МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ЧЕРНІГІВСЬКА ПОЛІТЕХНІКА

ПАРАЛЕЛЬНІ ТА РОЗПОДІЛЕНІ ОБЧИСЛЕННЯ

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт з дисципліни

“Паралельні та розподілені обчислення.”
для студентів напряму підготовки
123- "Комп'ютерна інженерія"

затверджено
на засіданні кафедри
інформаційних та комп'ютерних систем
Протокол №1 від 27.08.2021 р

Чернігів 2021

.

Паралельні та розподілені обчислення. Методичні вказівки до лабораторних робіт з дисципліни “Паралельні та розподілені обчислення.” для студентів напрямку підготовки 123 “Комп'ютерна інженерія”. / Укл. доц. В.А. Бичко – Чернігів: ЧДТУ, 20121. – 66 с. Укр. мовою.

Укладач:	Бичко В. А, кандидат фізико-математичних наук, доцент
Відповідальний за випуск:	Базилевич В.М., завідувач кафедри інформаційних та комп'ютерних систем, кандидат технічних наук, доцент
Рецензент:	Бівойно П.Г, кандидат технічних наук, доцент

ЗМІСТ

Вступ

Дисципліна "Паралельні та розподілені обчислення." відноситься до розряду дисциплін по вибору професійно-орієнтованого напряму "Комп'ютерна інженерія".

Необхідною передумовою для освоєння даної дисципліни є знання студентами таких навчальних курсів, як "Програмування".

Потреба розв'язання складних прикладних задач з великим обсягом обчислень і принципова обмеженість максимальної швидкодії «класичних» - за схемою фон Неймана - ЕОМ зумовили появу багатопроцесорних обчислювальних систем (БОС). Використання таких засобів обчислювальної техніки дозволяє істотно збільшувати продуктивність ЕОМ за будь-якого сучасного рівня розвитку комп'ютерного устаткування. Але при цьому потрібне «паралельне» узагальнення традиційної - послідовної технології розв'язування задач на ЕОМ. Так, чисельні методи у разі БОС мають проектуватися як системи паралельних і взаємодіючих процесів, що передбачають виконання на незалежних процесорах.

Мета навчальної дисципліни «Паралельні та розподілені обчислення» - створити теоретичне підґрунтя щодо паралельного програмування на підставі вивчення математичних моделей, методів і технологій паралельного програмування для багатопроцесорних обчислювальних систем.

Завдання навчальної дисципліни «Паралельні та розподілені обчислення» - формування у студентів теоретичних знань і практичних умінь розробляти, аналізувати, оптимізувати, настроювати на конкретну архітектуру обчислювальної системи паралельні програми використовуючи при цьому мову C++ та бібліотеку MPI.

За результатами вивчення навчальної дисципліни студенти повинні знати:

- технічні можливості сучасних обчислювальних систем, призначених для реалізації паралельних процесів і обчислень;
- методи та мовні механізмам конструювання паралельних програм;
- сучасні методи і засоби підтримки паралельного програмування;

уміти:

- розробляти паралельні програми використовуючи мову C++ та бібліотеку MPI;

аналізувати, оптимізувати, настроювати на конкретну архітектуру обчислювальної системи паралельні програми, розроблені мовою C++ та з використанням бібліотеки MPI.

Практикум містить чотрири лабораторні роботи, що охоплюють такі розділи: знайомство зі структурою MPI, процедури блокувального та

двоточкового обміну, знайомство з процедурами буферизованого обміну, процедури колективного обміну MPI, похідні типи та віртуальні топології в MPI. Кожна лабораторна робота містить короткий теоретичний матеріал, приклади, забезпечені необхідними коментарями, порядок виконання лабораторної роботи і варіанти індивідуальних завдань

1 Лабораторна робота №1

Знайомство з MPI. Двоточковий обмін даними

Мета. - набуття навичок застосування системи MPI для написання програм з можливістю паралельних обчислень.

1.1 Встановлення MPICH

1. Виконати інсталятор mpich2-L0.7-win32-ia32.msi , використовуючи права привілеями адміністратора. Для цього потрібно зайти в меню Пуск -> Програми -> Стандартні програми «Командний рядок», натисніть на неї правою кнопкою миші, і виберіть пункт «Запуск від імені адміністратора» (Рисунок 2). Підтвердити свої наміри і введіть пароль, якщо необхідно.

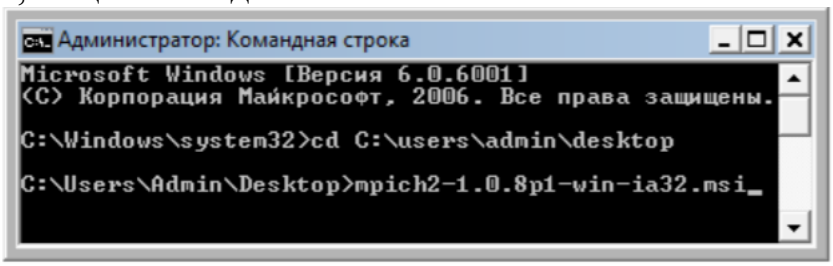

Рисунок 1.1 Запуск інсталятора з командного рядка

2. Ввести в командному рядку повний шлях до програми інсталяції і натиснути Enter (Рисунок 1.1).

к 4. Вказівка

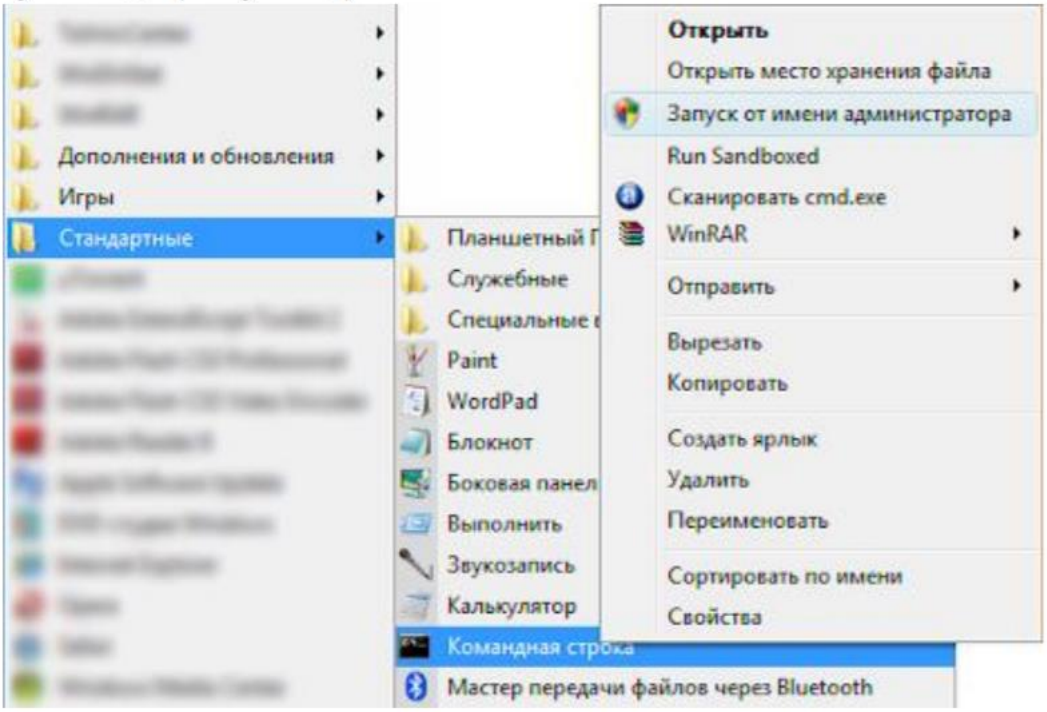

Рисунок 1.2. Вказівка

3. Під час установки вам треба буде ввести пароль «mpi_pass» для доступу до менеджера процесів SMPD. Необхідно ввести однаковий пароль на усіх комп'ютерах (Рисунок 4).

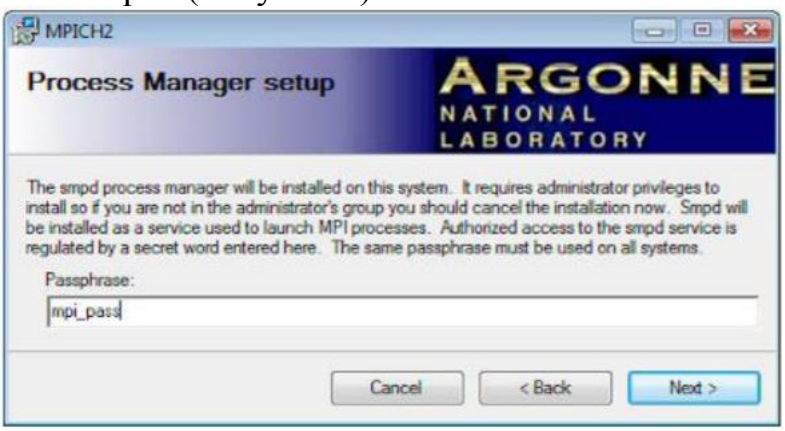

Рисунок 1.3. Вказівка пароля для доступу до менеджера процесів

4. У вікні вказівки шляху установки рекомендую залишити каталог за умовчанням. Крім того, поставите відмітку в пункті «Everyone» (Рисунок 5). Якщо Windows запитає, чи дозволити доступ в мережу програмі smpd.exe, то слід натиснути «Дозволити».

