допомогою регістра BPtr0. Процеси високого пріоритету виконуються до повного завершення або переходу в пасивний стан, в якому дескриптор (вміст регістру W) процесу зберігається в слові стану каналу (ССК) або таймерної черги. Вони повинні виконуватися за короткі інтервали часу і давати можливість виконуватися процесам низького пріоритету, що становить більшу частину програмного забезпечення.

Процес низького пріоритету може безперервно виконуватися тільки протягом одного кванта часу тривалістю 2048 мкс. За цей проміжок часу з ним може статися одна з чотирьох подій:

1) процес завершить своє виконання до кінця і припинить своє існування, при цьому кількість активних процесів зменшується на один;

2) процес не виконається до кінця і залишиться в активному стані, при цьому він буде додано в кінець черги активних процесів, кількість яких не зміниться;

3) процес перерветься і перейде на час в пасивний стан очікування даних або відкладеного часу старту, не буде доданий в активну чергу, а потім відновить своє виконання в перерваному місці при обміні даними або по таймеру;

4) процес перерветься процесом високого пріоритету і перейде на час його виконання у пасивний стан, не буде доданий в активну чергу, а потім відновить своє виконання з перерваного місця з колишнім станом на ЦП.

При відкладеному старті процесу його дескриптор зберігається в таймерної черги. При перериванні процесом високого пріоритету процесу низького пріоритету вміст регістрів A, B, C, W, I, Error зберігається в 6-ти словах на початку накрістальной пам'яті транспьютера. Після виконання всіх активних процесів високого пріоритету апаратний планувальник відновить вміст 6-ти регістрів і продовжить виконання перерваного процесу низького пріоритету без зміни його стану, що міститься в його службовій області.

## Комунікаційні можливості трансп’ютерів

Кожен трансп’ютер має 4 або 2 линка, відповідно на кристалі трансп’ютера знаходяться 4 або 2 інтерфейси лінків. Лінки є зовнішніми комунікаційними каналами зв'язку, які з'єднують трансп’ютер шляхом простого з'єднання за допомогою провідників між собою. Існують фізичні обмеження на здійснення обміну даними через лінки:

1. Лінк одного трансп’ютер з'єднується з одним лінком іншого транспьютера;

2. До лінку одного трансп’ютера можна під'єднати тільки один лінк іншого;

Кожен лінк має тільки два канали, які фізично реалізуються двома проводами. Один канал передає дані в один бік, інший - в іншу.

Один лінк може одночасно обслуговувати два Occam-процесу, це пов'язано з особливостями протоколу паралельного інтерфейсу. Інтерфейси лінків коректно передають дані на відстань до 300 мм. Лінки з'єднуються двома витими парами проводів, в кожній з яких один провід - сигнальний, а інший - земляний. Для зменшення рівня шуму кожну пару екранізують.

При обміні на великі відстані може використовуватися кабельна лінія на 100 Ом і підключеним согласующим опором (на відстань 6-10 метрів). Для передачі на великі відстані повинні застосовуватися буферні пристрої. На відстань понад 100 м може застосовуватися волоконно-оптичний кабель. Швидкість передачі даних при цьому знижується до 5 Мбіт / сек. Лінки забезпечують передачу даних з трьома фіксованими швидкостями: 5, 10, 20 Мбіт / сек (у трансп’ютера Т9000 ця швидкість дорівнює ~ 150 Мбіт / сек.). Швидкість передачі даних по лінкам встановлюється за допомогою системного пристрою, шляхом передачі на його контакти комбінацій з високого і низького сигналу. Передача повідомлень по лінку не залежить від фази сигналу таймера. Адреси зовнішніх каналів введення і виведення інтерфейсу линка можна побачити в таблиці 1.3.

## Системи на основі трансп’ютерів

На прикладі трансп’ютерів можна показувати характеристики і особливості різних сполук і структур локальних обчислювальних паралельних систем.

### Лінійний ланцюжок



Рис. 1.8. Структура ланцюжок

1) максимально можливу відстань може бути велике при великій кількості трансп’ютерів.

2) простий алгоритм маршрутизації повідомлень, але кожен трансп’ютер бере участь у маршрутизації повідомлень

3) просте розширення мережі шляхом додавання трансп’ютерів ліворуч або праворуч

4) передача великих обсягів даних утрудняє роботу мережі через необхідність транзитної передачі даних через проміжні трансп’ютери.

### Кільце

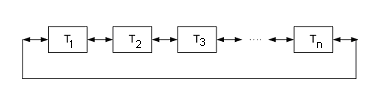


Рис. 1.9. Структура кільце

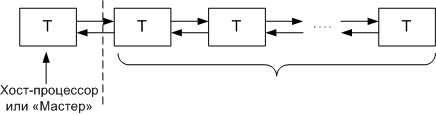
1) максимально можливу відстань може бути великим, але в 2 рази менше, ніж в аналогічної лінійного ланцюжка.

2) простий алгоритм маршрутизації повідомлень, кожен трансп’ютер бере участь у маршрутизації повідомлень

3) просте розширення мережі, трансп’ютери можна додати в будь-яке місце ланцюжка

4) неефективно при передачі великих обсягів даних, однак за рахунок з'єднання 1-го та останнього процесора між собою можна передавати дані в обох напрямках.

### Ферма

**

«Работники»

Рис. 1.10. Структура ферма

1) максимально можлива велика відстань.

2) простий алгоритм маршрутизації повідомлень, кожен трансп’ютер бере участь у маршрутизації повідомлень

3) просте розширення мережі

4) ускладнена передача великих обсягів даних.

На відміну від лінійних структур або кільцевих ирансп’ютери в такій системі не є рівноправними - один з них є головним і називається хост - процесор або «майстер», а всі інші є «працівниками».

Функції трансп’ютерів: «майстер» розподіляє і контролює роботу інших трансп’ютерів, збирає результати. Працівники виконують тільки обчислювальну роботу і повідомляють «майстра» про свій стан.

Ефективність лінійної ферми може бути підвищена при використанні деревовидних структур.

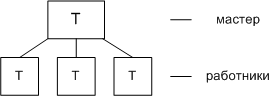
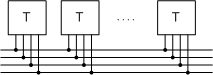


Рис. 1.11. Ферма з використанням структури дерева

### Багатомагістральна



линковая

магистраль

Рис. 1.12. Багатомагістральна структура

1) максимально можливу відстань дорівнює нулю.

2) маршрутизація повідомлень не викликає проблем

3) легко додати додатковий трансп’ютер.

