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 3 курсу IO-24 групи

напряму підготовки 050102 «Комп’ютерна інженерія»

\_\_\_\_\_\_\_Іванцова О.О.\_\_\_\_\_\_\_\_\_\_

(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

Іванцову Олександру Олександровичу

(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 2014 р.

3. Вхідні дані до роботи :

- огляд засобів роботи з процесами в бібліотеці MPI

- . Мова програмування C#.



- . Мова програмування Ада.



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

- огляд засобів роботи з процесами в бібліотеці MPI

- розробка і тестування програми ПРГ1 для ПКС ОП

- розробка і тестування програми ПРГ2 для 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 2.02.2015р.

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 \_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_**Іванцов О.О.\_\_\_\_\_

( підпис ) (прізвище та ініціали)

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

( підпис ) (прізвище та ініціали)

**ЗМІСТ**

[ВСТУП 3](#_Toc417594442)

[РОЗДЛ 1. ОГЛЯД ЗАСОБІВ РОБОТИ З ПРОЦЕСАМИ В БІБЛІОТЕЦІ MPI 3](#_Toc417594443)

[1.1 Поняття процесу та потоку 3](#_Toc417594444)

[1.2 Бібліотека MPI. Визначення та історія розвитку 3](#_Toc417594445)

[1.3 Поняття паралельної програми в рамках MPI 3](#_Toc417594446)

[1.4 Операції передачі даних в MPI 3](#_Toc417594447)

[1.5 Поняття комунікаторів та груп 3](#_Toc417594448)

[1.6 Типи даних MPI 3](#_Toc417594449)

[1.7 Базові процедури бібліотеки MPI 3](#_Toc417594450)

[1.8 Точки синхронізації, вони ж - бар'єри. 3](#_Toc417594451)

[1.9 Визначення часу виконання MPI-програми 3](#_Toc417594452)

[1.10 Переваги та недоліки використання бібліотеки MPI: 3](#_Toc417594453)

[1.11 Висновки до розділу 1 3](#_Toc417594454)

[2. РОЗРОБКА ПРОГРАМИ ПРГ1 ДЛЯ ПКС ЗІ СПІЛЬНОЮ ПАМ’ЯТТЮ 3](#_Toc417594455)

[2.1 Розробка паралельного математичного алгоритму 3](#_Toc417594456)

[2.2 Аналіз паралелизму вихідної математичної задачі в рамках концепції необмеженого паралелизму 3](#_Toc417594457)

[2.3 Розробка алгоритмів процесів 3](#_Toc417594458)

[2.4 Розробка схеми взаємодії задач 3](#_Toc417594459)

[2.5 Розробка програми ПРГ1 3](#_Toc417594460)

[2.6 Тестування програми ПРГ1 3](#_Toc417594461)

[2.7 Висновки до розділу 2 3](#_Toc417594462)

[3.РОЗРОБКА ПРОГРАМИ ПРГ2 ДЛЯ ПКС З ЛОКАЛЬНОЮ ПАМ'ЯТТЮ 3](#_Toc417594463)

[3.1 Розробка паралельного математичного алгоритму 3](#_Toc417594464)

[3.2 Розробка алгоритмів процесів 3](#_Toc417594465)

[3.3 Розробка схеми взаємодії задач 3](#_Toc417594466)

[3.4 Тестування програми ПРГ2 3](#_Toc417594467)

[3.5 Висновки до розділу 3 3](#_Toc417594468)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 3](#_Toc417594469)

[ДОДАТОК А. СТРУКТУРА ПАРАЛЕЛЬНОЇ КОМП’ЮЕРНОЇ СИСТЕМИ 3](#_Toc417594470)

[ДОДАТОК Б. БЛОК-СХЕМА АЛГОРИТМУ ОСНОВНОЇ ПРОГРАМИ 3](#_Toc417594471)

[ДОДАТОК В. БЛОК-СХЕМА ПОТОКОВОЇ ФУНКЦІЇ 3](#_Toc417594472)

[ДОДАТОК Г. ЛІСТИНГ ПРОГРАМИ ПРГ1 3](#_Toc417594473)

[ДОДАТОК Д. ЛІСТИНГ ПРОГРАМИ ПРГ2 3](#_Toc417594474)

# 

# ВСТУП

З ростом наукового прогресу, задачі, які розв’язуються на комп’ютерах набули значної складності, а вимоги до часу їх розв’язання, точності результату стали більш жорсткішими. 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. Звичайні персональні комп’ютери не можуть забезпечити необхідну швидкість обчислень. Для розв’язання цієї проблеми застосовують паралельні та розподілені обчисленя і відповідні машини, що мають декілька процесорів.

Ефективність розв’язання задачі шляхом паралельного програмування залежить не лише від потужності машини, наявних технічних ресурсів(швидких ліній передач), а і від навиків програміста, його вміння правильно розпаралелити задачу. В даній роботі буде розв’язуватись задача, шляхом розпаралелення на декілька процесів, що будуть взаємодіяти між собою. Це дасть базові навички в розробці паралельних алгоритмів і програм.

В розділі 1 проводиться огляд роботи з процесами в бібліотеці MPI. Дана бібліотека ефективна при використанні систем з розподіленою пам'яттю де процесори працюють незалежно один від одного. Для організації паралельних обчислень в таких умовах необхідно мати можливість розподіляти обчислювальне навантаження і організувати інформаційну взаємодію (передачу даних) між процесорами. Рішення цих проблем і забезпечує інтерфейс передачі даних (Message Passing Interface - MPI).

В розділі 2 розробляється програма ПРГ1 для паралельної комп’ютерної системи зі спільною пам’яттю, та необмеженою кількістю процесорів. Проводиться тестування розробленої програми для визначення коефіцієнта прискорення та коефіцієнта ефективності.

В розділі 3 розробляється програма ПРГ2 для ПКС з локальною пам’яттю. Визначаються коефіцієнти прискорення та ефективності.

# РОЗДЛ 1. ОГЛЯД ЗАСОБІВ РОБОТИ З ПРОЦЕСАМИ В БІБЛІОТЕЦІ MPI

## 1.1 Поняття процесу та потоку

Термін "процес" вперше з'явився при розробці операційної системи Multix і має кілька визначень, використання яких залежить від конкретного контексту [4]. Отже, процес - це:

* програма на стадії виконання
* "об'єкт", якому виділений процесорний час
* асинхронна робота

Процес характеризує деяку суккупність набору команд, асоційований з ним набір ресурсів (виділена пам'ять, адресний простір, стеки, пристрої введення виведення) та інформації про поточний момент його виконання, що знаходиться під управлінням операційної системи [3].

За час свого життя процес може знаходитись в одному з трьох станів:

* виконання
* готовність
* очікування

*Виконання* - це активний стан процесу, під час якого він має всі необхідні йому ресурси для виконання. В цьому стані процес безпосередньо виконується на процесорі.

*Очікування* - пасивний стан процесу, під час якого процес блокований і очікує якусь подію, наприклад, дані від іншого процесу.

*Готовність* - пасивний стан, процес заблокований і чекає звільнення ресурсів.

Можна виділити ще два стани процесу - це на родження і смерть. Народження процесу - пасивний стан, коли готова структура для появлення процесу, але самого процесу, як такого, ще немає. Смерть процесу - самого процесу вже немає, але його "місце", тобто структура данних, ще залишилась у списку процесів. Такі процеси називають зобмі [4].

Напрямки переходів між станами зручніше показати на діаграмі.

Виконання

Готовність

Смерть(завершення)

Очікування

Народження(створення)

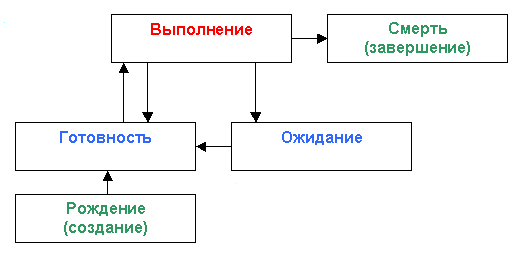


Рис. 1.1 Діаграма переходів між станами процесу

Незалежно від того, які засоби використовуються для створення процесу, вони мають мають хабезпечувати набір наступних операцій з процесами:

1. Створення процесу - перехід із стану народження у встан готовності.
2. Знищення процесу - перехід зі стану виконання у стан смерті.
3. Відновлення процесу - перехід зі стану готовності у стан виконання.
4. Зміна пріорітету процесу - надання переваги перед іншими процесами.
5. Блокування процесу - перехід в стан очікування.
6. Розблокування процесу - перехід із стану очікування у стан готовності.
7. Запуск процесу - перехід зі стану готовні готовності в стан виконання.