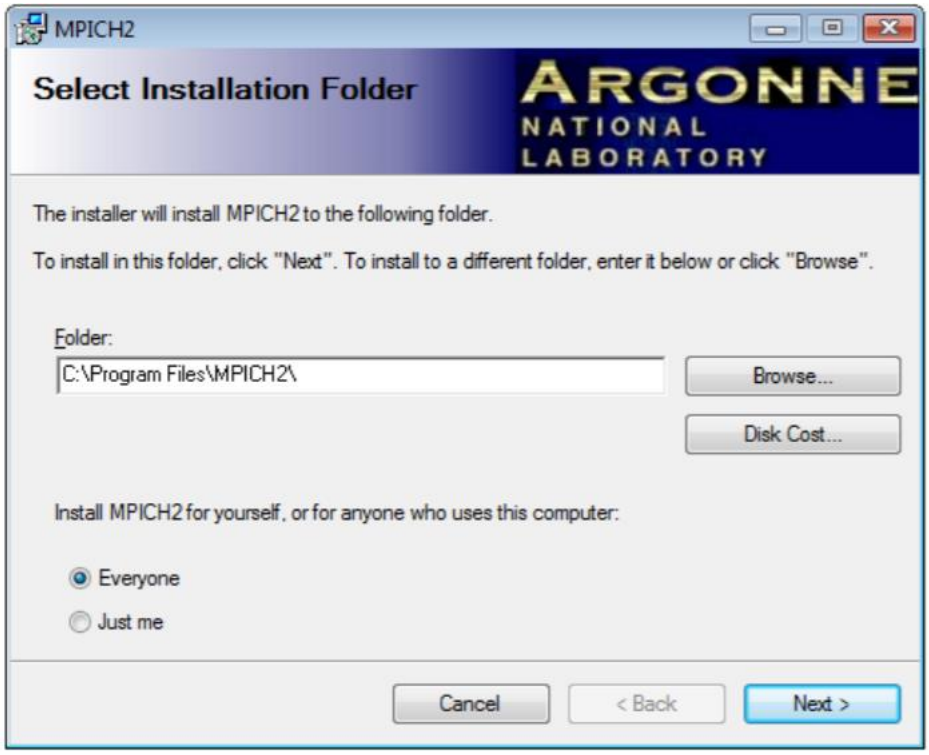

Рисунок 1.4. Вказівка шляху установки Проте, перш ніж переходити до налаштування, обов'язково слід перевірити дві речі: чи запущена служба «MPICH 2 Process Manager», і чи дозволений цій службі доступ в мережу. Для цього слід натиснути Пуск -> Налаштування -> Панель управління -> Адміністрування -> Служби. «MPICH2 Process Manager» повинен бути в списку служб (Рисунок 6). Ця служба повинна працювати. Якщо служба в списку відсутня, то запуск інсталятора не відбувся від імені адміністратора.

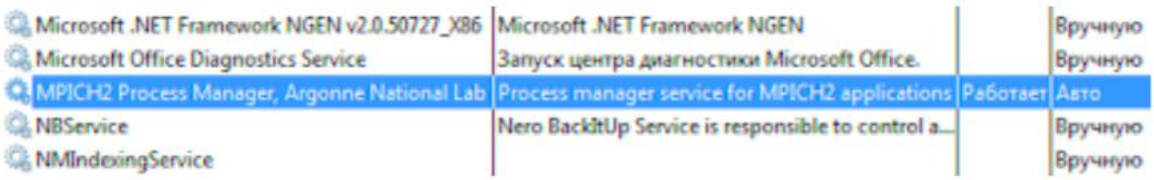

Рисунок 1,5. Служба «MPICH 2 Process Manager» в списку служб

Далі слід перевірити, чи дозволений доступ в мережу для MPICH. В меню Пуск -> Налаштування -> Панель управління -> Брандмауер Windows. слід натиснути «Дозвіл запуску програми через брандмауер Windows». В списку дозволених програм повинен бути «Process launcher for MPICH2 applications» і «Process manager service for MPICH2 applications» (Рисунок 1,6)

.Якщо якась з перерахованих програм відсутня в списку дозволених програм, то потрібно додати її вручну. Для цього слід натиснути кнопку «Додати програму.»., і додати C:\program files\mpich2\bin\mpiexec.exe, якщо відсутній «Process launcher for MPICH2 applications», і C:\program files\mpich2\bin\smpd.exe, якщо відсутній «Process manager service for MPICH2 applications».

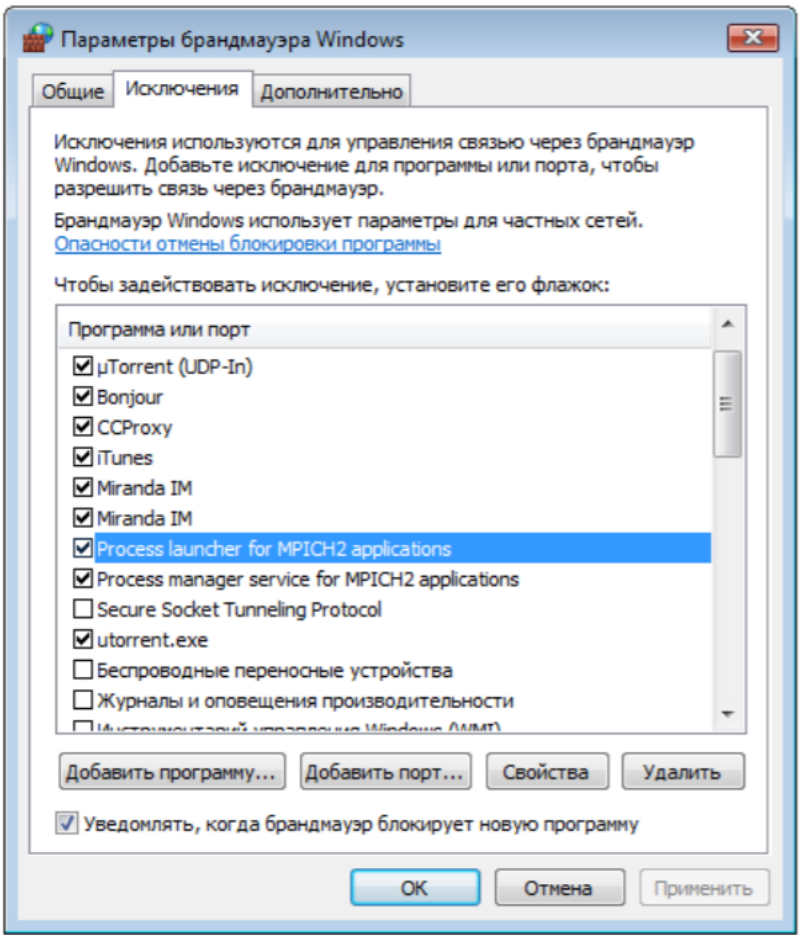

Рисунок 1.6. Програми MPICH в списку виключень брандмауера

Наступним етапом роботи є налаштування MPICH. Для цього потрібно виконати наступні пункти:

1) Необхідно створити обліковий запис на усіх комп'ютерах користувача з однаковим ім'ям і паролем; від імені цього користувача запускатимуться MPI - програми (якщо у вас один комп'ютер — цей крок можете пропустити). Найпростіше це зробити, встановивши однаковий пароль користувачам Адміністратор.

2) Щоб переконатися, що обліковий запис працює, і встановити на нього пароль, потрібно зайти в систему з обліковим записом «Адміністратор», і встановити пароль з допомогою Пуск -> Налаштування -> Панель управління -> Облікові записи користувачів -> Зміна свого пароля.

3) Як вже було сказано раніше, будь-яку дію система MPICH виконує від вказаного імені користувача. Для того, щоб запитувати ім'я користувача і пароль, використовується програма Wmpiregister. Проблема в тому, що ім'я користувача і пароль запитується досить часто. Для того, щоб цього уникнути, Wmpiregister може зберігати ім'я користувача і пароль в реєстрі Windows. Необхідно запустити Wmpiregister на тому комп'ютері, з якого буде запускатися MPI-програма. Для цього слід натиснути Пуск -> Програми -> MPICH2 -> wmpiregister.exe. Вікно програми наведено на рисунку 1.7.

Зміст кнопок (справа-наліво) :

- «Cancel» — закрити програму без виконання будь-якої дії.
- «OK» — передати введені ім'я користувача і пароль програмі. Якщо Wmpiregister запущена як окреме застосування, то натиснення кнопки OK еквівалентне натисненню кнопки Cancel.
- «Remove» — натиснення цієї кнопки видаляє збережені раніше ім'я користувача і пароль з реєстру Windows.
- «Register» — зберігає ім'я користувача і пароль в реєстрі.

Введіть ім'я користувача і пароль у вікні програми і натисніть кнопку «Register». Повинен з'явитися напис «Password encrypted into the Registry» (Рисунок 8). Після цього вікно програми більше не з'являтиметься при роботі з MPICH. Якщо потрібно видалити ім'я користувача і пароль з реєстру, то потрібно знову запустити цю програму, і натиснути кнопку «Remove».

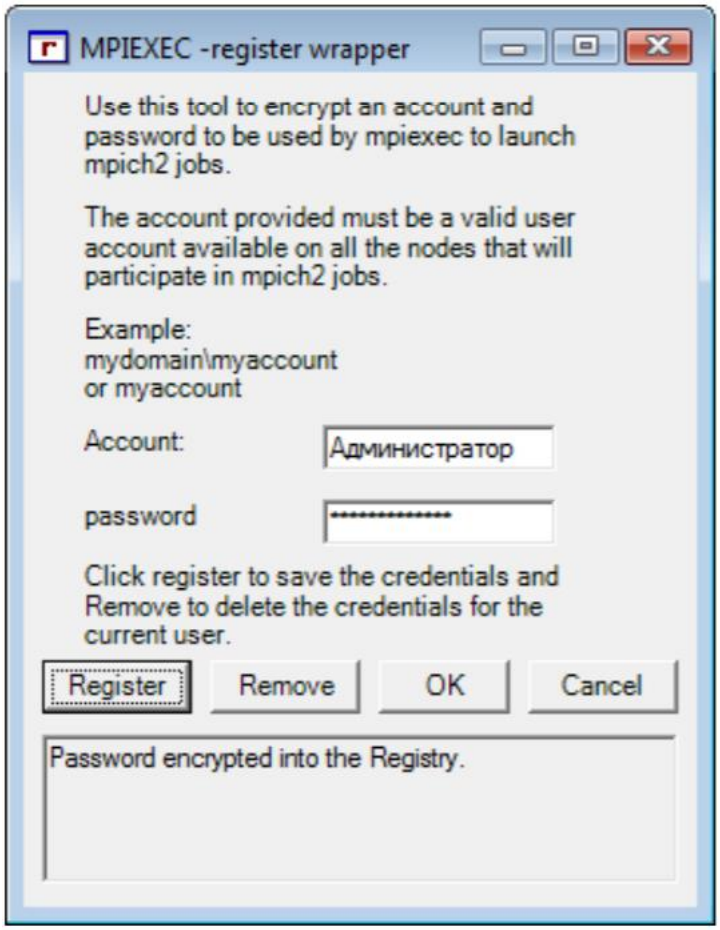

Рисунок 1.7. Програма Wmpiregister

4) Далі слід запустити на усіх комп'ютерах програму Wmpiconfig. Якщо усі попередні кроки зроблені правильно, то в полі «version» в лівій колонці таблиці повинна бути версія встановленого менеджера процесів (Рис.9). Якщо менеджер процесів не встановлений, або йому закритий доступ в мережу, то з'явиться повідомлення напис «MPICH 2 not installed or unable to query the host» в одному з полів лівого стовпця.