4) передача великих обсягів даних обмежується пропускною здатністю загальної магістралі передачі даних.

### Гіпердерево

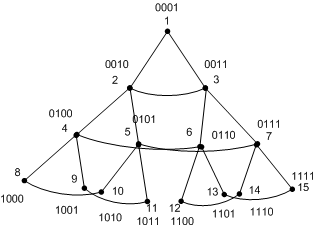


Рисунок 1.13 Гіпердерево

Пронумеруємо вершини і кожній вершині поставимо у відповідність 4-х бітовий двійковий код. В основі гіпердерева лежить бінарне дерево, у якого процесори з адресами, що відрізняються на 1 бінарний розряд, попарно з'єднані горизонтальними зв'язками.

Характеристики:

1) гіпердерево зменшує відстань між елементами бінаронго дерева. Максимально можливу відстань велика, але воно менше, ніж у бінарному.

2) можливо використання достатньо простого алгоритму маршрутизації повідомлень.

3) можливе розширення мережі без порушення роботи вже існуючої частини.

4) можливість обробки великих масивів даних. Вузьким місцем з передачі даних є кореневий вузол, а також вузли високого рівня, пропускна спроможність яких більше, ніж у структурі бінарного дерева.

## Огляд мови Оккам

Мова програмування Occam була розроблена компанією INMOS на основі теорії Ентоні Хоара (англ. CAR Hoare) про взаємодію процесів і був призначений для програмування. (14) Occam є алголо-подібною мовою високого рівня, при цьому мова оптимізована з точки зору ефективності її трансляції в систему команд транспьютера. (18)

Спочатку INMOS навіть пропонувала сприймати Occam в якості трансп’ютерного асемблера, але пізніше випустила пакет низькорівневих засобів для розробників компіляторів.

### Процеси в мові Оккам

Оккам-програма це процес. У основі програмування трансп’ютерних систем лежить поняття процесу як деякої сукупності дій, які виконує обчислювальна машина. Кожен процес може бути або процесом-примітивом, або процесом-композицією. Процес-композиція має деревоподібну структуру, вузли якої відповідає підпроцесу-композиціям, а листя - процесам-примітивам. Вузол дерева, який не є підпроцесів, тобто складовою частиною деякого підпроцесу-композиції більш високого рівня, називається кореневим процесом. Кожен наступний підрівень записується з відступом на два пропуски більше ніж попередній підрівень.

Всього існує 5 основних процесів-примітивів.

- Присваювання

- Введення

- Виведення

- Пропуск

- Стоп.

Під час присваювання відбувається установка значень списку змінних відповідним значенням з іншого списку. Наприклад, в x, y, z: = 1,3,5 після виконання x встановиться в 1, y в 3, а z у 5. Кількість змінних у лівій частині має відповідати кількості значень у правій частині. Для прийому та передачі значення з каналу в змінну використовується процес-примітив введення, де символ «?» позначає введення з каналу значення для подальшого зберігання отриманого значення у змінну, а символ «!» позначає виведення з змінної в канал передачі. Наприклад, chan ! 5.

Процес Пропуск (SKIP) є аналогом елементарної операції NOP. А процес Стоп (STOP) має ті ж властивості, що й SKIP, але з однією відмінністю. Процес STOP збуджує виняткову ситуацію, яка викликає передчасне закінчення кореневого процесу.

Процесів-композицій всього 6 основних і 3 додаткові.

**SEQ** - конструктор послідовного виконання процесів (підпроцесів). Всі підпроцеси цього процесу будуть виконуватися послідовно один за іншим як і було записано. SEQ без підпроцесів буде виконуватися також як і процес SKIP.

**PAR** - конструктор паралельного виконання процесів (подпорцессов). Всі підпроцеси цього процесу будуть виконуватися паралельно, незалежно один від одного.

**WHILE** - конструктор циклу виконання процесів (підпроцесів).

**IF** - конструктор умовного виконання процесів (підпроцесів).

**CASE** - конструктор вибору виконання процесів (підпроцесів).

**ALT** - конструктор альтернативного виконання процесів (підпроцесів).

Додаткові конструктори:

**FOR** - конструктор реплікатор виконання процесів (підпроцесів).

**PRI** - конструктор префікс прямого виконання процесів (підпроцесів).

**PROC** - являє собою цілу групу процесів, що реалізують цю процедуру. Все робиться так само, як і в основній програмі. Різновидом процедури є функція, що вводиться ключовим словом FUNCTION. (4, 10, 13)

### Взаємодія процесів

Мова Оккам пропонує просту і однорідну структуру механізму передач повідомлення, яке в чомусь схоже з механізмом рандеву в Аді. Механізм заснований на передачі повідомленнь реалізується за допомогою процесу ALT. Процес ALT складається з альтернативного процесу та процесу-сторожа. Процес-сторож може бути або процесом-введенням або процесом SKIP. Процес-сторож може мати логічні вираження розділені знаками амперсанда (&). Сторож готовий тоді і тільки тоді, коли в паралельно перебіг процесу стає можливим висновок в канал з відповідному ім'ям в логічному вираженні сторожа або якщо в логічному вираженні стоїть значення TRUE. Як тільки сторож готовий, виконується його альтернативний процес. Сторож SKIP готовий завжди. Якщо в списку процесів-сторож немає сторожа SKIP тоді при досягненні цієї ділянки виникає deadlock, і процес блокується до тих пір, поки один із сторожів не стане готовий. (2, 15)

Приклад програми взаємодії процесів.

SEQ --послідовний процесс

PAR --паралельний процесс (паралельно реєструємо типи

- каналів передачі даних)

CHAN OF INT Output1, Output2; --канали передачі даних

CHAN OF BOOL Input1, Input2; --каналы прийому даних

INT rez, a, b; -- оголошення змінних цілого типу

BOOL Data; -- оголошення змінних булевого типу

a, b := 8, 15;

rez := a – b; -- rez = -7

ALT -- альтернативний процесор (синхронізація завдань)

Input1 ? Data --введення з каналу зв'язку з транспֹютером 1

PAR -- якщо він активується, то передамо 1

-- трансп’ютеру дані і т.д.

Output1 ! REZ

Input2 ? Data

PAR

Output2 ! REZ

## Висновки до Розділу 1

1. Розглянуто мікропроцесорні пристрої – трансп’ютери, що є реалізацією MIMD архітектури. Показано, що вони є зручним інструментарем для програмно-апаратної реалізації комп'ютерних систем та мереж з локальною пам'яттю.