Для створення процесу операційна система робить наступне:

1. Присвоює ім’я процесу
2. Добавляє інформацію про процес в список процесів
3. Визначає пріорітет процесу
4. Формує блок управління процесом
5. Надає процесу потрібні йому ресурси

Зрозуміло що процес не може з’явитись сам по собі. Його має запустити інший процес. Процес, запущений іншим процесом називаєься дочірнім, а процес, який запустив інший процес, вдповідно – материнським. Коден процес повинене мати PID (Process ID)та PPID (Parent process ID). Таким чином процеси створюють дерево процесів, у вершині якого стоїть процес init його ID = 1.

Поняття потік і процес часто путають між собою. На практиці це принципово різні речі. Потік це підпроцес, чи так званий «легкий процес»(Light Weight process), що виконуться в контексті одного процесу. Завдання потоків – забезпечити паралельне виконання процесу. Процес інкапсулює в собі один чи більше потоків. Виходячи з визначення, поняття процес і потік, можна сформулювати визначення багатопотоковості. Багатопотоковість – це здатність операційної системи підтримувати в рамках одного процесу виконання декількох потоків.

Концепція потоків надає можливість одночасного виконання в одному і тому ж процесі декількох програм, які є достатньою мірою незалежні одне від одного. Кілька потоків, що працюють паралельно в одному процесі, аналогічні декільком процесам, що виконуються поаралельно на одному комп’ютері. У першому випадку потоки поділяють адресний простір, відкриті файли та інші ресурси. У другому випадку процеси спільно користуються фізичною пам'яттю, дисками, принтерами і іншими ресурсами [4].

Кожному процесу відповідає адресний простір і одиночний потік команд що викнд що виконуються. У багатокористувацьких системах, при кожному зверненні до одного і того ж сервісу, доводиться створювати новий процес для обслуговування клієнта. Це менш вигідно, ніж створити квазіпараллельний потік всередині цього процесу з одним адресним простором [3].

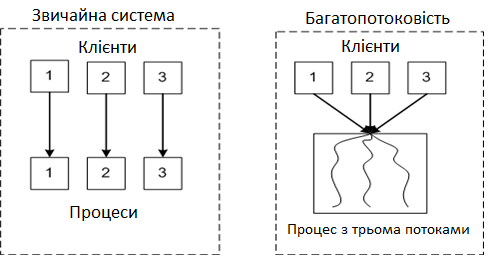


Рис. 1.2 Пороівняння однопотокової та багатопотокової системи

На практиці значно вигідніше застосовувати концепцію потоків ніж створювати окремий процес для кожної задачі, тому що :

* На створення потоку затрачається у 100 разів менше часу аніж на створення процесу.
* На завершення потоку, також, витрачається менше часу
* Час переключення між потоками значно менший ніж час переключення між двома задачми
* Створення потоку менш ресурсозатратне, оскільки потоки розділяють ресурси процесу в якму вони знаходяться.
* Дані, які продукуаються одним потоком негайно стають доступні іншим потокам, оскільки всі потоки знаходяться в одному адресному просторі.

## 1.2 Бібліотека MPI. Визначення та історія розвитку

**Message Passing Interface** (MPI, інтерфейс передачі повідомлень) — програмний інтерфейс для передачі інформації, який дає можливість обмінюватись повідомленнями процесам, що виконують одну задачу [2], [6].

Бібліотека MPI доступна в мовах С/С++, Fortran. Існують безкоштовні та комерційні реалізації даної бібліотеки для більшості суперкомп’ютерних платформ. Сьогодні MPI широко використовується і динамічно розвивається у своїй області застосування [8].

Коротка історія розвитку бібліотеки MPI :

* 1992 року почалася робота над стандартом ефективної бібліотеки передачі повідомлень (Oak Ridge National Laboratory, Rice University). У той час вже використовувалися декілька різних комунікаційних бібліотек і відчувалася гостра необхідність в єдиному стандарті, який зібрав би воєдино кращі риси цих бібліотек.
* 29-30 квітня 1992 року Workshop on Standards for Message Passing in a Distributed Memory Environment (Williamsburg, Virginia). Обговорено необхідні основні риси стандарту MPI, заснована робоча група по стандартизації.
* Листопад 1992 рік. З'явився робочий варіант стандарту MPI1, що викликав велику дискусію.
* Лютий 1992 рік. Опублікований переглянутий варіант MPI1 (Dongarra, Hempel, Hey and Walker). Створена організація MPI Forum (link is external) для подальшого розвитку стандарту. Для обговорення стандарту та окремих його розділів були створені кілька відкритих списків розсилки. Зборів MPI Forum призначалися кожні 6 тижнів.
* Листопад 1993 рік. Робочий варіант стандарту представлений на конференції Supercomputing'93.
* 5 травня 1994 року. Остаточний варіант стандарту MPI 1.0.
* 12 червня 1995 року. Нова версія стандарту - MPI 1.1. Виправлені деякі помилки оригінального стандарту і внесені роз'яснення.
* 18 липня 1997 року. Опубліковано стандарт MPI-2: Extensions to the Message-Passing Interface.

## 1.3 Поняття паралельної програми в рамках MPI

Паралельна програма в рамках MPI - це множина процесів, що виконується олночасно. Процеси можуть виконуватись на різних процесорах, але на одному процесорі можуть розміщуватись і кілька процесів. Кожен процес породжується за допомогою копіювання одного і того ж програмного коду. Даний програмний код, має бути доступний на всіх процесорах в иомент запуску [5].

Кількість процесів та процесорів, що будуть задіяні визначаються засобами середовища виконання MPI програм. Починаючи з версії MPI 2.0, бібліотека має засоби для динамічного породження процесів та керування ними. Всі процеси програми пронумеровані від 0 до p-1, де р – кількість процесорів, що викорстовуються для розв’язку задачі. Номер процесу називається рангом процесу [5], [2].

## 1.4 Операції передачі даних в MPI

Основу MPI становлять операції передачі повідомлень. Серед передбачених у складі MPI функцій розрізняються парні (point to point) операції між двома процесами та колективні (collective) комунікаційні дії для одночасної взаємодії декількох процесів [5], [8].

Функція MPI\_Send забезпечує так званий стандартний (standard) режим відправки повідомлень, при якому:

* на час виконання функції процес - відправник повідомлення блокується;
* після завершення функції буфер може бути використаний повторно;
* стан відправленого повідомлення може бути різним – повідомлення може знаходитись в процесі відправнику, може перебувати в стані передачі, може зберігатися на процесі отримувачу або ж може бути прийнятий процесом отримувачем за допомогою функції MPI\_Recv.

Крім стандартного режиму в MPI передбачаються наступні додаткові режими передачі повідомлень:

* Синхронний (synchronous) режим полягає в тому, що завершення функції відправки повідомлення відбувається тільки при отриманні від процесу одержувача підтвердження про початок прийому відправленого повідомлення. Відправлене повідомлення або повністю прийнято процесом одержувачем, або знаходиться в стані прийому;
* Буферизованний (buffered) режим передбачає використання додаткових системних буферів для копіювання в них повідомлень, що відправляються. Функція відправки повідомлення завершується відразу ж після копіювання повідомлень до системного буфера;
* Режим передачі по готовності (ready) може бути використаний тільки, якщо операція прийому повідомлення вже ініційована. Буфер повідомлення після завершення функції відправки повідомлення може бути повторно використаний.

Для іменування функцій відправки повідомлення для різних режимів виконання в MPI застосовується назва функції MPI\_Send, до якого як префікс додається початковий символ назви відповідного режиму роботи, тобто :

• MPI\_Ssend - функція відправки повідомлення в синхронному режимі;

• MPI\_Bsend - функція відправки повідомлення в буферизованного режимі;

• MPI\_Rsend - функція відправки повідомлення в режимі по готовності.

Список параметрів всіх перерахованих функцій збігається зі складом параметрів функції MPI\_Send.

Для застосування буферизованного режиму передачі може бути створений і переданий MPI буфер пам'яті, функція, що в икориствується для цього має вигляд:

int MPI\_Buffer\_attach (void \* buf, int size),

де

• buf - адреса буфера пам'яті;

• size - розмір буфера.

Після завершення роботи з буфером він повинен бути відключений від MPI за допомогою функції:

int MPI\_Buffer\_detach (void \* buf, int \* size),

де

• buf - адреса буфера пам'яті;

• size - повертаний розмір буфера.

Щодо практичного використання режимів можна навести такі рекомендації:

* стандартний режим зазвичай реалізується як буферізірованний або синхронний, в залежності від розміру переданого повідомлення, і найчастіше є найбільш оптимізованим по продуктивності;
* режим передачі по готовності формально є найбільш швидким, але використовується досить рідко, т. к. зазвичай складно гарантувати готовність операції прийому;
* буферізованние режим також виконується досить швидко, але може приводити до великих витрат ресурсів (пам'яті), - в цілому може бути рекомендований для передачі коротких повідомлень;
* синхронний режим є найбільш повільним, тому вимагає підтвердження прийому, однак не потребує додаткової пам'яті для зберігання повідомлення. Цей режим може бути рекомендований для довгого повідомлення.