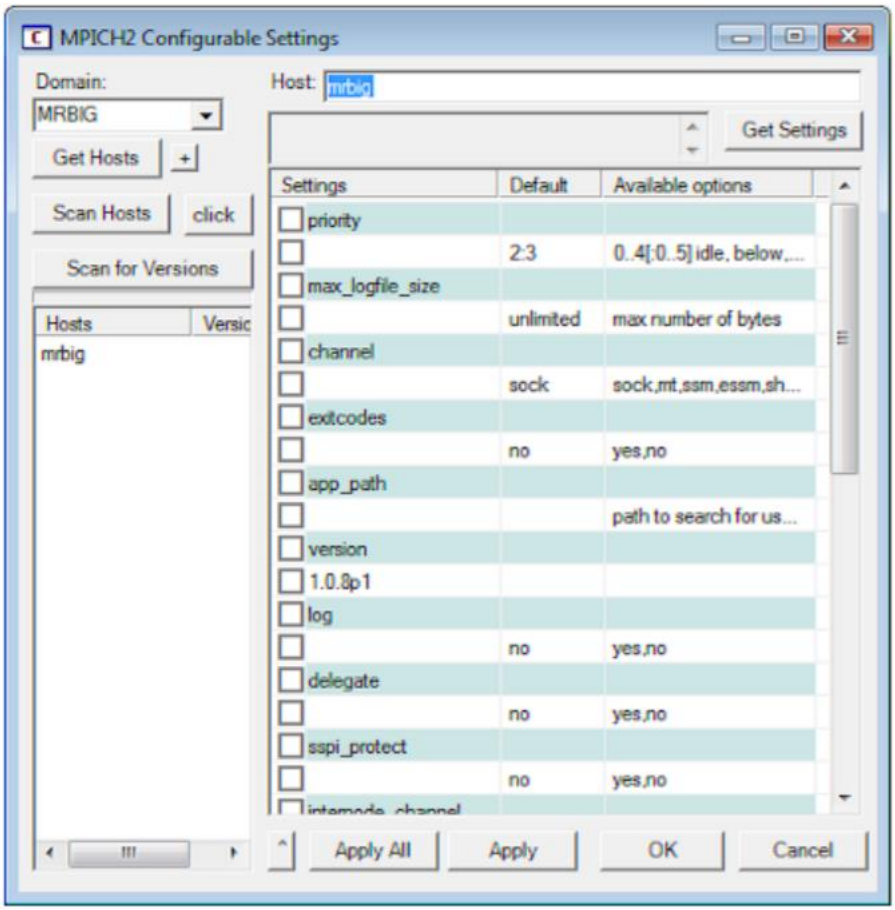

Рисунок 1.8. Програма Wmpiconfig

Програма Wmpiconfig призначена для налаштування менеджерів процесів на поточному комп'ютері та інших комп'ютерах мережі. Для цього вона під'єднується до менеджерів процесів на вибраних комп'ютерах, читає наявні у них налаштування, та повідомляє їм нові налаштування, якщо потрібно.

Елементи управління програми Wmpiconfig виконують наступні дії: - Ліворуч-внизу є список комп'ютерів, з якими працює програма налаштування. Ім'я комп'ютера на білому фоні означає, що не було спроб зв'язатися з цим комп'ютером; зелений фон означає, що зв'язок зроблений успішно; сірий фон означає, що при встановленні зв'язку виникла помилка.

- Видалити комп'ютер зі списку можна клавішею Del. Слід мати на увазі, що цей список призначений тільки для зручності налаштування, і не має ніякого відношення до списку комп'ютерів, на яких буде запущена MPI - програма.
- Кнопка «Get Hosts» отримує список комп'ютерів в заданому домені або робочій групі (задається у випадному списку «Domain»). Отриманий список замінює наявний список комп'ютерів або, якщо натиснута кнопочка «+», додає комп'ютери до поточного списку.
- Кнопка «Scan Hosts» отримує налаштування з усіх комп'ютерів списку; кнопка «Scan for Versions» отримує тільки номери версій.
- Кнопка «Get Settings» отримує поточні налаштування того комп'ютера, ім'я якого введене в поле введення «Host». При виборі

комп'ютера в списку комп'ютерів його ім'я автоматично вводиться в поле «Host». Якщо натиснута кнопка «Click», то налаштування будуть отримані автоматично при виборі комп'ютера зі списку.

- Справа у вікні розташована таблиця налаштувань. Якщо Ви хочете змінити які-небудь налаштування, то треба двічі клацнути на відповідному полі в першому стовпці таблиці. Порожнє поле означає, що використовується налаштування за умовчанням, вказана в другому стовпці. Налаштування, призначені до зміни, слід відмічати установкою опції ліворуч.
- Кнопка «Apply» застосовує виділені опцією налаштування до того комп'ютера, ім'я якого знаходиться в полі «Host». Кнопка «Apply All» застосовує налаштування до усіх комп'ютерів списку.
- Кнопка «Cancel» закриває програму. Наскільки я зрозумів, дія кнопки «OK» нічим не відрізняється від дії кнопки «Cancel».

Для успішного виконання потрібно, щоб імена комп'ютерів містили тільки латинські букви і цифри. Для того, щоб змінити ім'я, натисніть правою кнопкою миші на «Комп'ютер» -> Властивості -> Додаткові параметри системи -> Ім'я комп'ютера -> Змінити...

5) На тому комп'ютері, з якого планується запуск програм, треба вказати список доступних обчислювальних вузлів (якщо використовується тільки один комп'ютер — переходьте до пункту «запуск MPI — програм»). Цей список треба ввести (через пропуск) в поле hosts лівого стовпця таблиці (Рисунок 1.9), і натиснути кнопку «Apply».

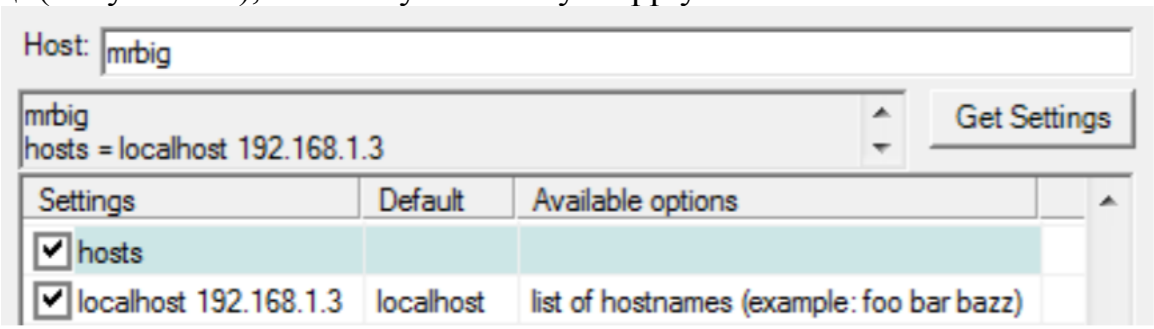

Рисунок 1.9. Вказуємо список доступних обчислювальних вузлів

6) Для зручного запуску MPI-програм слід створити на одному з комп'ютерів загальний мережевий ресурс. Якщо MPI-програми будуть запущені на одному комп'ютері, можна пропустити цей етап. Створіть папку (наприклад «Lab_Ivanov_MPI»), в яку ви викладатимете MPI - програми, натисніть на неї правою кнопкою миші, і виберіть «Властивості».

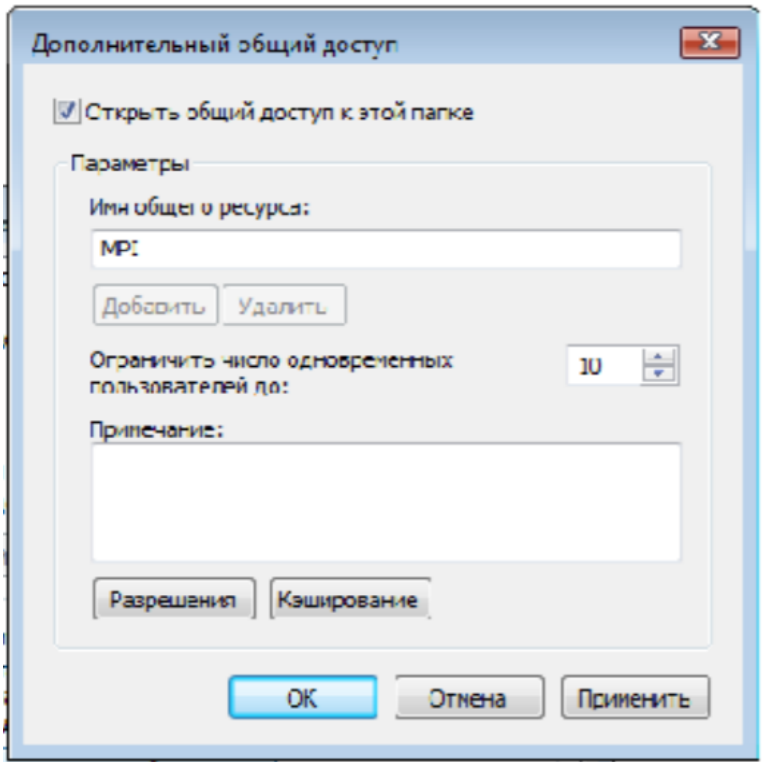

Рисунок 1.10. Вікно властивостей папки

7) У вікні, що з'явилося, виберіть вкладку «Доступ» (Рисунок 1.10), в якій натисніть кнопку «Додатковий доступ». У вікні, що з'явилося, «Додатковий загальний доступ» поставте опцію «Відкрити загальний доступ до цієї теки», і встановіть «число одночасних користувачів» таким, щоб воно перевищувало кількість комп'ютерів мережі, призначених для запуску MPI -програм.