2. Розглянуто архітектуру трансп’ютера Т800. Показано, що трансп’ютери спроектовані таким чином, що їх апаратних можливостей достатньо для зберігань даних, обчислень і синхронізації всередині системи.

3. Процес буває в двох станах - в пасивному або активному стані. У трансп’ютері реалізована бінарна пріоритетність процесів. Процес може бути високого пріоритету і низького. Також у трансп’ютері присутній апаратний планіровщик процесів, який реалізує механізм ефективного планування виконання черги процесів.

4. Трансп’ютер можна з'єднати з іншими трансп’ютерами, утворюючи єдину мережу за допомогою з'єднання через вбудовані інтерфейси - лінки.

5. На прикладі комбінацій з'єднання трансп’ютерів показані особливості різних структур паралельних обчислювальних систем з локальною пам'яттю, серед яких зустрічаються такі структури як: лінійка, кільце, багатомагістраль, гіпердерево, гіперкуб і пр.

6. Розглянуто мову програмування Оккам, спроектовану для програмування трансп’ютерних систем. Програмування на мові Оккам відбувається за допомогою оперування найпростішими процесами, такими як присвоєння, введення та інше, які комбінуються у процеси більш високого рівня.

7. У мові Оккам реалізований механізм синхронізації процесів з допомогою рандеву, яке відрізняється від механізму рандеву Ади структурою і реалізований в ALT - процесі.

# РОЗДІЛ 2. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ПАРАЛЕЛЬНИХ КОМП’ЮТЕРНИХ СИСТЕМ

## 2.1 Розробка паралельного математичного алгоритму

При розробці паралельного алгоритму передбачається, що розмірність задачі (N) більше або дорівнює числу процесорів (P) (N> = P), причому N кратно P, що дозволяє розбити задачу на окремі шматочки Н, кожен з яких обробляється своїм завданням. .

Розрахунок даного матричного рівняння можливо провести у один етап:

## Аналіз завдання

### Оцінка розробленого паралельного алгоритму в рамках концепції необмеженого паралелізму

Оцінити паралельний алгоритм зручніше всього двома етапами: розглядом обчислення виразу в дужках і загального виразу.

На рис. 2.1 зображена ​​ярусно-паралельна форма алгоритму обчислення одного елемента матричного рівняння МТ = МО \* МХ - МЕ.

Рис. 2.1. Ярусно-паралельна форма обчислення матриці МТ = MO \* MX - ME

Кількість ярусів для обчислення одного елемента матриці рівне log2N+2. Всю матрицю обчислювати паралельно неможливо. Для подальшого обчислення потрібно сформувати стовпець матриці. Кількість ярусів у даному випадку буде дорівнює N (log2N + 2).

На рис. 2.2 зображена ярусно-паралельна форма алгоритму обчислення одного елемента матрично-векторного рівняння А = В + С \* МТ з урахуванням ярусів обчислення стовпця МТ.

Рис. 2.2. Ярусно-параллельная форма вычисления вектора А = B + C \* MT

Як видно, для формування N елементів досить N(log2N+2)+N2(log2N+2) ярусів. Вважаємо, що всі операції виконуються за одиницю часу, тоді має місце наступну рівність кількості ярусів і кількості операцій

log2N + 1 = 2N - 1. Таким чином, використовуючи відому кількість ярусів можна дізнатися основні параметри та характеристики даного матричного рівняння.

**T1=2N3+2N2** - час обчислення завдання на 1-процесорної машині.

**TN= N(log2N + 2) + N2(log2N + 2)** - час обчислення завдання на машині з кількістю процесорів N.

**КП =** (2N3+2N2)/(N(log2N + 2) + N2(log2N + 2)) =

= 2N2 (N+1)/N(N+1)(log2N + 2) **= 2N/(log2N + 2)** – коефіцієнт прискорення

**КЕ** = **2/(log2N + 2)** – коефіцієнт еффективності.

Проаналізувавши ці вирази, робимо висновок, що при збільшенні кількості процесорів КП зростає, проте КЕ зменшується.

Оцінимо час виконання завдання.

Згідно з концепцією необмеженого паралелізму час на синхронізацію не враховується. Вважаємо, що всі операції виконуються за одиницю часу. Заданий математичне вираз на однопроцесорній обчислювальної системі буде вважатися за час **T1=2N3+2N2**.

При P = р **Tp = log2p+(2N3+2N2+1-2]log2p[)/p**

При P => ∞ **Tp = log2(2N3+2N2+1)**

Провівши дослідження аналітичним способом ми отримали результати досліджень, на підставі яких можна зробити висновок про те, що при використанні програми розробленої для паралельної обчислювальної системи з Р процесорами ми отримуємо значну перевагу в швидкості обчислення.

## Розробка ПЗ для системи з спільною пам’яттю

### Розробка алгоритмів процесів

|  |  |  |
| --- | --- | --- |
| **№** | **Алгоритм задачі №1** | **ТС/КД** |
| 1. | Введення В, MО |  |
| 2. | Сигнал усім задачам про введення В та MО | S\*-1 |
| 3. | Чекати завершення введення у задачі №P | WP-1 |
| 4. | Копіювання MО, С | КД |
| 5. | Розрахунок AH = BH + С \* (MО \* MХH – MЕH) |  |
| 6. | Чекати закінчення розрахунку AH в усіх задачах | W\*-1 |
| 7. | Виведення А |  |
|  |  |  |
| **№** | **Алгоритм задачі №P** | **ТС/КУ** |
| 1. | Введення С, MХ, MЕ |  |
| 2. | Сигнал усім задачам про введення С, MХ, MЕ | S\*-1 |
| 3. | Чекати завершення введення у задачі №1 | W1-1 |
| 4. | Копіювання MО, С | КД |
| 5. | Розрахунок AH = BH + С \* (MО \* MХH – MЕH) |  |
| 6. | Сигнал задачі №1 про завершення розрахунку | S1-2 |
|  |  |  |
| **№** | **Алгоритм задачі №2 - (P-1)** | **ТС/КУ** |
| 1. | Чекати завершення введення у задачах №1 та №P | W1-1, WP-1 |
| 2. | Копіювання MО, С | КД |
| 3. | Розрахунок AH = BH + С \* (MО \* MХH – MЕH) |  |
| 4. | Сигнал задачі №1 про завершення розрахунку | S1-1 |

## Розробка схеми взаємодії задач



Рис. 2.3. Схема взаємодії задач для системи зі спільною пам’яттю

### Розробка програми

Для реалізації алгоритму обчислення матричного рівняння на обчислювальній системі зі спільною пам’яттю було використано мову Java. Ця мова має потужний апарат для забезпечення паралельної обробки.