## 1.5 Поняття комунікаторів та груп

Ще одним важливим поняттям в MPI, є поняття комунікатора. Під комунікатором в MPI розуміється спеціальний службовий об'єкт, який об'єднує в своєму складі групу процесів і ряд додаткових параметрів (контекст), використовуваних при виконанні операцій передачі даних. Парні операції передачі даних виконуються тільки для процесів, які належать до одного комунікатора. Колективні операції застосовуються одночасно для всіх процесів одного комунікатора. Як результат, вказівка комунікатора, що використовується, є обов'язковим для операцій передачі даних в MPI [1].

У ході обчислень можуть створюватися нові і видалятися існуючі групи процесів і комунікаторів. Один і той же процес може належати різним групам і комунікаторам. Усі наявні в паралельній програмі процеси входять до складу комунікатора за замовчуванням з ідентифікатором MPI\_COMM\_WORLD [1], [8].

У версії 2.0 з'явилася можливість створювати глобальні комунікатори (intercommunicator), що об'єднують в одну структуру пару груп при необхідності виконання колективних операцій між процесами з різних груп.

Для створення нових комунікаторів застосовні два основних способи їх отримання:

• дублювання вже існуючого комунікатора:

int MPI\_Comm\_dup (MPI\_Comm oldcom, MPI\_comm \* newcomm),

• створення нового комунікатора з підмножини процесів існуючого комунікатора:

int MPI\_comm\_create (MPI\_Comm oldcom, MPI\_Group group, MPI\_Comm \* newcomm).

Дублювання комунікатора може використовуватися, наприклад, для усунення можливості перетину за тегами повідомлень в різних частинах паралельної програми (в т.ч. і при використанні функцій різних програмних бібліотек).

Слід зазначити також, що операція створення комунікаторів є колективною і, тим самим, повинна виконуватися всіма процесами вихідного комунікатора.

Група - це впорядкована множина процесів. Кожному процесу в групі зіставлено ціле число - ранг або номер. MPI\_GROUP\_EMPTY - порожня група, яка не містить жодного процесу. MPI\_GROUP\_NULL - значення, яке використовується для помилковою групи [1].

Групи процесів можуть бути створені тільки з уже існуючих груп. В якості вихідної групи може бути використана група, пов'язана з наперед визначеним комунікатором MPI\_COMM\_WORLD. Для отримання групи, пов'язаної з існуючим комунікатором, використовується функція:

int MPI\_Comm\_group (MPI\_Comm comm, MPI\_Group \* group).

Далі, на основі існуючих груп, можуть бути створені нові групи:

• створення нової групи newgroup з існуючої групи oldgroup, яка включатиме в себе n процесів, ранги яких перераховуються в масиві ranks:

int MPI\_Group\_incl (MPI\_Group oldgroup, int n, int \* ranks, MPI\_Group \* newgroup),

• створення нової групи newgroup з групи oldgroup, яка включатиме в себе n процесів, ранги яких не збігаються з рангами, перерахованими в масиві ranks:

int MPI\_Group\_excl (MPI\_Group oldgroup, int n, int \* ranks, MPI\_Group \* newgroup).

Для отримання нових груп над наявними групами процесів можуть бути виконані операції об'єднання, перетину і різниці:

• створення нової групи newgroup як об'єднання груп group1 і group2:

int MPI\_Group\_union (MPI\_Group group1, MPI\_Group group2, MPI\_Group \* newgroup);

• створення нової групи newgroup як перетину груп group1 і group2:

int MPI\_Group\_intersection (MPI\_Group group1, MPI\_Group group2,

 MPI\_Group \* newgroup),

• створення нової групи newgroup як різниці груп group1 і group2:

int MPI\_Group\_difference (MPI\_Group group1, MPI\_Group group2,

 MPI\_Group \* newgroup).

При конструюванні груп може виявитися корисною спеціальна порожня група MPI\_COMM\_EMPTY.

Ряд функцій MPI забезпечує отримання інформації про групу процесів:

• отримання кількості процесів в групі:

int MPI\_Group\_size (MPI\_Group group, int \* size),

• отримання рангу поточного процесу в групі:

int MPI\_Group\_rank (MPI\_Group group, int \* rank).

Після завершення використання група повинна бути вилучена:

int MPI\_Group\_free (MPI\_Group \* group)

(Виконання даної операції не торкається комунікатори, в яких використовується видаляється група).

## 1.6 Типи даних MPI

При виконанні операцій передачі повідомлень, для вказання даних що передаватимуться чи прийматимуться в процедурах MPI необхідно вказувати тип даних. MPI містить стандартні типи даних, які співпадають з типами мов С та Fortran. В MPI також є можлвість створити новиі похідні типи даних для більш точного і короткого описаня повідомлень, що перессилаються [2].

## 1.7 Базові процедури бібліотеки MPI

Для створення паралельної програми засобами MPI, можна використовувати процедурні мови програмування типу C чи Fortran. Всі процедури бібліотеки MPI стають доступними після виклику процедури MPI\_Init.

MPI\_Init (int\* argc, char\*\*\* argv) – ініціалізація паралельної частини програми. Реальна ініціалізація для кожної програми виконується один раз. В разі успішного виконання процедура повертає значення MPI\_SUCCESS, інакше – код помилки. (Всі інші процедури MPI повиртають такі ж значення ) [2], [6].

Кожний процес має викликати процедуру MPI\_Finalise.

MPI\_Finalize() - завершення паралельної частини програми. Усі наступні звернення до будь-якої MPI-процедури, у тому числі до MPI\_Init, заборонені. До моменту виклику процесом процедури MPI\_Finalize всі дії, що вимагають його участі в обміні повідомленнями, повинні бути завершені.

int MPI\_Comm\_size(MPI\_Comm comm, int \*size) – визначення кількості процесів у програмі що виконуться. Параметри:

comm – комунікатор, розмір якого визначається.

size – сюди процедура покладе значення що дорівнює кількості процесів в комункікаторі.

int MPI\_Comm\_rank(MPI\_Comm comm, int \*rank) – процедура, яка визначає ранг процесу[6]. Параметри:

comm – комунікатор, в якому визначається ранг процеса

rank – ранг процесу в комунікаторі

int MPI\_Send(void \*buf, int count, MPI\_Datatype type, int dest, int tag, MPI\_Comm comm) – процедура призначеня для передачі повідомлення.Викликається в тілі програми процесу – відправника[2]. Параметри:

buf - адреса буфера пам'яті, в якому розташовуються дані

повідомлення, що відправляється;

count - кількість елементів даних в повідомленні;

type - тип елементів даних пересилається повідомлення;

dest - ранг процесу, якому відправляється повідомлення;

tag – значення тег, що використовується для ідентифікації повідомлення;

comm - комунікатор, в рамках якого відбуватиметься передача

даних.

Після перегляду основних процедур бібіліотеки MPI, можна виокремити деякі особливості синтаксису. Імена процедур починаються з префікса MPI, далі йде одне чи кілька слів назви процедури, слова розділяється нижнім підкреслення м. Назви процедур пояснюють призначення тієї чи іншої процедури.

Приклад паралелельної програми з використанням засобів MPI:

#include "mpi.h" // Пдключення бібіліотеки MPI

int main(int argc, char \*argv[]){

MPI\_Init(&argc, &argv); // інціалізація паралельної частини програми

// Тіло паралельної програми

MPI\_Finalize(); // завершення паралельної частини програми

return 0;

}

## 1.8 Точки синхронізації, вони ж - бар'єри.

Цим займається всього одна функція:

    int MPI\_Barrier (MPI\_Comm comm);

MPI\_Barrier зупиняє виконання викликала її завдання до тих пір, поки не буде викликана з усіх інших завдань, під'єднаних до що вказується коммуникатору. Гарантує, що до виконання наступної за MPI\_Barrier інструкції кожна задача приступить одночасно з іншими [2], [5].

Це єдина в MPI функція, викликами якої гарантовано синхронізується в часі виконання різних гілок! Деякі інші колективні функції залежно від реалізації можуть володіти, а можуть і не мати властивість одночасно повертати управління всім гілкам; але для них це властивість є побічним і необов'язковим - якщо Вам потрібна синхронність, використовуйте тільки MPI\_Barrier [5].

## 1.9 Визначення часу виконання MPI-програми

Практично відразу ж після розробки перших паралельних програм виникає необхідність визначення часу виконання обчислень для оцінки досягнутого прискорення процесів вирішення задач за рахунок використання паралелизму. Використовувані зазвичай засоби для вимірювання часу роботи програм залежать, як правило, від апаратної платформи, операційної системи, алгоритмічної мови і т.п. Стандарт MPI включає визначення спеціальних функцій для вимірювання часу, використання яких дозволяє усунути залежність від середовища виконання паралельних програм [5].