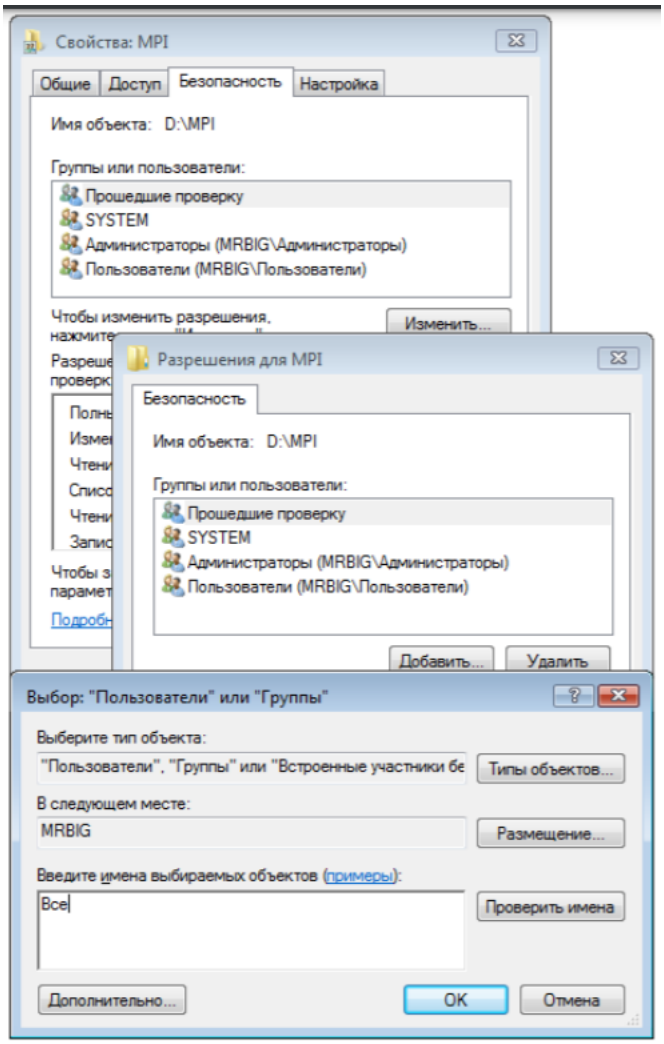

Рисунок 1.11. Додавання дозволів для доступу до теки

8. Натисніть «OK» у вікні «Додатковий загальний доступ» і перейдіть у вкладку «Безпеку». Там натисніть кнопку «Змінити» (Рисунок 12), з'явиться вікно «Дозволу для [ім'я теки]». У цьому вікні натисніть кнопку «Додати.»., з'явиться вікно вибору об'єкту для додавання (користувача або групи). Введіть в поле введення «Усі» (чи «All», якщо у вас англійська версія Windows) .Натисніть кнопку «OK» в двох вікнах. У вкладці «Безпека» у верхньому списку повинна додатися рядок «Все», при виборі якої в списку дозволів повинні стояти опції навпроти пунктів «Читання і виконання», «Список вмісту теки» і «Читання».

9. Останнім етапом є запуск MPI -програми. Для цього в комплект MPICH2 входить програма з графічним інтерфейсом Wmpiexec, яка є оболонкою навколо відповідної утиліти командного рядка Mpiexec. Вікно програми Wmpiexec показане на Рисунку 13 (зверніть увагу, що включений прапорець «more options»).

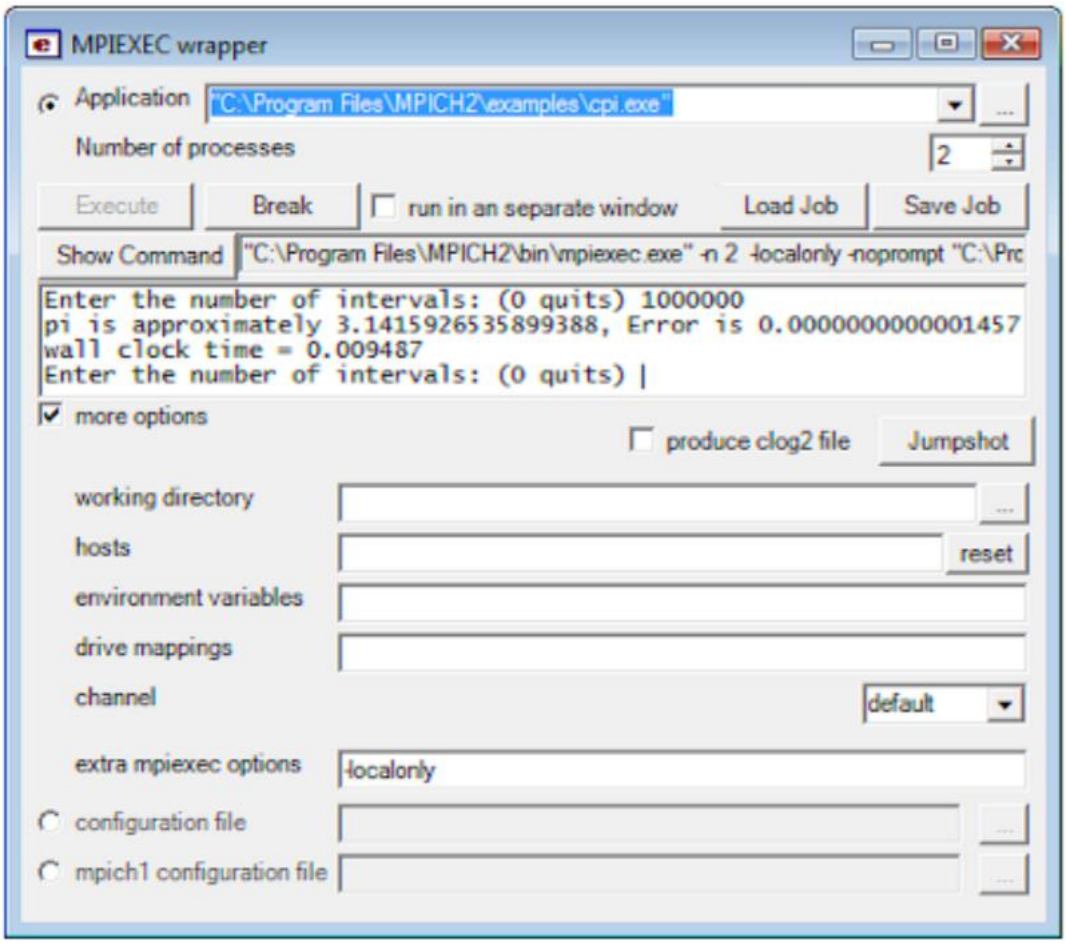

Рисунок 1.12. Програма Wmpiexec

Елементи управління вікна мають наступний зміст:

- Поле введення «Application» : сюди вводиться шлях до MPI -програми. Як вже було сказано раніше, шлях передається в незмінному вигляді на усі комп'ютери мережі, тому бажано, щоб програма розташовувалася в загальній мережевій папці.
- «Number of processes»: число процесів, що запускаються. По замовчуванню процеси розподіляються порівну між комп'ютерами мережі, проте цю поведінку можна змінити за допомогою конфігураційного файлу.
- Кнопка «Execute» запускає програму; кнопка «Break» примусово завершує усі запущені екземпляри.
- Прапорець «run in a separate window» перенаправляє виведення усіх екземплярів MPI -програми в окреме консольне вікно.
- Кнопка «Show Command» показує в полі справа командний рядок, який використовується для запуску MPI -програми (Wmpiexec — усього лише оболонка над Mpiexec). Командний рядок складається з усіх налаштувань, введених в інших полях вікна.
- Далі йде велике текстове поле, в яке потрапляє уведення-виведення усіх екземплярів MPI-програми, якщо не встановлений прапорець «run in a separate window».
- Прапорець «more options» показує додаткові параметри.

- «working directory»: сюди можна ввести робочий каталог програми. Знову ж таки, цей шлях має бути вірний на усіх обчислювальних вузлах. Якщо шлях не вказаний, то як робочий каталог

використовуватиметься місце знаходження MPI -програми.

- «hosts»: тут можна вказати через пропуск список

обчислювальних вузлів, використовуваних для запуску MPI - програми. Якщо це поле порожнє, то використовується список, що зберігається в налаштуваннях менеджера процесів поточного вузла (дивіться розділ

"Налаштування MPICH").

- «environment variables»: в цьому полі можна вказати значення додаткових змінних оточення, що встановлюються на усіх вузлах на час запуску MPI - програми. Синтаксис наступний: імя1=значення1, імя2=значення2.
- «drive mappings»: тут можна вказати мережевий диск, що підключається на кожному обчислювальному вузлі на час роботи MPI - програми. Синтаксис: Z:\\winsrv\wdir.
- «channel»: дозволяє вибрати спосіб передання даних між екземплярами MPI -програми.
- «extra mpiexec options»: в це поле можна ввести додаткові ключі для командного рядка Mpiexec.

1.2 Налаштування MPI -програми в Visual Studio

У Visual Studio 2010 налаштування каталогів перенесені у властивості проекту. Це означає, що спочатку треба створити проект, а потім вже настроїти для нього каталоги у вікні Project -> Properties. Передусім, треба налаштувати Visual Studio, щоб він знаходив заголовні файли і .lib-бібліотеки MPICH. Для цього запустіть Visual Studio і натисніть Tools-> Options, в дереві ліворуч виберіть Projects and Solutions -> VC++ Directories. Справа-вгорі виберіть Show directories for: Include files. Натисніть кнопку «Add» і додайте шлях до .h -файлів:

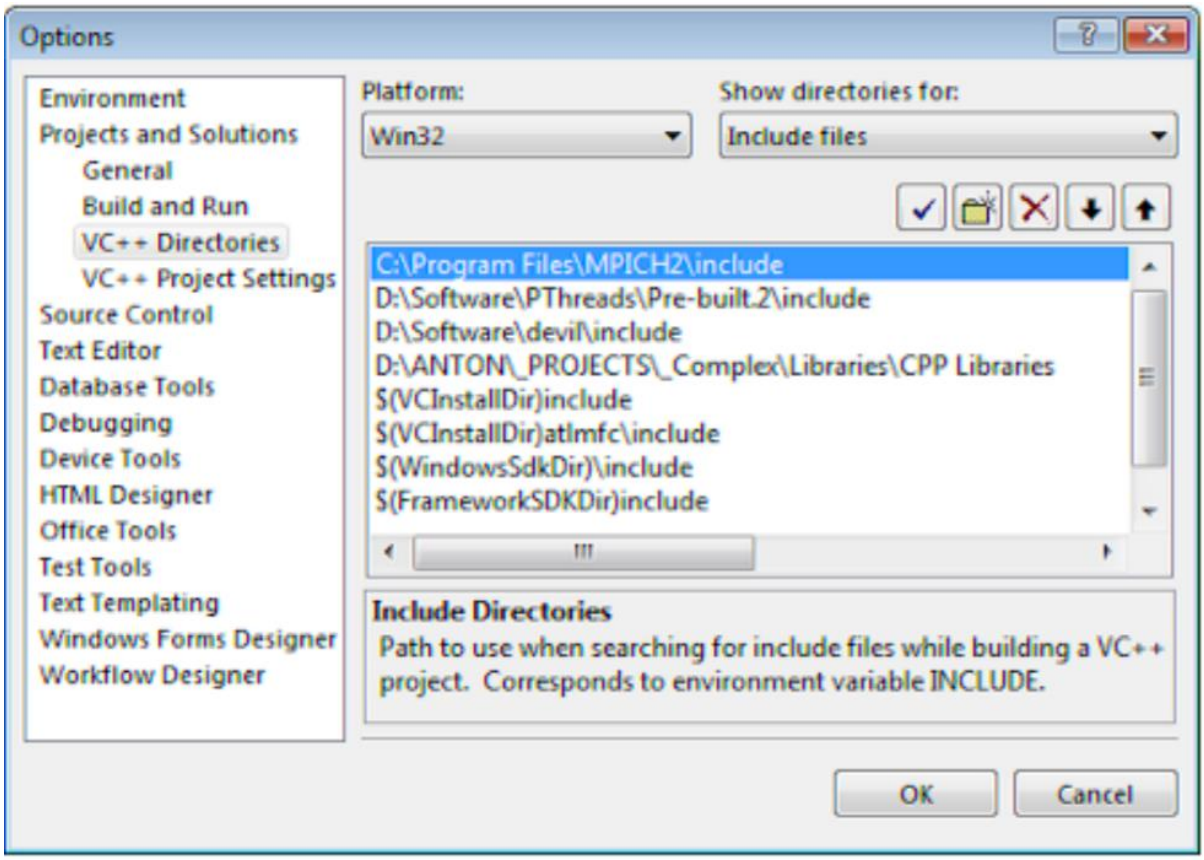

Рисунок 1.13. Налаштування шляху до заголовних файлів MPICH

Після цього слід виконати ту ж процедуру для бібліотек (Show directories for: Library files), рисунок 15.

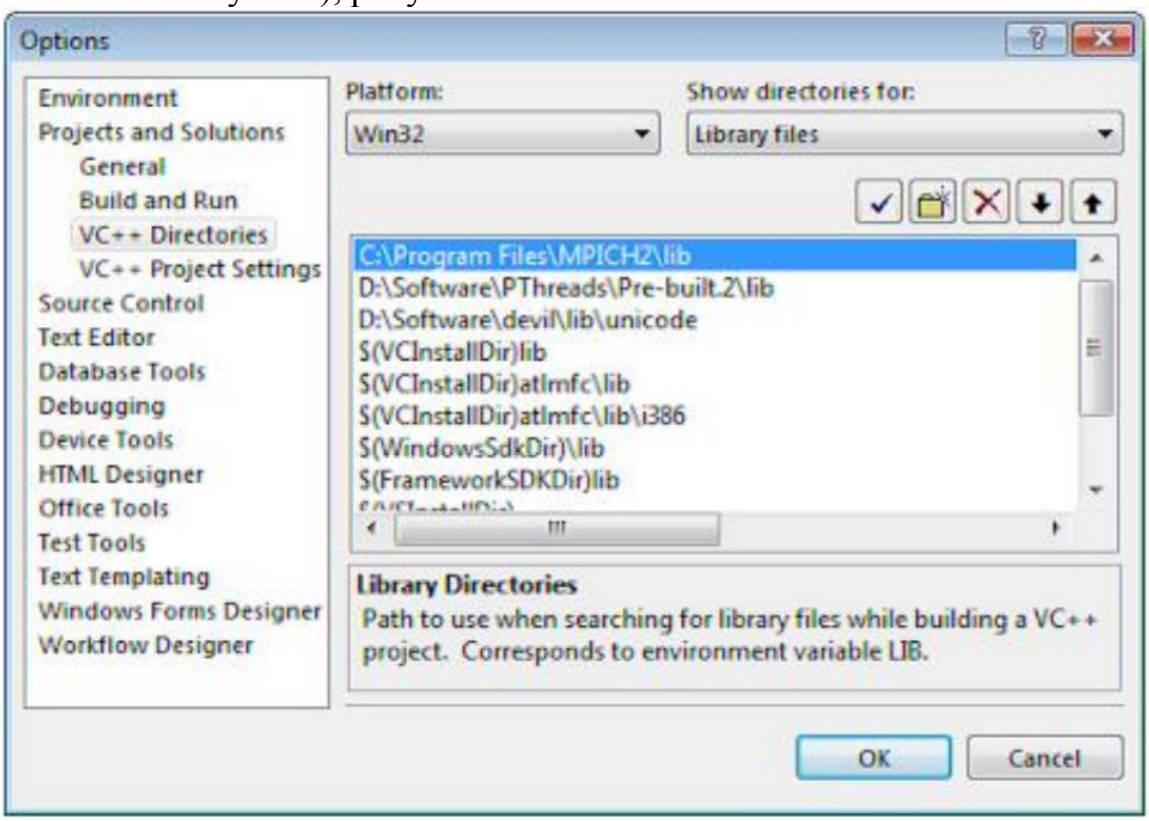

Рисунок 1.14. Налаштування шляху до бібліотечних файлів MPICH

Відкрийте вікно налаштувань проекту (Project-> Properties), виберіть Configuration: All Configurations, в дереві ліворуч виберіть Configuration справа.

Далі в полі Additional Dependencies, яке знаходиться в Properties -> Linker -> Input додайте mpi.lib. В список додаткових бібліотек для компонування окрім mpi.lib слід включати cxx.lib. Тому, якщо компонувальник (linker) видає помилку, спробуйте замість mpi.lib у вікні налаштувань, показаному на Рисунку 16, написати mpi.lib cxx.lib (через пропуск)

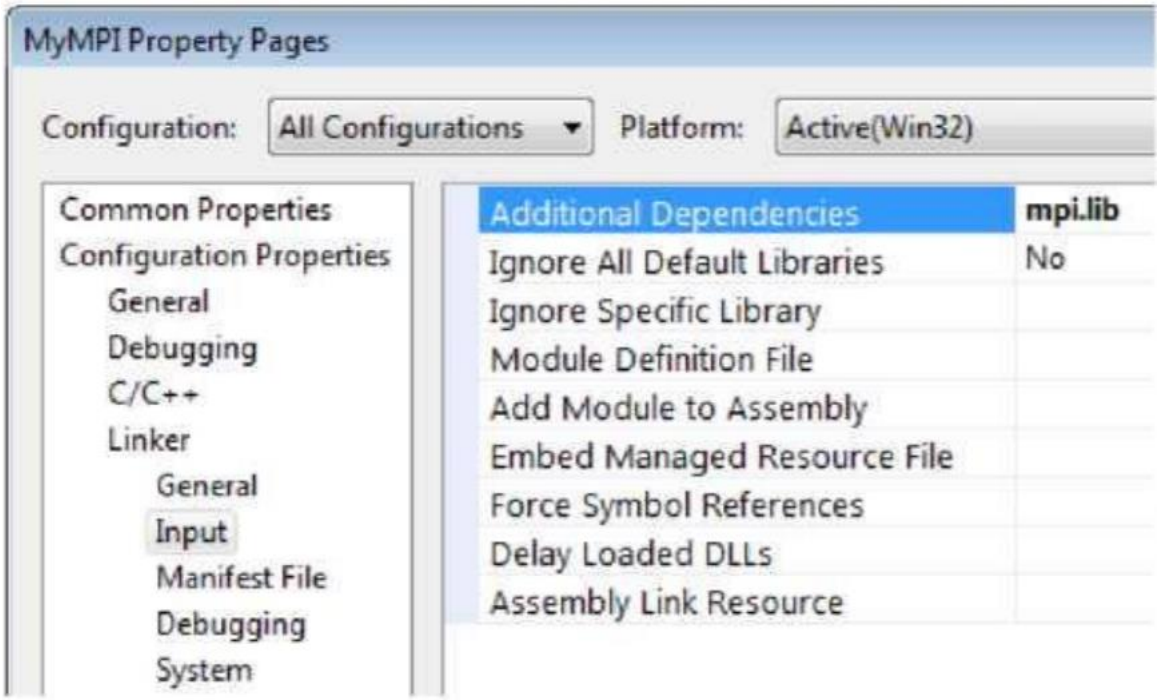

Рисунок 1.15. Додавання mpi.lib до програми

Тепер створіть консольний проект відповідно до рисунків 17,18.

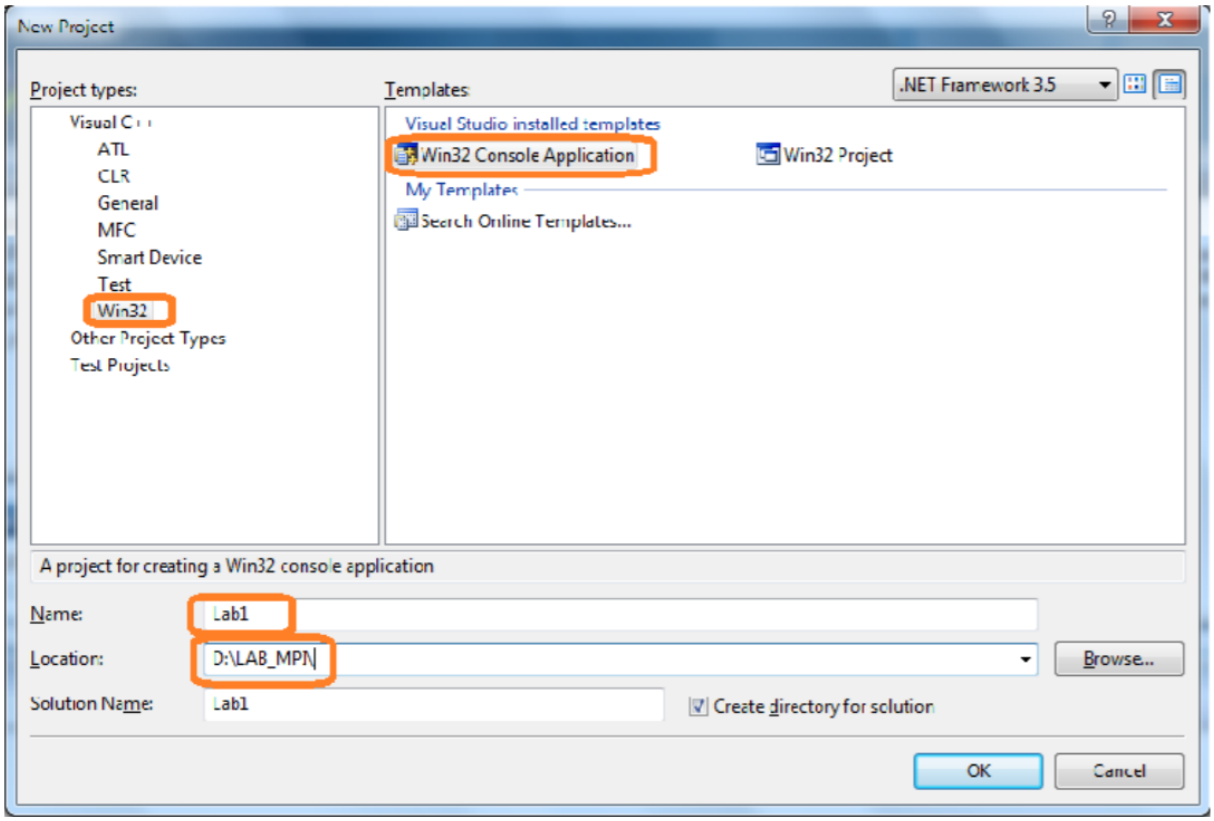

Рисунок 1.16. Створення нового проект

1.3 Модель паралельної програми в MPI

У моделі програмування MPI паралельна програма із запуском породжує кілька процесів, що взаємодіють між собою за допомогою повідомлень. Сукупність усіх процесів, що становлять паралельний додаток, або їх частини, описується спеціальною структурою, яка називається комунікатором (областю взаємодії).