Для вирішення проблем синхронізації та доступу до спільного ресурсу було створено клас, що містить синхронізовані (synchronized) методи. У один момент часу такий метод може бути використаний тільки одним потоком.

Лістинг програми наведено у Додатку Ж.

## Розробка ПЗ для системи з локальною пам’яттю

### Розробка алгоритмів процесів

|  |  |
| --- | --- |
| **№** | **Алгоритм задач з rank < P/2 (задачі що зверху)** |
| 1. | Якщо rank = P/2-1, то ввести MХ, МЕ, С |
| 2. | Якщо rank < P/2-1, то отримати MХXH, МЕXH, С від правої задачі rank + 1 |
| 3. | Якщо rank > 0, то передати MХ(X-1)H, МЕ(X-1)H, С задачі rank - 1 зліва |
| 4. | Якщо rank > 0, то отримати BXH, МO від задачі rank - 1 зліва |
| 5. | Якщо rank < P/2-1, то передати B(X-1)H, МO правій задачі rank + 1 |
| 6. | Передати MХH, МЕH, BXH, МO, С задачі rank + P/2 знизу |
| 7. | Розрахунок AH = BH + С \* (MО \* MХH – MЕH) |
| 8. | Отримати AH від задачі знизу rank + P/2 |
| 9. | Отримати A(X-1)H від правої задачі rank + 1 |
| 10. | Якщо id = 1, то вивести А, інакше передати AH задачі rank - 1 зліва |
|  |  |
| **№** | **Алгоритм задач з id >= P/2 (задачі що знизу)** |
| 1. | Отримати MХH, МЕH, BXH, МO, С від задачі rank - P/2, що зверху |
| 2. | Розрахунок AH = BH + С \* (MО \* MХH – MЕH) |
| 3. | Передати AH задачі rank - P/2, що зверху |

### Розробка програми

Для реалізації алгоритму обчислення матричного рівняння на обчислювальній системі з локальною пам’яттю було використано мову Java та реалізацію технології MPI (бібліотека MPJ (19)) для цієї мови. MPI є найпоширенішою системою для розробки паралельних систем, що використовують підхід пересилки повідомлень.

Програма являє собою універсальну заготовку, алгоритм роботи якої визначається в залежності від виданого номеру (rank) процессу.

У реалізації широко використовуються функції MPI: MPI\_Send та MPI\_Recv.

Лістинг програми наведено у Додатку З.

### Розробка схеми взаємодії задач

Схема взаємодії задач показано на рис.2.4. На рисунку: H – кусочок матриці або вектора, а NH – N кусочків матриці або вектора.



## Висновки до Розділу 2

1. Мова Java має дуже зручні інструменти організації паралельних обчислень. Прозорий механізм синхронізованих методів, а також об’єктно-орієнтований підхід значно спрощують реалізацію кінцевої програми. Додатково Java надає зручні методи роботи з двовимірними масивами і позбавляє клопоту слідкування за «витоками» пам’яті.

2. При написанні програми для системи з локальною пам’яттю велика вірогідність зробити помилку у механізмі передачі повідомлень. Такі помилки можливо виявити тільки на етапі часу виконування, але так як у суті роботи MPI стоїть створення багатьох процесів, налагодження такої програми є досить проблематичним.

3. Порівнюючи процес розробки для системи зі спільною пам’яттю та системи з локальною пам’яттю можемо зробити висновок, що варіант зі спільною пам’яттю є більш простим у розробці та реалізації. Це зумовлено у першу чергу більш простим механізмом взаємодії потоків.

# РОЗДІЛ 3. ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

**3.1 Методика тестування.**

Для замірів часу виконання розробленного програмного забезпечення використовувались функції зі стандартних бібліотек мов, що були використані (бібліотека java.util.Date). Програмне забезпечення приймало різне значення обсягів даних та запускалось на різних кількостях ядер.

Для завдання певної кількості ядер, що використовуються програмою було використано програму PsExec з комплекту программ PsTools. Приклад використання цієї програми для запуску процессу що використовує 2 ядра процессора:

*Java*: psexec -a 1,2 java -Xmx1024M -jar JavaCoursPRO.jar 1000 2

*MPI:* psexec -a 1,2 java -jar D:/mpj-v0\_36/lib/starter.jar -np 2 -Xmx1024M MPICoursPRO.jar 1000 2

Основні характеристики розробленного програмного забезпечення визначаются за допомогою формул.

Коефіцієнт прискорення (5):

Коефіцієнт ефективності (5):

Тестування проводилося на системі з наступними характеристиками:

**Центральний процесор**: Intel Core 2 Quad Q6600, 2.41 GHz

**Оперативна пам’ять:** 1.99 Gb

**Операційна система**: Microsoft Windows Proffesional Version 2002 SP3

**Версії програмного забезпечення:**

*Java:* JVM Version 6 update 20 (build 1.6.0\_17)

*MPI:* MPJ Express Version 0.36

*Керування режимом роботи процесора:* PsTools PsExec v1.94

## 3.2 Тестування ПЗ для системи з спільною пам’яттю

Таблиця 3.1. Час виконання програми обчислення на системі зі СП

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N P | 1 | 2 | 3 | 4 |
| 900 | 7078 | 3600 | 2400 | 1344 |
| 1800 | 75234 | 40313 | 26578 | 19875 |
| 2700 | 235125 | 132437 | 92453 | 65921 |
| 3600 | 664515 | 336859 | 243422 | 183532 |

Таблиця 3.2. Значення коефіцієнтів прискорення для системи с СП

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | P | | | |
| 1 | 2 | 3 | 4 |
| 900 | 1 | 1,961111 | 2,94916 | 3,838394 |
| 1800 | 1 | 1,866246 | 2,83068 | 3,785358 |
| 2700 | 1 | 1,775372 | 2,54318 | 3,56676 |
| 3600 | 1 | 1,972679 | 2,72988 | 3,620703 |

Таблиця 3.3 Значення коефіцієнтів ефективності для системи с СП

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | P | | | |
| 1 | 2 | 3 | 4 |
| 900 | 1 | 0,98055 | 0,96305 | 0,95959 |
| 1800 | 1 | 0,94812 | 0,94356 | 0,94244 |
| 2700 | 1 | 0,87768 | 0,84772 | 0,89169 |
| 3600 | 1 | 0,95633 | 0,90996 | 0,90517 |