Отримання часу поточного моменту виконання програми забезпечується за допомогою функції:

 double MPI\_Wtime (void),

результат виклику якої є кількість секунд, що минув від деякого певного моменту

часу в минулому. Цей момент часу в минулому, від якого походить відлік секунд, може залежати від середовища реалізації бібліотеки MPI і, тим самим, для відходу від такої залежності функцію MPI\_Wtime слід використовувати тільки для визначення тривалості виконання тих чи інших фрагментів коду паралельних програм. Можлива схема застосування функції MPI\_Wtime може полягати в наступному:

 double t1, t2, dt;

 t1 = MPI\_Wtime ();

 ...

 t2 = MPI\_Wtime ();

 dt = t2 - t1;

Точність вимірювання часу також може залежати від середовища виконання паралельної програми. Для визначення поточного значення точності може бути використана функція:

 double MPI\_Wtick (void),

що дозволяє визначити час у секундах між двома послідовними показниками часу апаратного таймера використовуваної комп'ютерної системи.

## 1.10 Переваги та недоліки використання бібліотеки MPI:

Переваги:

* 1. MPI допомагає вирішити проблему переносимості паралельних програм між різними комп'ютерними системами.
  2. MPI сприяє підвищенню ефективності паралельних обчислень: практично для кожного типу обчислювальних систем існують реалізації бібліотек MPI, що враховують можливості використовуваного комунікаційного устаткування.
  3. MPI полегшує процес написання паралельних програм, коли при розробці використовуються бібліотеки програмних модулів, написаних з використанням цього інтерфейсу.
  4. MPI дозволяє створювати легко масштабовані паралельні програми.

Недоліки:

* 1. MPI є низькорівневим інструментом програміста.
  2. Не існує реалізацій MPI, яка повною мірою забезпечить суміщення обмінів між обчисленнями.
  3. MPI не надає механізмів завдання початкового розміщення процесів по процесорах.

## 1.11 Висновки до розділу 1

1. Проведено огляд засобів роботи з процесами в бібліотеці MPI, який показав, що MPI є одним з поширених засобів програмування для комп’ютерних систем з розподіленою пам’яттю.

Процеси в MPI об’єднуються в комунікатор, який дозволяє одночасно звертатись до всіх задач що знаходяться в ньому. Це дає можливість виділити групу процесів що розв’язує спільну задачу.

В середині комунікатора всі процеси мають унікальні ідентифікатори – ранки. Ранк потрібен для того, щоб створити взаємодію задач, шляхом передачі повідомлень між процесами.

2. Проведено огляд взаємодії задач через що механізм відправки повідомлень, що є основним інструментарієм для забезпечення зв'язку між окремими процесами паралельної програми в MPI. Бібліотека надає програмісту єдиний механізм взаємодії процесів всередині паралельнгої програми незалежно від машинної архітектури та взаємного розташування процесів

3. На основі виконаного огляду можна сказати, що бібліотека MPI - це добре стандартизований механізм для побудови програм по моделі обміну повідомленнями. Програму, що використовує MPI, легко налагоджувати і переносити на інші платформи, часто для цього достатньо простої перекомпіляції вихідного тексту програми. MPI орієнтований на системи з розподіленою пам'яттю, тобто коли витрати на передачу даних великі, бібліотека MPI має всі засоби для розв’язання даної проблеми.

# РОЗДІЛ2 2. РОЗРОБКА ПРОГРАМИ ПРГ1 ДЛЯ ПКС ЗІ СПІЛЬНОЮ ПАМ’ЯТТЮ

В даному розділі проводиться розробка та тестування програми зі спільною пам’яттю. Згідно технічного завдання, розробка програми проводиться на мові програмування C#. Для розв’язання задач синхронізації та взаємного виключення використовуються події, критичні секції, атомарні змінні.

Математична задача:



Структура ПКС з спільною пам’яттю:

СП

. . .

P

1

MA, MX, MR

MO, MK, a

Рис. 2.1 Структура паралельної комп’ютерної системи зі спільною пам’яттю

## 2.1 Розробка паралельного математичного алгоритму

Для розв’язання математичної задачі засобами паралельного програмування потрібно розробити паралельний математичний алгоритм.



де



N – розмірність матриць

P – кількість процесорів

Спільні ресурси:



## 

## 2.2 Аналіз паралелизму вихідної математичної задачі в рамках концепції необмеженого паралелизму

Оцінка паралельного алгоритму проводиться на основі теореми Мунро-Патерсона.

Теорема Мунро-Патерсона. Якщо в комп’ютерній системі з P процесорів виконуєтьс обчислення скалярної величини, яке потребує m бінарних операцій, то необхідний час визначається як:



Для знаходження мінімуму в кожному рядку матриці необхідно виконати N порівнянь ( обчислення 1 ), далі, лдя знаходження мінімуму серед мінімумів рядків, необхідно виконати ще N порівнянь (обчислення 2 ). Всього, для знаходження мінімуму потрібно виконати N \* N порівнянь. Отже:



m = N\*N;



Для виконання обчислення 3 () знадобиться 1 операція множення, 1 операція додавання, 1 операція множення, N операцій множення, N – 1 операцй додавання. Загалом виходить m = 2\*N + 2 бінарних операцій. Отже:



В кінцевому результаті час розв’язання задачі:



## 2.3 Розробка алгоритмів процесів

**Задача Т1: ТС, КД**

1. Введення MA, MX, MR
2. Сигнал задачам Т2..ТР про введення даних S2..P; 1
3. Обчислення 1



1. **Обчислення 2 КД**



1. Сигнал задачам Т2..ТР про завершення

обчислення 2 S2..P; 2

1. Чекати сигналу від задач Т2..ТР про завершення

обчислення 2 W2..P; 1

1. **Копіювання m1 = m, a1 = a, MK1 = MK КД**
2. Обчислення 3



1. Чекати сигналу від задач Т2..ТР про завершення

обчислення 3 W2..P; 2

1. Виведення МА

**Задача Ті:**

1. Чекати сигналу від Т1 про введення даних W1; 1
2. Обчислення 1



1. **Обчислення 2 КД**



1. Сигнал задачам Т1..Ті-1, Ті+1..ТР про завершення

обчислення 2 S1..i-1, i+1..P; 1

1. Чекати сигналу від задач Т1..Ті-1, Ті+1..ТР про

завершення обчислення 2 W1..i-1, i+1..P; 2

1. **Копіювання mі = m, aі = a, MKі = MK КД**
2. Обчислення 3



1. Сигнал задачі Т1 про завершення обчислення 3 S1; 2

**Задача Тр:**

1. Введення МО, МК, а
2. Чекати сигналу від Т1 про введення даних W1; 1
3. Обчислення 1



1. **Обчислення 2 КД**



1. Сигнал задачам Т1..Тр-1  про завершення

обчислення 2 S1..P-1; 1

1. Чекати сигналу від задач Т1..Тр-1  про завершення

обчислення 2 W1..P-1; 2

1. **Копіювання mР = m, aР = a, MKР = MK КД**
2. Обчислення 3



1. Сигнал задачі Т1 про завершення обчислення 3 S1; 2

## 2.4 Розробка схеми взаємодії задач

На соснові алгоритмів процесів розроблена схема взаємодії задач (рис. 2.1). Мова програмування, на якій реалізовуватиметься програма ПРГ1 – C#. Для розв’язання задач синхронізації обрано механізм подій (Event). Для розв’язання задачі взаємного виключення використовуються механізми атомарних змінних (Volitaile) і механізм критичних секцій (Lock).

Перша критична ділянка захищена за допомогою механізму атомарних змінних. Змінна m, яка є спільним ресурсом в першій критичній ділянці, об’явленя як volitaile. Це означає, що операційна система «не дозволить» одночасний доступ до цієї змінної з кількох потоків одночасно.

Друга критична ділянка захищена за допомогою механізму критичних секцій. Для цього створено об’єкт lockObject, який може використовувати лише один потік в один момент часу.

Для розв’язання задачі синхронізації, створюються масиви подій (EventCalculating2Finished, EventCalculating3Finished) довжиною P, кожен елемент масиву використовується для відправки сигналу про подію з однієї задачі в іншу.

## 2.5 Розробка програми ПРГ1

Програма ПРГ1 розроблена на мові програмування C#.

Програма складається з двох класів Data та Program. Клас Data містить в собі набір статичних методів для розв’язання математичної задачі згідно з технічного завдання. Опис методів класу Data:

* minInMatrix(int[][] matrix, int leftBound, int rightBound) – метод для пошуку мінімуму в матриці matrix, з рядка під номером leftBound по рядок під номером rightBound.
* public static int minValue(int x, int y) – метод повертає мінімальне з чисел x та y.
* public static void outputMatrix(int[][] matrix, String name) – метод для виведення матриці matrix з іменем name.
* public static int[][] inputMatrix(int size) – метод для введення матриці розміром size.