Кожному процесу в області взаємодії призначається унікальний числовий ідентифікатор - ранг, значення якого від 0 до np - 1 (np - число процесів). Ранги, які призначаються одному й тому ж процесу в різних комунікаторах загалом різні.

Структура програми, написаної за схемою господар/працівник:

program para
if (ранг процесса = рангу мастер-процесса) then
код мастер-процесса
else
код подчиненного процесса (подчиненньїх процессов)
endif
end

1.4 Повідомлення

Повідомлення містять дані та службову інформацію, що пересилаються. Для того, щоб передати повідомлення, необхідно вказати: ранг процесу- відправника повідомлення; адресу, за якою розміщуються дані процесу- відправника, що пересилаються; тип даних, що пересилаються; кількість даних; ранг процесу, який повинен отримати повідомлення; адреса, за якою мають бути розміщені дані процесом-одержувачем; тег повідомлення; ідентифікатор комунікатора, що описує область взаємодії, усередині якої відбувається обмін.

Тег - це ціле число від 0 до 32767, що задається користувачем, яке виконує функцію ідентифікатора повідомлення і дозволяє розрізняти повідомлення, що надходять від одного процесу.

1.5Різновиди обмінів повідомленнями

У MPI реалізовані різні види обмінів:

1) двоточкові (задіяні тільки два процеси) - для організації локальних і неструктурованих комунікацій;

2) колективні (задіяні більше двох процесів) - для виконання глобальних операцій.

Різновиди двоточкового обміну:

- блокувальні прийом/передача - припиняють виконання процесу на час отримання повідомлення;
- не блокувальні прийом/передача - виконання процесу продовжується у фоновому режимі, а програма в потрібний момент може запросити підтвердження завершення прийому повідомлення;
- синхронний обмін - супроводжується повідомленням про закінчення прийому повідомлення;

асинхронний обмін - повідомленням не супроводжується.

1.6Коди завершення

У MPI прийняті стандартні угоди про коди завершення викликів підпрограм. Так, наприклад, повертаються значення MPI_SUCCESS - у разі успішного завершення виклику і MPI_ERR_OTHER - у разі спроби повторного виклику процедури MPI_Init.

Замість числових кодів у програмах використовують спеціальні іменовані константи:

- MPI_ERR_BUFFER - неправильний покажчик на буфер;
- MPI_ERR_COMM - неправильний комунікатор;
- MPI_ERR_RANK - неправильний ранг;

- MPI_ERR_OP - неправильна операція;
- MPI_ERR_ARG - неправильний аргумент;
- MPI_ERR_UNKNOWN - невідома помилка;
- MPI_ERR_TRUNCATE - повідомлення обрізано під час прийому;
- MPI_ERR_INTERN - внутрішня помилка. Зазвичай виникає, якщо системі не вистачає пам'яті.

1.7 Основні комунікатори MPI

Комунікатор - структура, що містить або всі процеси, які виконуються в рамках певної програми, або їх підмножину. Процеси, що належать одному і тому ж комунікатору, наділяються загальним контекстом обміну. Операції обміну можливі тільки між процесами, пов'язаними із загальним контекстом, тобто, належать одному і тому ж комунікатору. Кожному комунікатору присвоюється ідентифікатор. У MPI є кілька стандартних комунікаторів:

- MPI_COMM_WORLD - включає всі процеси паралельної програми;
- MPI_COMM_SELF - включає тільки даний процес;
- MPI_COMM_NULL - порожній комунікатор, не містить жодного процесу.

У MPI є процедури, що дозволяють створювати нові комунікатори, містять підмножини процесів.

1.8Двоточкові блокувальні обміни

Учасниками двоточкового обміну є два процеси: процес-відправник і процес-одержувач. Блокувальні операції двоточкового обміну призупинюють виконання виклику процесу. Він переходить у стан очікування завершення передачі даних. Блокування гарантує виконання дій у заданому порядку, забезпечуючи передбачуваність поведінки програми. З іншого боку, вона створює умови для виникнення кутових ситуацій, коли обидва процеси- учасника обміну блокуються одночасно.

1.9 Порядок виконання роботи

1. Вивчити необхідний для виконання роботи теоретичний матеріал.
2. Встановити необхідне для виконання ПО.
3. Підключити ПО до Visual Studio.

4. Відкрийте середовище Visual Studio та, підключивши бібліотеки MPI, перевірте роботу програм.

1. Підключення до MPI

int MPI_Init (int * argc, char ** argv)

Аргументи argc і argv потрібні лише у програмах на C, де вони задають кількість аргументів командного рядка запуску програми і вектор цих аргументів. Цей виклик передує всім іншим викликам підпрограм MPI.

2. Завершення роботи з MPI

int MPI_Finalize () Після виклику цієї підпрограми не можна викликати підпрограми MPI.

3. Визначення розміру області взаємодії

int MPI_Comm_size (MPI_Comm comm, int * size) Вхідні параметри:

- comm - комунікатор. Вихідні параметри:
- size - кількість процесів в області взаємодії.

4. Визначення рангу процесу

int MPI_Comm_rank (MPI_Comm comm, int * rank) Вихідні параметри:

- rank - ранг процесу в області взаємодії.

5. Визначення назви вузла, на якому виконується процес

MPI_Get_processor_name (char * name, int * resultlen) Вихідні параметри:

- name - ідентифікатор обчислювального вузла. Масив не менше ніж MPI_MAX_PROCESSOR_NAME елементів;
- resultlen - довжина імені.

6. Час, що минув з довільного моменту в минулому

double MPI_Wtime ()

Далі наводиться опис інтерфейсу підпрограм, що реалізують різні види двоточкового обміну.

7. Стандартна блокувальна передача

int MPI_Send (void * buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)

8. Стандартний блокувальний прийом

int MPI_Recv (void * buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status * status)

Вхідні параметри:

- count - максимальна кількість елементів у буфері прийому. Фактичну їх кількість можна визначити за допомогою підпрограми MPI_Get_count;

datatype - тип даних, що приймаються. Нагадаємо про необхідність дотримання відповідності типів аргументів підпрограм прийому та передачі;

- source - ранг джерела. Можна використовувати спеціальне значення MPI_ANY_SOURCE, відповідне довільному значенню рангу. У

програмуванні ідентифікатор, що відповідає довільному значенню параметра, часто називають «джокером».

Вихідні параметри:

- buf - початкова адреса буфера прийому. Його розмір має бути достатнім, щоб розмістити прийняте повідомлення, інакше під час виконання прийому відбудеться збій - виникне помилка переповнення;

status - статус обміну.

Якщо повідомлення менше, ніж буфер прийому, змінюється вміст лише тих елементів пам'яті буфера, що стосуються повідомлення.

9. Визначення розміру отриманого повідомлення (count)

int MPI_Get_count (MPI_Status * status, MPI_Datatype datatype, int * count)

Аргумент datatype повинен відповідати типу даних, зазначеному в операції передачі повідомлення.

Синхронна передача

int MPI_Ssend (void * buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)

Параметри цієї підпрограми збігаються з параметрами підпрограми MPI_Send.

1.10 Завдання для самостійної роботи

1. У вихідному тексті програми мовою С пропущені виклики процедур підключення до MPI, визначення кількості процесів і рангу процесу. Додати ці виклики, відкомпілювати і запустити програму.

```
#include "mpi.h" #include <stdio.h>
int main(int argc,char *argv[]) {
int myid, numprocs;
fprintf(stdout,"Process %d of %d\n", myid,numprocs); MPI_Finalize();
return 0; }
```

2. У вихідному тексті програми мовою С пропущені виклики процедур стандартного блокувального двоточкового обміну. Передбачається, що під час запуску двох процесів один з них відправляє повідомлення іншому. Додати ці виклики, відкомпілювати і запустити програму.

```
#include "mpi.h" #include <stdio.h>
int main(int argc,char *argv[]) {
int myid, numprocs; char message[20]; int myrank; MPI_Status status; int TAG = 0; MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
if (myrank == 0) {
strcpy(message, "Hi, Second Processor!");
MPI_Send(...); }
else {
```

```
MPI_Recv(...);
printf("received: %s\n", message); }
MPI_Finalize();
return 0; }
```

3. У вихідному тексті програми мовою С пропущені виклики процедур стандартного блокувального двоточкового обміну. Передбачається, що під час запуску парної кількості процесів, ті з них, які мають парний ранг, відправляють повідомлення наступним за величиною рангу процесам. Додати ці виклики, відкомпілювати і запустити програму.

```
#include "mpi.h" #include <stdio.h>
int main(int argc,char *argv[]) {
int myrank, size, message; int TAG = 0; MPI_Status status; MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank); MPI_Comm_size(MPI_COMM_WORLD, &size); message = myrank;
if((myrank % 2) == 0) {
if((myrank + 1) != size)
MPI_Send(...); }
else {
if(myrank != 0) MPI_Recv(...);
printf("received :%i\n", message);
}
MPI_Finalize();
return 0;
}
```

1.11 Вимоги до звіту

Звіт повинен містити:
- Титульну сторінку з даними про виконавця і перевіряючого.
- Тему і мету роботи.
- Теоретична частина.
- Постановка завдання і алгоритм його розв'язування.
- Лістинг програми мовою С++.
-Висновки за результатами лабораторної роботи.