**Рис. 3.1. Залежність коефіцієнту прискорення від кількості ЦП для систем з спільною пам’яттю

****

Рис. 3.2**.** Залежність коефіцієнту ефективності від кількості ЦП для систем з спільною пам’яттю

## Тестування ПЗ для системи з локальною пам’яттю

Таблиця 3.4. Час виконання програми обчислення на системі зі ЛП

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N P | 1 | 2 | 3 | 4 |
| 900 | 7203 | 3656 | 2484 | 2047 |
| 1800 | 57093 | 28781 | 20047 | 16016 |
| 2700 | 192218 | 97000 | 66297 | 53015 |
| 3600 | 456406 | 228208 | 167531 | 125640 |

Таблиця 3.5. Значення коефіцієнтів прискорення для системи с ЛП

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| NN | P | | | |
| 1 | 2 | 3 | 4 |
| 900 | 1 | 1,970185 | 2,88975 | 3,51880 |
| 1800 | 1 | 1,983704 | 2,84795 | 3,56474 |
| 2700 | 1 | 1,981628 | 2,89934 | 3,62572 |
| 3600 | 1 | 1,999956 | 2,72430 | 3,63264 |

Таблиця 3.6. Значення коефіцієнтів ефективності для системи с СП

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N | P | | | |
| 1 | 2 | 3 | 4 |
| 900 | 1 | 0,985092 | 0,96325 | 0,87970 |
| 1800 | 1 | 0,991852 | 0,94931 | 0,89118 |
| 2700 | 1 | 0,990814 | 0,96644 | 0,90643 |
| 3600 | 1 | 0,978478 | 0,92810 | 0,90816 |

****

Рис. 3.3. Залежність коефіцієнту прискорення від кількості ЦП для систем з локальною пам’яттю

****

Рис. 3.4. Залежність коефіцієнту ефективності від кількості ЦП для систем з локальною пам’яттю

## Висновки до Розділу 3

1. В системі зі спільною пам’яттю коефіцієнт прискорення лінійно зростає зі збільшенням кількості ядер, однак чим більше процесорів, тим більший розкид прискорення між різними обсягами пам'яті. На рис. 3.1 зображен графік залежності коефіцієнта прискорення від кількості ядер, що показує лінійність коефіцієнта прискорення і розкид прискорення. Коефіцієнт ефективності в даній системі спадає зі збільшенням кількості ядер. На рис. 3.2 видно, що з кожним наступним додаванням ядер, коефіцієнт ефективності спадає менш інтенсивно. Тобто із збільшенням ядер зменшується інтенсивність спадання ефективності, що означає, що для якогось критичного кількістю ядер коефіцієнт досягне деякого критичного коефіцієнта ефективності, і нижче нього спадати вже не буде. Це говорить про гарний потенціал масштабованості програмного забезпечення.

2. Коефіцієнт прискорення системи з локальною пам’яттю лінійно зростає зі збільшенням кількості процесорів. Як видно на рис. 3.3, при 2 ядрах коефіцієнт прискорення практично дорівнює один одному при різних обсягах даних, це пояснюється тим, що ядра між собою з'єднані попарно (8), і дані між ними передаються досить швидко, тоді як між парами ядер інформація передається повільніше. Це видно при збільшенні кількості ядер, відбувається розкид коефіцієнта прискорення який залежить від кількості переданих даних. Коефіцієнт ефективності одно пропорційний коефіцієнт прискорення, це видно на рисунку 3.4. Також на цьому рисунку видно, що чим більше ядер використовується, тим більше падає коефіцієнт ефективності. Це пояснюється тим, що ядра неефективно використовують час виконання завдання, багато часу виконання йде на вирішення завдань синхронізації і взаємного виключення.

3. Порівнюючи програмне забезпечення для обох систем, можна зазначити те, що програмне забезпечення для локальної пам’ятті програє програмному забезпеченню для спільною пам’яті, як це видно по графікам коеффіцієнта еффективності (рис. 3.2 та рис 3.4).

# Висновки

1. Були розглянуті різні класифікації комп'ютерних архітектур таких авторів як Флінн, Скіллікорн і Хокні. Розкрита ідеологія Флінна, що розбиває комп'ютерні системи на чотири типи: SISD - реалізацією якої є звичайні одноядерні комп'ютери, SIMD, MISD та MIMD - реалізацією яких є трансп’ютерні системи.

2. Більш детально були розглянуті MIMD пристрої – трансп’ютери. Показано їх внутрішню архітектуру, яка складається з інтерфейсів зв'язку, центрального процесора, процесора з плаваючою комою і флеш-пам'яттю. Центральний процесор має стек, що складається з трьох регістрів (А, В, С) які використовуються для обчислення виразів. Вся пам'ять в трансп’ютері циклічна і на логічному рівні ділиться на «мінусову» і на «плюсову» пам'ять.

3. У трансп’ютері реалізований бінарний тип пріоритетності процесів, це означає, що процес може мати високий чи низький пріоритети. Високі пріоритети виконуються без переривань, а низькі квантуються і виконуються в процесорі суворо певну кількість часу. Також у трансп’ютері є простий апаратний планувальник процесів, що оперує процесами, які перебувають у двох чергах на виконання - у черзі високопріоритетних процесів і у черзі низькопріоритетних процесів.

4. Трансп’ютери мають чотири лінка для зв'язку та комунікації з іншими трансп’ютерами, і так само є можливість використання спеціальних інтерфейсів для підключення периферійних пристроїв. За допомогою трансп’ютерів можна будувати локальні обчислювальні системи з різними топологіями, такими як: зірка, лінійка, кільце, гіперкуб та інше.

5. Також було розглянуто мову програмування Оккам, розроблену спеціально для програмування трансп’ютерних систем. Будь-які дії в мові Оккам це процес, який агрегує в собі процеси низького рівня, які також в свою чергу можуть агрегувати процеси більш низького рівня. Таким чином виявлено, що Оккам-програма це дерево процесів, що складається з складових (або процесів-композицій) і примітивних процесів.

6. У мові Оккам примітивних процесів п'ять штук, це такі процеси як присвоювання, нулль-процес (нічого не робить), стоп-процес, процес введення і процес виведення. Процесів-композицій 6 основних і 3 додаткових. Є два найважливіших процесів-композицій: процес в якому процеси виконуються строго послідовно, і процес у якому процеси виконуются паралельно. Для синхронізації між трансп’ютерами використовується процес ALT, який є дуже зручною Оккамовскою реалізацією механізму рандеву.