?????

Volitaile

ТР

.

.

.

Event

Calculating2

Finished [i]

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

E\_1

Т1

Event

Calculating2

Finished [1]

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

Event

Calculating3

Finished [i]

Ті

Event

Calculating2

Finished [P]

Event

Calculating3

Finished [P]

m

a, m, MK

lockObject

Рис. 2.1 Схема взаємодії процесів

* public static void function(int[][] MA, int m, int[][] MO, int a, int[][] MK, int[][] MR, int LeftBound, int rightBound) – метод що проводить обчислення 3
* public static int[][] copyMatrix(int[][] matrix) – метод що повертає копію матриці matrix.

Клас Program містить статичні поля матриці, константи та масиви подій, що необхідні для розв’язку математичної задачі засобами паралельного програмування.

Методи клас Program:

* + static void threadFunction(Object threadNumber) – потоковий метод, параметром якого є ідентифікатор (порядковий номер) потоку. Залежно від номеру потоку, метод виконує ті чи інші операції.
  + static void Main(string[] args) – основний метод, де ініціюються всі змінні, створюються і запускаються потоки.

Лістинг розробленої програми знаходиться в додатку Б.

## 2.6 Тестування програми ПРГ1

В даному розділі проводиться обчислення коефіцієнта прискорення та коефіцієнта ефективності для числа процесорів – 1, 2, 3, 4 при розмірностях матриць – 900, 1800, 2400.

Коефіцієнт прискорення обчислюється за формулою:



Для обчислення коефіцієнта прискорення, необхідно визначити час розв’язання математичної задачі. Час розв’язання задачі при різних P та N занесено до таблиці 2.1.

Таблиця 2.1 ХХХХХХХХХХХХХХХХХХХХХХХХХХХХХ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N P | 1 | 2 | 3 | 4 |
| 900 | 25с | 15с | 12с | 8с |
| 1800 | 293с | 180с | 133с | 92с |
| 2400 | 717с | 486с | 315с | 230с |

В таблиці 2.2 наведено пораховані коефіцієнти прискорення.

Таблиця 2.2 ХХХХХХХХХХХХХХХХХХХХХХ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N P | 1 | 2 | 3 | 4 |
| 900 | 1 | 1,67 | 2,08 | 3,18 |
| 1800 | 1 | 1,63 | 2,2 | 3,28 |
| 2400 | 1 | 1,47 | 2,28 | 3,45 |

На основі даних з таблиць 2.1 і 2.2 проведено обчислення коефіцієнта ефективності, який розраховується за формулою.



де – коефіцієнт прискорення при P процесорах.



Розраховані коефіцієнти ефективності наведено в таблиці 2.3.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N P | 1 | 2 | 3 | 4 |
| 900 | 100% | 83,5% | 69,33% | 62,5% |
| 1800 | 100% | 81,5% | 73,33% | 67,25% |
| 2400 | 100% | 73,5% | 76% | 67,75% |

Таблиця 2.3 ХХХХХХХХХХХХХХХХХХХХХХХХХХХ

На основі даних з таблиць 2.1, 2.2, 2.3 будуються графіки залежності часу виконання від кількості процесорів, коефіцієнта прискорення від кількості процесорів, коефіцієнта ефективності від кількості процесорів.

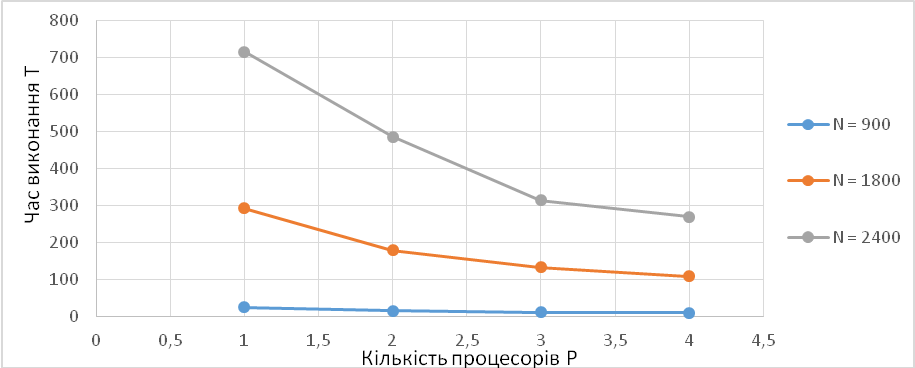


Рис 2.2 Графік залежності часу виконання задачі від кількості процесорів

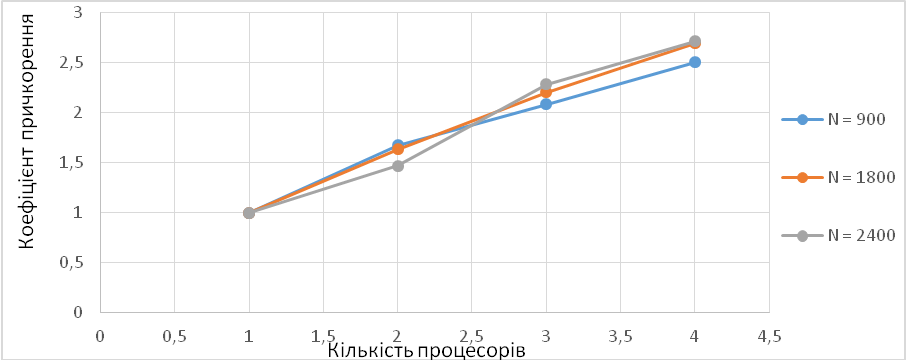


Рис. 2.3 Графік залежності коефіцієнта прискорення від кількості процесорів

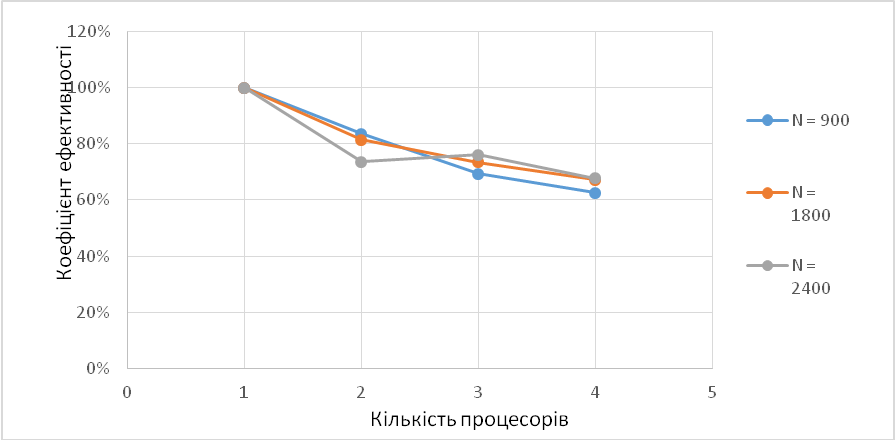


Рис. 2.4 Графік залежності коефіцієнта ефективності від кількості процесора

## 2.7 Висновки до розділу 2

1. В даному розділі розроблено програму ПРГ1 для розв’язку математичної задачі, згідно з варіанту. Програма розроблена на мові C#, кількість задач нефіксована. Така система дозволяє ефективно розв’язувати задачу, налаштовуючи програму в залежності від введених даних.
2. Коефіцієнт прискорення росте зі збльшенням розмірностей матриць та зі збільшенням кількості ядер. Коефіцієнт ефективності росте зі зменшенням кількості ядер. Це зумовлено тим, що при більшій кількості ядер, вони можуть простоювати, чекаючи, поки на іншому ядрі задача виконається до необхідної точки.
3. Виходячи з результатів тестування програми, можна сказати, що ефективність використання паралельних програм росте зі збільшенням об’єму самих задач. Тобто паралельні комп’ютерні системи ефективні при розв’язанні масштабних задач і втрачають свою ефективність при розв’язання нескладних задач. Це пов’язано з тим що значна частина часу затрачається на ініціалізацію потоків та забезпечення їх взаємодії.

# 3.РОЗРОБКА ПРОГРАМИ ПРГ2 ДЛЯ ПКС З ЛОКАЛЬНОЮ ПАМ'ЯТТЮ

В даному розділі проводиться розробка та тестування програми з локальною пам’яттю. Згідно технічного завдання, розробка програми проводиться на мові програмування Ада. Взаємодія задач реалізована за допомогою механізму рандеву.

Математична задача:



Структуру ПКС з локальною пам’яттю представлено на рис. 3.1:

MO, MK, a

MA, MX, MR

Рис. 3.1 Структура паралельної комп’ютерної системи з локальною пам’яттю

## 3.1 Розробка паралельного математичного алгоритму

Для розв’язання математичної задачі на ПКС з локальною пам’яттю, потрібно розробити паралельний математичний алгоритм.