Звіт повинен бути оформлений відповідно до вимог СОКР/

1.12 Контрольні питання

1. Як поводиться паралельна програма в MPI під час запуску?
2. Яку структуру називають комунікатором?
3. Що треба зробити, щоб передати повідомлення?

4. Що називають тегом?
5. Які є різновиди двоточкового обміну?
6. Що повертає MPI під час успішного завершення виклику?
7. Назвіть стандартні комунікатори MPI?
8. Яка процедура використовується для визначення рангу процесу?

2 Лабораторна робота №2

Знайомство з процедурами буферизованого і не блокувального двоточкового обміну MPI

Мета. - Написати програми MPI з процедурами буферизованного і неблокувального двоточкового обміну

2.1Теоретичні відомості

Під час передачі повідомлення в буферизованому режимі джерело копіює повідомлення в буфер, а потім передає його в неблокувальному режимі. Виділення буфера і його розмір контролюються програмістом, який повинен заздалегідь створити буфер достатнього розміру. Буферизована передача завершується відразу, оскільки повідомлення негайно копіюється в буфер для подальшої передачі.

Після завершення роботи з буфером його необхідно відключити. Після відключення буфера можна знову використати займану ним пам'ять, проте слід пам'ятати, що в мові С цей виклик не звільняє автоматично пам'ять, відведену для буфера.

Буферизований обмін рекомендується використовувати в тих ситуаціях, коли програмісту потрібно більший контроль над розподілом пам'яті.

2.2Двоточкові неблокувальні обміни

Виклик підпрограми неблокувальної передачі ініціює, але не завершує її. Передача даних з буфера або їх зчитування відбувається одночасно з виконанням інших операцій. Завершується обмін викликом додаткової процедури, яка перевіряє, чи скопійовані дані у буфер передачі. До завершення обміну запис у буфер або зчитування з нього проводити не можна, оскільки повідомлення може бути ще не отримано чи отримано.

Неблокувальна передача може бути прийнята підпрограмою блокуючого прийому і навпаки.

Неблокувальний обмін виконується в два етапи:

1. Ініціалізація обміну.
2. Перевірка завершення обміну.

Для маркування неблокувальних операцій обміну використовуються ідентифікатори операцій обміну.

Перевірка фактичного виконання передачі або приймання в неблокувальному режимі здійснюється за допомогою виклику підпрограм очікування, які блокують роботу процесу до завершення операції або

неблокувальних підпрограм перевірки, які повертають логічне значення «істина», якщо операція виконана.

Підпрограма MPI_Wait блокує роботу процесу до завершення приймання або передачі повідомлення. Функції MPI_Wait і MPI_Test можна використовувати для завершення операцій приймання та передачі.

Хід роботи

2.3 Порядок виконання роботи

У середовищі Visual Studio перевірте роботу наступних підпрограм

пробників. Отримати інформацію про повідомлення до його розміщення в буфер приймання можна за допомогою підпрограм-пробників MPI_Probe та MPI_IProbe. На підставі отриманої інформації приймається рішення про подальші дії. За допомогою виклику підпрограми MPI_Probe фіксується надходження (але не приймання!) повідомлення. Потім визначається джерело повідомлення, його довжина, виділяється буфер відповідного розміру і виконується приймання повідомлення.

1. Буферизований обмін

int MPI_Bsend(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm) Параметри збігаються з параметрами підпрограми MPI_Send.

2. Створення буфера

int MPI_Buffer_attach (void * buf, size)

Вихідний параметр:

buf - буфер розміром size байтів. За один раз до процесу може бути підключений тільки один буфер.

3. Відключення буфера

int MPI_Buffer_detach (void * buf, int * size)

Вихідні параметри: buf - адреса;

size - розмір буфера, що відключається.

Виклик цієї підпрограми блокує роботу процесу до тих пір, поки всі повідомлення, що знаходяться в буфері, не будуть опрацьовані. У мові С цей виклик не звільняє автоматично пам'ять, відведену для буфера.

4. Передавання за готовністю

int MPI_Rsend (void * buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm) Параметри збігаються з параметрами підпрограми MPI_Send.

5. Блокувальна перевірка доставлення повідомлення

int MPI_Probe (int source, int tag, MPI_Comm comm, MPI_Status * status)

Вхідні параметри:

- source - ранг джерела або «джокер»;

- tag - значення тега або «джокер»;
- comm - комунікатор. Вихідний параметр:
- status - статус.

6. Неблокуюча перевірка повідомлення

int MPI_Iprobe (int source, int tag, MPI_Comm comm, int * flag, MPI_Status * status)

Вхідні параметри цієї підпрограми ті ж, що і у підпрограми MPI_Probe. Вихідні параметри: flag - прапор;

- status - статус.

Якщо повідомлення вже надійшло і може бути прийнято, повертається значення прапора «істина».

7. Приймання і передавання даних з блокуванням

int MPI_Sendrecv (void * sendbuf, int sendcount, MPI_Datatype sendtype, int dest, int sendtag, void * recvbuf, int recvcount, MPI_Datatype recvtype, int source, int recvtag, MPI_Comm comm, MPI_Status * status)

Вхідні параметри:

- sendbuf - початкова адреса буфера передавання;
- sendcount - кількість переданих елементів;
- sendtype - тип переданих елементів;
- dest - ранг адресата;
- sendtag - тег переданого повідомлення;
- recvbuf - початкова адреса буфера приймання;
- recvcount - кількість елементів у буфері приймання;
- recvtype - тип елементів у буфері приймання;
- source - ранг джерела;
- recvtag - тег прийнятого повідомлення;
- comm - комунікатор.

Вихідні параметри: - recvbuf

- status - статус операції приймання. Приймання і передавання використовують один і той же комунікатор. Буфери приймання і передавання не повинні перетинатися, у них може бути різний розмір, типи переданих і прийнятих даних також можуть різнитися.

2.4Завдання для самостійної роботи

1. У вихідному тексті програми мовою С пропущені виклики процедур буферизованого обміну. Додати ці виклики, відкомпілювати і запустити програму.

#include "mpi.h" #include <stdio.h>

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

int *buffer; int myrank; MPI_Status status; int buffsize = 1; int TAG = 0; MPI_Init(&argc, &argv);

```
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
if (myrank == 0) {
buffer = (int *) malloc(buffsize + MPI_BSEND_OVERHEAD);
buffer = (int *) 10; }
else {
MPI_Recv(&buffer, buffsize, MPI_INT, 0, TAG,
MPI_COMM_WORLD, &status);
printf("received: %i\n", buffer); }
MPI_Finalize();
return 0; }
```

2. У вихідному тексті програми мовою С пропущені виклики підпрограм- пробників. Додати ці виклики, відкомпілювати і запустити програму. #include "mpi.h" #include <stdio.h>

```
int main(int argc,char *argv[]) {
int myid, numprocs, **buf, source, i;
int message[3] = {0, 1, 2};
int myrank, data = 2002, count, TAG = 0;
MPI_Status status;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
if(myrank == 0) {
MPI_Send(&data, 1, MPI_INT, 2, TAG, MPI_COMM_WORLD); }
else if (myrank == 1) {
MPI_Send(&message, 3, MPI_INT, 2, TAG, MPI_COMM_WORLD); }
else {
source = status.MPI_SOURCE;
MPI_Get_count(...);
for (i = 0; i < count; i++){
buf[i] = (int *)malloc(count*sizeof(int)); }
MPI_Recv(&buf[0], count, MPI_INT, source, TAG,
MPI_COMM_WORLD,
&status);
for (i = 0; i < count; i++){
printf("received: %d\n", buf[i]); }
}
MPI_Finalize();
return 0; }
```

2.5Вимоги до звіту

Звіт повинен містити:

- Титульну сторінку з даними про виконавця і перевіряючого.
- Тему і мету роботи.

-Лістинг програми.

-Результати застосування створеної функції до тестового зображення.

-Висновки за результатами лабораторної роботи.

Звіт повинен бути оформлений відповідно до вимог СОКР/

2.6Контрольні питання

1. Як працює буферизований режим передавання повідомлення?
2. У яких ситуаціях рекомендується використовувати буферизований обмін?
3. Як виконується не блокувальний обмін?
4. Що робить підпрограма MPI_Wait?
5. Для чого використовуються підпрограми-пробники?
6. Яка функція використовується для створення буфера?
7. Які параметри приймає функція відключення буфера?
8. Яка функція використовується для буферизованого обміну?

3 Лабораторна робота №3

Знайомство з процедурами колективного обміну

Мета. - ознайомитись з процедурами та написати програми колективного обміну MPI.

3.1Теоретичні відомості Види колективних обмінів

Під час виконання колективного обміну повідомлення пересилається від одного процесу кільком чи навпаки, один процес збирає дані від декількох процесів. MPI підтримує такі види колективного обміну, як широкомовна передача, операції приведення (редукції), розподіл і збір даних і т. д.

Колективні обміни мають такі особливості:

- вони не можуть взаємодіяти з двоточковим;
- колективні обміни можуть виконуватися як із синхронізацією, так і без неї;
- усі колективні обміни є блокувальними для ініціатора їх обміну;
- теги повідомлень призначаються системою.

У колективному обміні бере участь кожен процес з деякої області взаємодії. Можна організувати обмін і в підмножині процесів, для цього є засоби створення нових областей взаємодії та відповідних їм комунікаторів.

3.2 Широкомовне розсилання

Широкомовне розсилання виконується виділеним процесом, який називається головним (root). Усі інші процеси, які беруть участь в обміні, отримують по одній копії повідомлення від головного процессу (рис. 1).

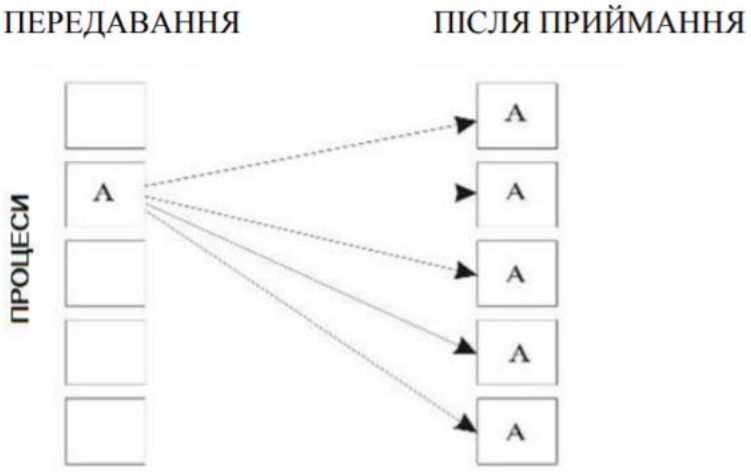

Рис. 1 - Широкомовне розсилання

Виконується широкомовна розсилка за допомогою підпрограми MPI_Bcast.