7. Після розробки програм для системи з локальною і загальною пам‘яттю, можемо стверджувати, що алгоритм роботи для загальної пам‘яті значно легший і при проектуванні, і при реалізації. У випадку системи з локальною пам‘яттю, велика кількість повідомлень різного, динамічно розрахованого розміру, ускладнюють систему з пересилкою повідомлень, а також підвищує вірогідність помилки при реалізації. Також потрібно відмітити, що система з загальною пам‘яттю легше у налагоджені.

8. При малих обсягах пам'яті програмне забезпечення система з локальною пам'яттю програє системі з глобальною пам'яттю. Це викликано тим, що складна топологія системи та час витрачений на передачу даних між процесорами в цілому впливають на загальний час роботи. Проте зі збільшенням обсягу оброблюваних даних локальна система повністю виправдовує зусилля витрачені на налагодження і реалізацію програмного забезпечення. З іншого боку виходячи з аналізу отриманих даних робіт систем, програмне забезпечення для локальних систем обмежено кількістю процесів або ядер які знаходяться в самій системі. При деякій критичній кількості ядер ефективність використання цих ядер буде дуже низьким, незважаючи на високу швидкість оброблюваних значень. Таким чином розроблене масштабоване програмне забезпечення для систем з локальною пам'яттю має більшу перевагу у швидкості обчислення ніж у програмного забезпечення для систем з глобальною пам'яттю. Проте ця перевага обмежена критичною кількістю ядер (чим більше ядер, тим менше ефективність їх використання) з одного боку, і вимогою обчислення великих об'ємів пам'яті з іншого.

# Література

1. **Внуков А. А.** Курс лекций: «Проектирование алгоритмического и программного обеспечения». Москва, 2009.

2. **Чефранов А.Г., Троценко** **Р. В.** Проектирование систем реального времени. *Учебное пособие.* - Таганрог: Изд-во ТРТУ, 2005. - 226 с.

3. **Корнеев В., Киселев А.** Современные микропроцессоры. БХВ-Петербург, 2003. С. 389 – 400.

4. **Джоунз Г.** Программирование на языке Оккам. Пер. с англ. Л. В. Шабанова, под ред. Ю. Г. Дадаева. М: Мир, 1989. — 208с.

5. **Жуков І.А., Корочкін О.В.** Паралельні та розподілені обчислення.Київ : Корнійчук, 2005. с. 240

6. **Лаборатория параллельных информационных технологий НИВЦ МГУ.** Классификация Флинна [З мережі] <http://parallel.ru/computers/taxonomy/flynn.html>

7. **Тарков М. С.** Введение в операционные системы. *Учебное пособие*, НГУ, 1999.

8. **Сайт Фцентр.** Обзор Intel Core 2 Extreme QX6700 и Core 2 Quad Q6600: четырёхъядерные процессоры – это уже не фантастика. *Статья.* [З мережі] http://www.fcenter.ru/online.shtml?articles/hardware/processors/19448

9. **Пескова С. А., Гуров А. И., Кузин А. В.** Центральные и периферийные устройства электронных вычислительных средств/Под ред. О.П.Глудкина. - М.: Радио и связь, 1999.

10. **Бахтеяров C. Д., Дудников Е. Е., Евсеев М. Ю.** Транспьютерная технология. Под ред. С. Е. Емельянова. - М: Радио и связь, 1993. — 304с.

11. **Фонд Викимедиа.** Wikipedia. [MIMD [З мережі] http://ru.wikipedia.org/wiki/MIMD](%20MIMD%20%5bЗ%20мережі%5d%20http://ru.wikipedia.org/wiki/MIMD)

12. **Фонд Викимедиа.** Wikipedia. [Классификация параллельных вычислительных систем [З мережі] http://ru.wikipedia.org/wiki/Классификация параллельных вычислительных систем](%20Классификация%20параллельных%20вычислительных%20систем%20%5bЗ%20мережі%5d%20http://ru.wikipedia.org/wiki/Классификация%20параллельных%20вычислительных%20систем)

13. **Antony J. G. Hey.** Transputers, Occam and General-purpose parallel computing. *Lectures* [З мережі] http://cdsweb.cern.ch/record/340472/files/p93.pdf

14. **Hoare C. A. R.** Communicating Sequential Processes [З мережі] <http://www.usingcsp.com/cspbook.pdf>

15. **Daniel Dr., Hyde C.** Introduction to the Programming Language Occam [З мережі] <http://www.eg.bucknell.edu/~cs366/occam.pdf>

16. **ipsa.kpi.ua.** Классификация архитектур вычислительных систем. [З мережі] [http://studsovet.ipsa.kpi.ua/uploads/mmsa/Kurs2/Arhitektura vychislitelnyh sistem/classification\_architecture.pdf](http://studsovet.ipsa.kpi.ua/uploads/mmsa/Kurs2/Arhitektura%20vychislitelnyh%20sistem/classification_architecture.pdf)

17. **itswtech.** Distributed system[З мережі] <http://www.itswtech.org/Lec/ameer(DCS4th)/lecture2.pdf>

18. **Internet Parallel Computing Archive.** Документация по языку OCCAM. [З мережі] <http://wotug.ukc.ac.uk/parallel/occam/>

19. **MPJ Express.** An Implementation of MPI in Java.Windows User Guide. *Документация к MPJ Express.*

20. **Hasyim Gautama.** A High-Level Performance Model of the T800 Transputer [З мережі] <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.24.8973&rep=rep1&type=pdf>

21. **Hockney R.** Classification and Evaluation of ParallelComputer Systems *Lecture Notes in Computer Science.* 1987. N 295. P.13-25.