де



N – розмірність матриць

P – кількість процесорів

## 3.2 Розробка алгоритмів процесів

Оскільки за технічним завданням, програма повинна адаптуватись під різну кількість процесорів, представляється алгоритм і – ї задачі.

**Задача і**

1. Якщо і = 1 то введення MX, MR.
2. Якщо і = P то введення МО, МК, a
3. Якщо і > 1 то прийняти MX, MR
4. Поки , якщо то надсилати MX, MR в задачі , де j = 1, 2, 4, 8, 16...



1. Якщо і < P то прийняти MO, MK, a
2. Поки , якщо то надсилати MO, MK, a в задачі , де k = 1, 2, 3, 4…



1. Обчислення 1



1. Якщо і < P то прийняти m i зробити обчислення 2



1. Поки , якщо то надсилати m в задачі , де k = 1, 2, 3, 4…



1. Якщо і > 1 то прийняти m
2. Поки , якщо то надсилати m в задачі , де j = 1, 2, 4, 8, 16...



1. Обчислення 3



1. Поки , де j = 0, 1, 2…



1. Якщо i > 1, знайдемо таке мінімальне j щоб виконувалось і надсилаєм в задачу



1. Якщо i = 1 то виведення матриці MA

## 3.3 Розробка схеми взаємодії задач

Задачі взаємодіють через механізм рандеву. Кожна задача має набір входів, через які задачі обмінюються повідомленнями. Для відображення схеми взаємодії задачі, розглядається випадок 2-х вимірного гіперкуба. Оскільки при вимірах 3 і більше важко зрозуміти принцип взаємодії задач. Схема взаємодії задач представлена на рисунку 3.2.



Рис. 3.2 Схема взаємодії задач 2-х вимірного гіперкуба

## 3.4 Тестування програми ПРГ2

В даному розділі проводиться обчислення коефіцієнта прискорення та коефіцієнта ефективності для числа процесорів – 1, 2, 4, 8 при розмірностях матриць – 900, 1800, 2400.

Як і під час тестування програми ПРГ1 заміряється час розв’язання задачі та заноситься до таблиці 3.1

Таблиця 3.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N P | 1 | 2 | 4 | 8 |
| 900 |  |  |  |  |
| 1800 |  |  |  |  |
| 2400 |  |  |  |  |

В таблицю 3.2 занесено пораховані коефіцієнти прискорення.

Таблиця 3.2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N P | 1 | 2 | 4 | 8 |
| 900 |  |  |  |  |
| 1800 |  |  |  |  |
| 2400 |  |  |  |  |

На основі даних з таблиць 3.1 і 3.2 проведиться обчислення коефіцієнта ефективності, який розраховується за формулою.



де – коефіцієнт прискорення при P процесорах.



Розраховані коефіцієнти ефективності наведено в таблиці 3.3.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N P | 1 | 2 | 4 | 8 |
| 900 |  |  |  |  |
| 1800 |  |  |  |  |
| 2400 |  |  |  |  |

Таблиця 3.3

На основі даних з таблиць 3.1, 3.2, 3.3 будуються графіки залежності часу виконання від кількості процесорів, коефіцієнта прискорення від кількості процесорів, коефіцієнта ефективності від кількості процесорів.

Місце для графіка

Рис 3.1 Графік залежності часу виконання задачі від кількості процесорів

Місце для графіка

Рис. 3.2 Графік залежності коефіцієнта прискорення від кількості процесорів

Місце для графіка

Рис. 3.3 Графік залежності коефіцієнта ефективності від кількості процесора

## 3.5 Висновки до розділу 3

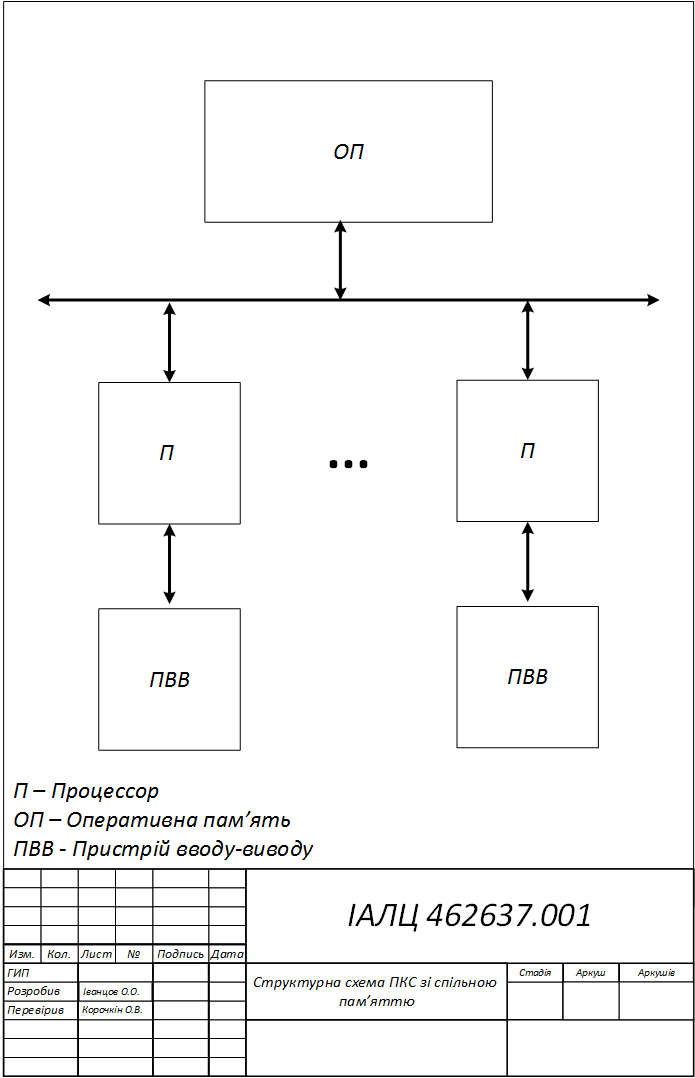
1. В третоьму розділі розроблено програму ПРГ2 для розв’язання математичної задачі в розподіленій комп’ютерній системі з локальною пам’яттю. Програма розроблена на мові програмування Ада.
2. Виходячи з резултатів тестування, можна зробити висновок що ефективність розв’язання задач на ПКС з локальною пам’яттю…

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

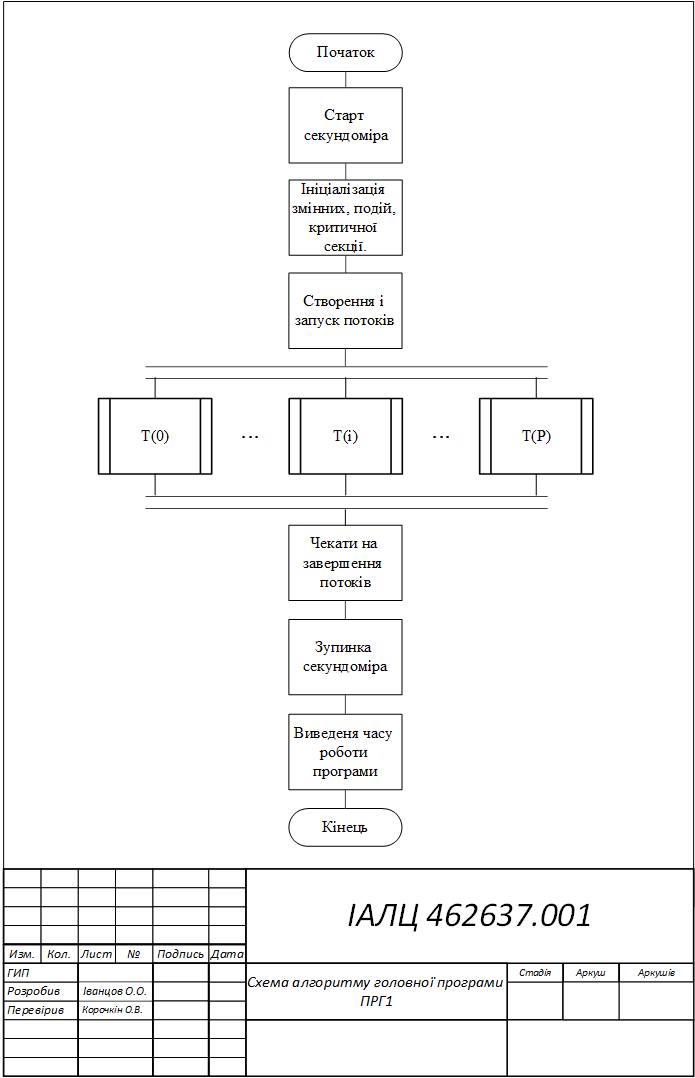
1. Группы и комуникаторы [Електронний ресурс]. Режим доступу: <https://parallel.ru/sites/default/files/info/parallel/antonov/groups.pdf>
2. Технологии параллельного программирования. Message Passing Interface (MPI) [Електронний ресурс]. Режим доступу: <http://parallel.ru/vvv/mpi.html>
3. [Процессы и потоки (нити)](http://www.moodle.ipm.kstu.ru/mod/page/view.php?id=49) [Електронний ресурс]. Режим доступу: <http://www.moodle.ipm.kstu.ru/mod/page/view.php?id=49>
4. Процессы, потоки, задачи [Електронний ресурс]. Режим доступу:

<http://www.4stud.info/rtos/lecture2.html>

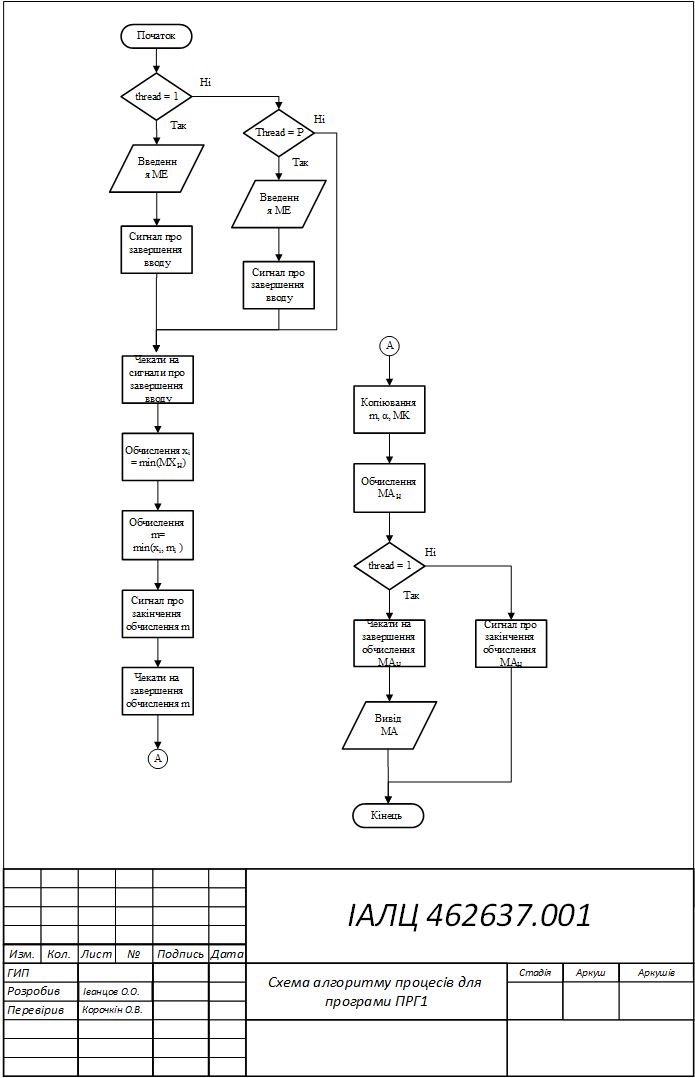
1. Материалы лекций В.П.Гергеля по MPI [Електронний ресурс]. Режим доступу: <http://www3.msiu.ru/~belova/par_prog/mpi_gergel.pdf>
2. Параллельное программирование на основе MPI [Електронний ресурс]. Режим доступу: <http://www.hpcc.unn.ru/mskurs/RUS/DOC/ppr04.pdf>
3. MPI для начинающих [Електронний ресурс]. Режим доступу: <http://www.opennet.ru/docs/RUS/MPI_intro/MPI_intro-prog.html.gz>
4. MPI: The Message Passing Interface [Електронний ресурс]. Режим доступу: <http://parallel.ru/tech/tech_dev/mpi.html>



# ДОДАТОК Б. БЛОК-СХЕМА АЛГОРИТМУ ОСНОВНОЇ ПРОГРАМИ



# ДОДАТОК В. БЛОК-СХЕМА ПОТОКОВОЇ ФУНКЦІЇ



# ДОДАТОК Г.

# ЛІСТИНГ ПРОГРАМИ ПРГ1

// Курсова робота

// ПРО

// Іванцов Олексанлр Олександрович

// ФІОТ, ІО-24

// MA = min(MX)\*MO + a\*(MK\*MR)

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

namespace cursova

{

class Program

{

static int N = 1800;

static int P = 4;

static int H = N / P;

static int a;

static volatile int m;

static int[][] MX, MR, MO, MK, MA;

static EventWaitHandle E\_1;

static EventWaitHandle [] eventCalculating2Finished;

static EventWaitHandle [] eventCalculating3Finished;

static object lockObject;

static void threadFunction(Object threadNumber)

{

int thread = (int)threadNumber;

System.Console.WriteLine("Thread "+ (thread+1) +" started!");

if (thread == 0)

{

//Введення MA, MX, MR

MX = Data.inputMatrix(N);

m = MX[0][0];

MR = Data.inputMatrix(N);

//Сигнал задачам Т2..ТР про введення даних

E\_1.Set();

}

if (thread == P - 1)

{

//Введення МО, МК, а

MO = Data.inputMatrix(N);

MK = Data.inputMatrix(N);

a = 1;

}

// Чекати сигналу від Т1 про введення даних

E\_1.WaitOne();

// Обчислення 1

int x = Data.minInMatrix(MX, thread\*H, (thread+1)\*H);

// Обчислення 2

m = Data.minValue(x, m);

// Сигнал задачам Т1..Ті-1, Ті+1..ТР про завершення обчислення 2

eventCalculating2Finished[thread].Set();

// Чекати сигналу від задач Т1..Ті-1, Ті+1..ТР про завершенян обчислення 2

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

eventCalculating2Finished[i].WaitOne();

// Копіювання mі = m, aі = a, MKі = MK

int mi, ai;

int[][] MKi;

lock (lockObject)

{

mi = m;

ai = a;

MKi = Data.copyMatrix(MK);

}

// Обчислення 3

Data.function(MA, mi, MO, ai, MK, MR, thread\*H, (thread+1)\*H);

if (thread == 0)

{

// Чекати сигналу від задач Т2..ТР про завершення обчислення 3

for (int i = 0; i < eventCalculating3Finished.Length; i++)

{

eventCalculating3Finished[i].WaitOne();

}

// Виведення МА

Data.outputMatrix(MA, "MA");

}

else

{

// Сигнал задачі Т1 про завершення обчислення 3

eventCalculating3Finished[thread - 1].Set();

}

System.Console.WriteLine("Thread " + (thread + 1) + " finished!");

}

static void Main(string[] args)

{

System.Console.WriteLine("Main thread started!");

Stopwatch stopWatch = new Stopwatch();

stopWatch.Start();

MX = new int[N][];

MR = new int[N][];

MO = new int[N][];

MK = new int[N][];

MA = new int[N][];

for (int i = 0; i < MK.Length; i++)

{

MO[i] = new int[MX.Length];

MK[i] = new int[MX.Length];

MA[i] = new int[MX.Length];

}

lockObject = new object();

E\_1 = new ManualResetEvent(false);

eventCalculating2Finished = new EventWaitHandle[P];

eventCalculating3Finished = new EventWaitHandle[P-1];

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

{

eventCalculating2Finished[i] = new ManualResetEvent(false);

if (i < P - 1)

{

eventCalculating3Finished[i] = new ManualResetEvent(false);

}

}

Thread thread = null;

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

{

thread = new Thread(threadFunction);

thread.Start(i);

}

thread.Join();

stopWatch.Stop();

// Get the elapsed time as a TimeSpan value.