22. **http://www.intel.com.** Процессор Intel® Pentium® с технологией MMX™ [[З мережі] http://www.intel.com/p/ru\_RU/support/highlights/processors/pentiummmx](%20http://ru.wikipedia.org/wiki/Таксономия_Флинна)

23. **Michael J Flynn.** Some Computer Organizations and Their Effectiveness. [З мережі] http://www.cs.utah.edu/~kirby/classes/cs6230/Flynn.pdf

# Додатки

## Структурна схема ПКС з ОП

## Структурна схема ПКС з ЛП

## Алгоритм головної программи для ПКС з ОП

## Алгоритми процессів головної программи для ПКС з ОП

## Алгоритм головної программи для ПКС з ЛП

## Алгоритми процессів головної программи для ПКС з ЛП

## Лістінг программи для ПКС з ОП

/\*--------------------------------------------------------------

-- Paralel and distributed computing --

-- Coursework. Java --

-- Func: А=В+С\*(МО\*МХ-МЕ) --

-- IO-83 Vorobyev vitaliy --

-- 14.04.2010 --

--------------------------------------------------------------\*/

package cours;

import java.io.FileWriter;

import java.util.Date;

public class Cours {

public static int N;

public static int P;

public static int H;

static int MX[][] = new int[N][N];

static int MO[][] = new int[N][N];

static int ME[][] = new int[N][N];

static int B[] = new int[N];

static int C[] = new int[N];

static int A[];

public static void main(String[] args) {

N = Integer.valueOf(args[0]);

if (N == 1) N = 2;

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

//if (P == 1) P = 2;

H = N/P;

A = new int[N];

Date date = new java.util.Date();

long startTime = date.getTime();

Tx arr[] = new Tx[P];

Data dat = new Data ();

for (int i = 1; i <= P; i++) {

arr[i-1] = new Tx (i, dat);

}

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

arr[i].start();

}

dat.waitEnd();

date = new java.util.Date();

long finishTime = date.getTime();

long time = finishTime - startTime;

try{

FileWriter sw = new FileWriter("out.txt",true);

sw.write("N = " + N + " P = " + P + " Work time: " + time + " ms \n /n");

sw.close();

}catch(Exception e){

System.out.print(e.getMessage());

}

System.out.println("N = " + N + " P=" + P + " Work time: " + time + " ms");

}

}

package cours;

/\*\*

\* @author Vorobyev Vitaliy

\* Класс, що являє собою захищений модуль. З його допомогою вирішуеться

\* проблема доступу до спільного ресурсу і проблема синхронізації.

\*/

public class Data {

private int F1, F2;

private int [][] MO;

private int [] C;

public Data () {

F1 = 0;

F2 = 0;

}

public synchronized void waitIn() {

try {

if (F1 < 2) { wait(); }

} catch (InterruptedException e) { e.printStackTrace(); }

}

public synchronized void signalIn() {

F1++;

if (F1 == 2) { notifyAll(); }

}

public synchronized void waitEnd() {

try {

while (F2 < Cours.*P* - 1) { wait(); }

} catch (InterruptedException e) {

e.printStackTrace();

}

}

public synchronized void signalEnd() {

F2++;

notifyAll();

}

public synchronized void writeMO(int [][] MTx) {

MO = *clonem*(MTx);

}

public synchronized void writeC(int [] Cx) {

C = *clonev*(Cx);

}

public int[][] copyMO() {

return *clonem*(MO);

}

public int[] copyC() {

return *clonev*(C);

}

public static int[][] clonem(int [][] MM) {

int MT[][] = new int [MM.length][MM[0].length];

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

for (int j = 0; j < MM[0].length; j++) {

MT[i][j] = MM[i][j];

}

}

return MT;

}

public static int[] clonev(int [] MM) {

int C1[] = new int [MM.length];

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

C1[i] = MM[i];

}

return C1;

}

}

package cours;

/\*\*

\* @author Vorobyev Vitaliy

\* Класс, за допомогою якого створюються процесси.

\*/

//А=Вh+С\*(МО\*МХh-МЕh)

public class Tx extends Thread {

public int id;

private int [][] MOx;

private int [] Cx;

private Data dat;

public Tx (int id, Data dat) {

this.id = id;

this.dat = dat;

}

public void run () {

System.out.println ("Process " + id + " STARTED");

if (id == 1) {

MOx = inputMatr();

Cours.B = inputVec();

dat.writeMO(MOx);

dat.signalIn();

}

if (id == Cours.P) {

Cours.MX = inputMatr();

Cours.ME = inputMatr();

Cx = inputVec();

dat.writeC(Cx);

dat.signalIn();

}

dat.waitIn();

System.out.println ("Process " + id + " data copying");

if (id != 1) MOx = dat.copyMO();

if (id != Cours.P) Cx = dat.copyC();

System.out.println ("Process " + id + " processing started");

process();

if (id == 1) {

dat.waitEnd();

if (Cours.N <= 30) outVec (Cours.A);

} else {

dat.signalEnd();

}

System.out.println ("Process " + id + " FINISHED");

}

private void outVec(int[] a) {

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

System.out.print(a[i] + " ");

}

System.out.println();

}

private void process () {

//MT=(!МО\*МХh-МЕh!)

int[][] MT = new int[Cours.N][Cours.N];

int sum1;

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

for (int j = (id-1)\*Cours.H; j < id\*Cours.H; j++) {

sum1 = 0;

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

sum1 = sum1 + MOx[z][j] \* Cours.MX[i][z];

}

MT[i][j] = sum1 - Cours.ME[i][j];

}

}

//А=Вh+С\*MTh

for (int i = (id-1)\*Cours.H; i < id\*Cours.H; i++) {

sum1 = 0;

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

sum1 = sum1 + Cx[j] \* MT[j][i];

}

Cours.A[i] = sum1 + Cours.B[i];

}

}

private int[][] inputMatr () {

int [][] temp = new int[Cours.N][Cours.N];

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

for (int j = 0; j < temp[0].length; j++) {

temp[i][j] = 1;

}

}

return temp;

}

private int[] inputVec () {

int [] temp = new int[Cours.N];

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

temp[i] = 1;

}

return temp;

}

}

## Додаток З. Лістінг программи для ПКС з ОП

/\*--------------------------------------------------------------

-- Paralel and distributed computing --

-- Coursework. MPI --

-- Func: А=В+С\*(МО\*МХ-МЕ) --

-- IO-83 Vorobyev vitaliy --

-- 14.04.2010 --

--------------------------------------------------------------\*/

import java.util.Date;

import mpi.\*;

public class MPI\_main {

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

MPI.Init(args);

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

int size = MPI.COMM\_WORLD.Size();

int P = size;

int N = Integer.parseInt(args[3]); //вводится с консоли при запуске приложения.

int H = N/P;

int currentH\_B\_A;

int currentH\_ME\_MX;

//если верхняя линия

if (rank <= (P / 2) - 1) {

currentH\_B\_A = N - (rank \* 2 \* H);

currentH\_ME\_MX = (rank + 1) \* 2 \* H;

} else { //если нижняя линия

currentH\_B\_A = H;

currentH\_ME\_MX = H;

}

int[] MX = new int[currentH\_ME\_MX \* N];

int[] ME = new int[currentH\_ME\_MX \* N];

int[] B = new int[currentH\_B\_A];

int[] MO = new int[N \* N];

int[] C = new int[N];

int[] A = new int[currentH\_B\_A];

Date date;

long startTime = 0;

if (rank == 0) {

date = new java.util.Date();

startTime = date.getTime();

}

//если НОМЕР ЗАДАЧИ 0 -> ввод В, МО

if (rank == 0) {

//ввод данных

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

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

MO[i\*N + j] = 1;

}

}

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

B[i] = 1;

}

}

//если НОМЕР ЗАДАЧИ == P-2 -> ввод С, МХ, МЕ

if (rank == (P / 2) - 1) {

//ввод данных

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

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

MX[i\*N + j] = 1;

ME[i\*N + j] = 1;

}

}

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

C[i] = 1;

}

int temp;

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

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

temp = MX[i\*N + j];

MX[i\*N + j] = MX[j\*N + i];

MX[j\*N + i] = temp;

temp = ME[i\*N + j];

ME[i\*N + j] = ME[j\*N + i];

ME[j\*N + i] = temp;

}

}

}

//РАССПРОСТРАНЕНИЕ ДАННЫХ

//если НОМЕР ЗАДАЧИ меньше Р

if (rank < (P / 2)) {

//если это не нулевая задача.

if (rank != (P / 2) - 1) {

//принять С, МЕ, МХ

MPI.COMM\_WORLD.Recv(C, 0, N, MPI.INT, rank + 1, rank);

MPI.COMM\_WORLD.Recv(ME, 0, ME.length, MPI.INT, rank + 1, rank);

MPI.COMM\_WORLD.Recv(MX, 0, MX.length, MPI.INT, rank + 1, rank);

}

if (rank != 0) {

//передать С, МЕ, МХ

MPI.COMM\_WORLD.Send(C, 0, N, MPI.INT, rank - 1, rank - 1);

MPI.COMM\_WORLD.Send(ME, 2 \* H \* N, ME.length - 2 \* H \* N, MPI.INT, rank - 1, rank - 1);

MPI.COMM\_WORLD.Send(MX, 2 \* H \* N, MX.length - 2 \* H \* N, MPI.INT, rank - 1, rank - 1);

}

if (rank != 0) {

MPI.COMM\_WORLD.Recv(B, 0, B.length, MPI.INT, rank - 1, rank - 1);

MPI.COMM\_WORLD.Recv(MO, 0, N \* N, MPI.INT, rank - 1, rank - 1);

}

//передать В и МО следующему узлу.

if (rank != (P / 2) - 1) {

MPI.COMM\_WORLD.Send(B, 2 \* H, B.length - 2 \* H, MPI.INT, rank + 1, rank);

MPI.COMM\_WORLD.Send(MO, 0, N \* N, MPI.INT, rank + 1, rank);

}

//передать В и МО нижнему узлу

MPI.COMM\_WORLD.Send(B, H, H, MPI.INT, rank + (P / 2), rank);

MPI.COMM\_WORLD.Send(MO, 0, N \* N, MPI.INT, rank + (P / 2), rank);

//передать С, МЕ, МХ нижнему узлу.

MPI.COMM\_WORLD.Send(C, 0, N, MPI.INT, rank + (P / 2), rank);

MPI.COMM\_WORLD.Send(MX, H \* N, H \* N, MPI.INT, rank + (P / 2), rank);

MPI.COMM\_WORLD.Send(ME, H \* N, H \* N, MPI.INT, rank + (P / 2), rank);

}

//если это нижний ряд, то получаем Н кусочек от В и МО от верхнего ряда

if (rank >= (P / 2)) {

MPI.COMM\_WORLD.Recv(B, 0, H, MPI.INT, rank - (P / 2), rank - (P / 2));

MPI.COMM\_WORLD.Recv(MO, 0, N \* N, MPI.INT, rank - (P / 2), rank - (P / 2));

//С, МЕ, МХ тоже получаем

MPI.COMM\_WORLD.Recv(C, 0, N, MPI.INT, rank - (P / 2), rank - (P / 2));

MPI.COMM\_WORLD.Recv(MX, 0, H \* N, MPI.INT, rank - (P / 2), rank - (P / 2));

MPI.COMM\_WORLD.Recv(ME, 0, H \* N, MPI.INT, rank - (P / 2), rank - (P / 2));

}

//ВЫЧИСЛЕНИЕ

int[] temp = new int[N \* N];

//MT=(!МО\*МХh-МЕh!)

int sum;

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

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

sum = 0;

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

sum += MO[i\*N + j] \* MX[z\*N + j];

}

temp[z\*N + i] = sum - ME[z\*N + i];

}

}

//А=Вh+С\*MTh

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

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

sum = 0;

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

sum += C[i] \* temp[z\*N + j];

}

A[z] = sum + B[z];

}

}

//ОТПРАВКА РЕЗУЛЬТАТА

if (rank >= (P / 2)) {

MPI.COMM\_WORLD.Send(A, 0, A.length, MPI.INT, rank - (P / 2), rank - (P / 2));

}

if (rank < (P / 2)) {

MPI.COMM\_WORLD.Recv(A, H, H, MPI.INT, rank + (P / 2), rank);

if (rank != (P / 2) - 1) {

MPI.COMM\_WORLD.Recv(A, 2 \* H, A.length - 2 \* H, MPI.INT, rank + 1, rank + 1);

}

if (rank != 0) {

MPI.COMM\_WORLD.Send(A, 0, A.length, MPI.INT, rank - 1, rank);

}

}

//РЕЗУЛЬТАТОМ ИМЕЕТСЯ A == [N-1][N][N-3][N-2]...[4][5][2][3][0][1]

if (rank == 0) {

if (N < 20) {

int[] As = new int[N];

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

As[i] = A[N - i - 2];

As[i + 1] = A[N - i - 1];

}

System.out.println("RESULT IS ");

for (int z = 0; z < A.length; z++) {

System.out.print(As[z] + " ");

}

}

}

if (rank == 0) {

date = new java.util.Date();

long finishTime = date.getTime();

long time = finishTime - startTime;

System.out.println();

System.out.println("N = " + N + " P = " + P + " Work time: " + time + " ms");

}

MPI.Finalize();

}

}