TimeSpan ts = stopWatch.Elapsed;

// Format and display the TimeSpan value.

string elapsedTime = String.Format("{0:00}:{1:00}:{2:00}.{3:00}",

ts.Hours, ts.Minutes, ts.Seconds,

ts.Milliseconds / 10);

Console.WriteLine(" \n RunTime " + elapsedTime+"\n");

System.Console.WriteLine("Main thread finished!");

Console.Read();

}

}

}

using System;

using System.Collections.Generic;

using System.Threading.Tasks;

namespace cursova

{

class Data

{

public static int minInMatrix(int[][] matrix, int leftBound, int rightBound)

{

int buffer = matrix[0][0];

for (int i = leftBound; i < rightBound; i++)

{

for (int j = 0; j < matrix.Length; j++)

{

if (matrix[i][j] < buffer) buffer = matrix[i][j];

}

}

return buffer;

}

public static int minValue(int x, int y)

{

if (x < y) return x;

else return y;

}

public static void outputMatrix(int[][] matrix, String name)

{

if (matrix.GetLength(0) <= 10)

{

String matrixString = "";

for (int i = 0; i < matrix.Length; i++)

{

for (int j = 0; j < matrix.Length; j++)

{

matrixString += " " + matrix[i][j].ToString();

}

matrixString += " \r\n";

}

System.Console.WriteLine("\r\n Matrix " + name + " : \r\n" + matrixString);

}

}

public static int[][] inputMatrix(int size)

{

int[][] matrix = new int[size][];

for (int i = 0; i < matrix.Length; i++)

{

matrix[i] = new int[size];

for (int j = 0; j < matrix.Length; j++)

{

matrix[i][j] = 1;

}

}

return matrix;

}

public static void function(int[][] MA, int m, int[][] MO, int a, int[][] MK, int[][] MR, int LeftBound, int rightBound){

for (int i = 0; i < MR.Length; i++){

for (int j = LeftBound; j < rightBound; j++){

int buffer = 0;

for (int k = 0; k < MR.Length; k++){

buffer += MK[i][k] \* MR[k][j];

}

MA[i][j] = m \* MO[i][j] + a\*buffer;

}

}

}

public static int[][] copyMatrix(int[][] matrix){

int[][] result = new int[matrix.Length][];

for (int i = 0; i < matrix.Length; i++){

result[i] = new int[matrix.Length];

for (int j = 0; j < matrix.Length; j++){

result[i][j] = matrix[i][j];

}

}

return result;

}

}

}

# ДОДАТОК Д. ЛІСТИНГ ПРОГРАМИ ПРГ2

GNAT GPL 2013 (20130314)

Copyright 1992-2013, Free Software Foundation, Inc.

Compiling: C:\Users\Oleksandr\main.adb (source file time stamp: 2015-04-23 18:38:16)

1. -------------------------------------------

2. --Курсова робота ПРО

3. --Ада. Рандеву

4. --Іванцов О.О.

5. --MA = min(MX)\*MO + a\*(MK\*MR)

6. --22.04.2015

7. -------------------------------------------

8.

9. with Ada.TEXT\_IO; use Ada.TEXT\_IO;

10. with Ada.INTEGER\_TEXT\_IO; use Ada.INTEGER\_TEXT\_IO;

11. with Ada.Calendar; use Ada.Calendar;

12. with Ada.float\_text\_io; use Ada.float\_text\_io;

13.

14. procedure Main is

15. N: Integer := 4;

16. R: integer := 2;

17. P: Integer := 2\*\*R;

18. H: Integer := N / P;

19.

20. type Vector is array(Integer range<>) of Integer;

21. type Matrix is array(Integer range<>) of Vector(1..N);

22.

23.

24. procedure inputVector(V : out Vector) is

25. begin

26. for i in 1..N loop

27. V(i) := 1;

28. end loop;

29. end inputVector;

30.

31. procedure outputVector(V : in Vector) is

32. begin

33. for i in 1..N loop

34. Ada.Integer\_Text\_IO.Put ( V(i) );

35. end loop;

36. end outputVector;

37.

38. procedure inputMatrix(M : out Matrix) is

39. begin

40. for i in 1..N loop

41. inputVector(M(i));

42. end loop;

43. end inputMatrix;

44.

45. function min2(x, y : in Integer) return Integer is

46. begin

47. if(x > y) then

48. return y;

49. end if;

50. return x;

51. end min2;

52.

53. function minH(M : Matrix) return Integer is

54. buffer : Integer;

55. begin

56. buffer := M(1)(1);

57. for i in 1..H loop

58. for j in 1..N loop

59. if(M(i)(j) < buffer) then

60. buffer := M(i)(j);

61. end if;

62. end loop;

63. end loop;

64.

65. return buffer;

66. end minH;

67.

68. procedure calculating(MA : out Matrix; a, m :in Integer; MK: in Matrix; MR, MO : in Matrix) is

69. buf : Integer := 0;

70. begin

71. for i in 1..N loop

72. for j in 1..H loop

73. buf := 0;

74. for k in 1..N loop

75. buf := buf + MK(k)(i) \* MR(j)(k);

76. end loop;

77. MA(j)(i):= m\*MO(j)(i)+a\*buf;

78. end loop;

79. end loop;

80. end calculating;

81.

82. task type Tsk (id: Integer) is

83. entry SetData(MX, MR: in Matrix; idp : in Integer);

84. entry SetDataN(MO, MK : in Matrix; a : Integer);

85. entry SetMj(m : in Integer);

86. entry SetM(m: in Integer);

87. entry SetMA(MA : in Matrix; idp : in Integer);

88. end Tsk;

89.

90. type PTask1 is access Tsk;

91. T: array (1..P) of PTask1;

92.

93.

94. --------------------------------------------------------------------------------

95. task body Tsk is

96. MXi, MRi, bufMA : Matrix( 1..(P+1-id)\*H );

97. MKi: Matrix(1..N);

98. MOi: Matrix(1..id\*H);

99. MAi : Matrix(1..N);

100. index, buf : Integer := 0;

101. ai, mi: Integer;

102. begin

103. delay 1.5;

104. Put\_Line("Task is started!");

105.

106. if id = 1 then

107. inputMatrix(MXi);

108. inputMatrix(MRi);

109. end if;

110.

111. if id = P then

112. inputMatrix(MOi);

113. inputMatrix(MKi);

114. ai := 1;

115. end if;

116.

117.

118. if (id > 1) then

119. accept SetData(MX, MR: in Matrix; idp : in Integer) do

120. MXi := MX(((id - idp)\*H + 1)..(P+1-idp)\*H);

121. MRi := MR(((id - idp)\*H + 1)..(P+1-idp)\*H);

122. end SetData;

123. end if;

124.

125. while (id + 2\*\*index <= P) loop

126. if (2\*\*index >= id) then

127. T(id + 2\*\*index).SetData(MXi, MRi,id);

128. end if;

129. index := index+1;

130. end loop;

131.

132. if(id < P) then

133. accept SetDataN(MO, MK : in Matrix; a : Integer) do

134. MOi := MO(1..id\*H);

135. MKi := MK;

136. ai := a;

137. end SetDataN;

138. end if;

139.

140. index := 0;

141. buf:= P+1-id;

142. while(buf + 2\*\*index <= P) loop

143. if (2\*\*index >= buf) then

144. T(P+1-(buf + 2\*\*index)).SetDataN(MOi, MKi, ai);

145. end if;

146. index := index + 1;

147. end loop;

148.

149. mi := minH(MXi(1..H));

150.

151. if (id < P) then

152. accept SetMj(m : in Integer) do

153. mi := min2(mi, m);

154. end SetMj;

155. end if;

156.

157. index := 0;

158. buf:= P+1-id;

159. while (buf + 2\*\*index <= P) loop

160. if (2\*\*index >= buf) then

161. T(P+1-(buf + 2\*\*index)).SetMj(mi);

162. end if;

163. index := index+1;

164. end loop;

165.

166. if (id > 1) then

167. accept SetM(m: in Integer) do

168. mi := m;

169. end SetM;

170. end if;

171.

172. index := 0;

173. while (id + 2\*\*index <= P) loop

174. if (2\*\*index >= id) then

175. T(id + 2\*\*index).SetM(mi);

176. end if;

177. index := index+1;

178. end loop;

179.

180.

181.

182. calculating(bufMA, ai, mi, MKi, MRi(1..H), MOi(1..H));

183.

184. MAi( (id-1)\*H+1..id\*H) := bufMA(1..H);

185.

186. index:= 0;

187. while(2\*\*index < id) loop

188. index := index+1;

189. end loop;

190. while((2\*\*index+id) <= P) loop

191. accept SetMA(MA : in Matrix; idp: in Integer) do

192. MAi(((idp-1)\*H+1)..idp\*H) := MA(((idp-1)\*H+1)..idp\*H);

193. buf := 0;

194. while(2\*\*buf < P) loop

195. if (2\*\*buf >= idp) then

196. MAi(((idp + 2\*\*buf-1)\*H+1)..(idp + 2\*\*buf)\*H) := MA(((idp + 2\*\*buf-1)\*H+1)..(idp + 2\*\*buf)\*H);

197. end if;

198. buf := buf+1;

199. end loop;

200. end SetMA;

201.

202. index := index+1;

203. end loop;

204.

205. if (id > 1) then

206. buf:= 0;

207. while(2\*\*buf < id) loop

208. buf := buf+1;

209. end loop;

210. buf := buf - 1;

211. buf := id - 2\*\*buf;

212. T(buf).SetMA(MAi, id);

213. end if;

214.

215.

216. if (id=1) then

217. for i in 1..N loop

218. outputVector(MAi(i));

219. end loop;

220. end if;

221.

222. Put\_Line("Task is finished!");

223. end Tsk;

224.

225. task T0;

226. task body T0 is

227. begin

228. for i in 1..P loop

229. T(i) := new Tsk(i);

230. end loop;

231. end T0;

232.

233. begin

234. NULL;

235. end Main;

235 lines: No errors