3.3Операції редукції

Операції редукції належать до категорії глобальних обчислень. У глобальній операції приведення до даних від усіх процесів із заданного комунікатора застосовується операція MPI_Reduce (рис. 2).

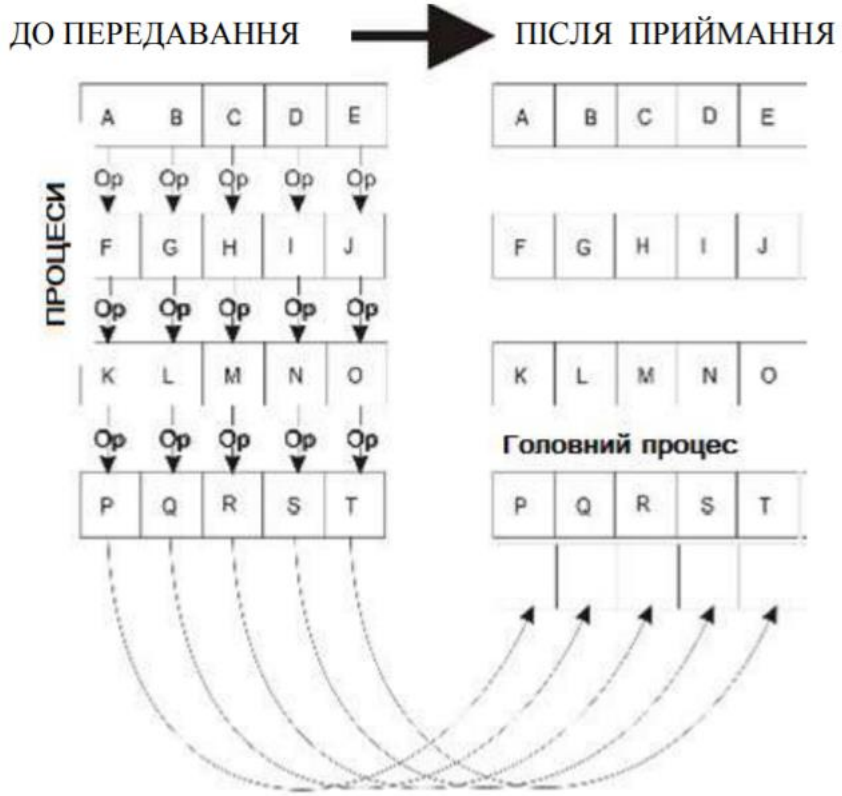

Рис. 2 - Глобальна операція приведення Аргументом операції приведення є массив даних - по одному елементу від кожного процесу. Результат такої операції - єдине значення.

1.4 Створення групових процесів Для організації колективних обмінів на підмножині процесів створюють групу і відповідний їй комунікатор.

Група - упорядкована множина процесів.

Кожному процесу в групі відповідає свій ранг. Операції з групами можуть виконуватися окремо від операцій з комунікаторами, але в операціях обміну використовуються тільки комунікатори. У MPI є спеціальна порожня група MPI_GROUP_EMPTY.

Комунікатори бувають двох типів: інтракомунікатори - для операцій усередині однієї групи процесів і інтеркомунікатори - для двоточкового обміну між двома групами процесів.

У MPI-програмі частіше використовуються інтракомунікатори, які включають примірник групи, контекст обміну для всіх її видів, а також, можливо, віртуальну топологію й інші атрибути.

Створенню нового комунікатора передує створення відповідної групи процесів. Операції створення груп аналогічні математичним операціям над множинами: об'єднання; перетин; різниця.

Нову групу можна створити тільки з уже наявних груп. Базова група, з якої формуються всі інші групи, пов'язана з комунікатором MPI_COMM_WORLD.

Доступ до групи group, пов'язаної з комунікатором comm можна отримати, звернувшись до підпрограми MPI_Comm_group.

Підпрограма MPI_Comm_create створює новий комунікатор з підмножини процесів іншого комунікатора. Виклик цієї підпрограми мають виконати всі процеси зі старого комунікатора, навіть якщо вони не входять до групи group, з однаковими аргументами. Ця операція застосовується тільки до інтракомунікаторів. Вона дозволяє виділяти підмножини процесів зі своїми областями взаємодії, якщо, наприклад, потрібно зменшити «зернистість» паралельної програми.

1. Широкомовна розсилка

int MPI_Bcast(void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)

Параметри цієї процедури одночасно є вхідними і вихідними:

- Buffer - адреса буфера;

Count - кількість елементів даних у повідомленні;

- Datatype - тип даних MPI;

Root - ранг головного процесу, що виконує трансляцію;

Comm - комунікатор.

2. Синхронізація за допомогою «бар 'єра»

int MPI_Barrier(MPI_Comm comm)

Під час синхронізації з бар'єром виконання кожного процесу із цього комунікатора припиняється до тих пір, поки всі процеси не виконають виклик процедури синхронізації MPI_Barrier.

3. Операція приведення, результат якої передається одному процесу int MPI_Reduce(void *buf, void *result, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)

MPI_Reduce застосовує операцію приведення до операндів з buf, а результат кожної операції міститься в буфері результату result. MPI_Reduce має викликатися всіма процесами в комунікаторі comm, а аргументи count, datatype і op у цих викликах повинні збігатися.

3.4Порядок виконання роботи

У середовищі Visual Studio створіть програми керування комунікаторами. Стандартний комунікатор MPI_COMM_WORLD створюється автоматично із запускои паралельної програми на виконання. Стандартні комунікатори:

- MPI_COMM_SELF - комунікатор, який містить процес, що є викликаючим;
- MPI_COMM_NULL - порожній комунікатор.

Отримання доступу до групи group, пов'язаної з комунікатором comm int MPI_Comm_group(MPI_Comm comm, MPI_Group *group) Вихідний параметр - група. Для виконання операцій з групою до неї спочатку необхідно отримати доступ.

Створення нової групи newgroup з n процесів, що входять до групи oldgroup

int MPI_Group_incl(MPI_Group oldgroup, int n, int *ranks, MPI_Group *newgroup)

Ранги процесів містяться в масиві ranks. До нової групи увійдуть процеси з рангами ranks [0], ..., ranks [n - 1], рангу і в новій групі відповідає ранг ranks [i] у старій групі. Із n = творюється порожняя група MPI_GROUP_EMPTY. За

допомогою цієї підпрограми можна не тільки створити нову групу, а й змінити порядок процесів у старій групі.

Знищення групи group

int MPI_Group_free(MPI_Group *group) Визначення кількості процесів (size) у групі (group) int MPI_Group_size(MPI_Group group, int *size) Створення нового комунікатора (newcomm) з підмножини процесів (group) іншого комунікатора (oldcomm)

int MPI_Comm_create(MPI_Comm oldcomm, MPI_Group group, MPI_Comm *newcomm)

Виклик цієї підпрограми має виконати всі процеси зі старого комунікатора, навіть якщо вони не входять до групи group, з однаковими аргументами. Якщо одночасно створюються кілька комунікаторів, вони мають створюватися в одній послідовності всіма процесами.

3.5Завдання для самостійної роботи

1. У вихідному тексті програми мовою С пропущені виклики процедур широкомовного розсилання. Додати ці виклики, відкомпілювати і запустити програму.

#include "mpi.h" #include <stdio.h>

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

char data[2 4]; int myrank, count = 25; MPI_Status status; MPI_Init(&argc, &argv);

MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

if (myrank == 0) {

strcpy(data, "Hi, Parallel Programmer!");

printf("send: %s\n", data); }

else {

printf("received: %s\n", data); }

MPI_Finalize();

return 0; }

2. У програмі мовоюС передбачається, що три числові значення, уведених з клавіатури, пересилаються широкомовним розсиланням всім іншим процесам. Виклики підпрограм широкомовлення пропущені. Додати ці виклики, відкомпілювати і запустити програму. #include "mpi.h" #include <stdio.h>

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

int myrank; int root = 0; int count = 1; float a, b; int n;

MPI_Init(&argc, &argv);

MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

if (myrank == 0) {

printf("Enter a, b, n\n");

scanf("%f %f %i", &a, &b, &n); }

{

printf("%i Process got %f %f %i\n", myrank, a, b, n); }

MPI_Finalize();

return 0; }

3. У програмі мовою С створюється новий комунікатор, а потім сполучення між процесами, що входять до нього, пересилаються широкомовним розсиланням. Виклики підпрограм створення нової групи процесів (на 1 менше, ніж повна кількість запущених на виконання процесів) і нового комунікатора пропущені. Додати ці виклики, відкомпілювати і запустити програму. #include "mpi.h" #include <stdio.h>

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

char message[24];

MPI_Group MPI_GROUP_WORLD;

MPI_Group group;

MPI_Comm fcomm;

int size, q, proc;

int* process_ranks;

int rank, rank in group;

MPI_Status status;

MPI_Init(&argc, &argv);

MPI_Comm_size(MPI_COMM_WORLD, &size);

MPI_Comm_rank(MPI_COMM_WORLD, &rank);

```
printf("New group contains processes:");
q = size - 1;
process_ranks = (int*) malloc(q*sizeof(int)); for (proc = 0; proc < q; proc++)
{
process_ranks[proc] = proc;
printf("%i process_ranks[proc]); }
printf("\n");
if (fcomm != MPI_COMM_NULL) { MPI_Comm_group(group, &fcomm); MPI_Comm_rank(fcomm, &rank_in_group); if (rank_in_group == 0) {
strcpy(message, "Hi, Parallel Programmer!"); MPI_Bcast(&message, 25, MPI_BYTE, 0, fcomm);
printf("0 send: %s\n", message); }
else {
MPI_Bcast(&message, 25, MPI_BYTE, 0, fcomm);
printf("%i received: %s\n", rank_in_group, message); }
MPI_Comm_free(&fcomm);
MPI_Group_free(&group); }
MPI_Finalize();
return 0; }
```

3.1Вимоги до звіту

Звіт повинен містити:

- Титульну сторінку з даними про виконавця і перевіряючого.
- Тему і мету роботи.

-Лістинг програми.

-Результати застосування створеної функції до тестового зображення.

-Висновки за результатами лабораторної роботи.

Звіт повинен бути оформлений відповідно до вимог СОКР/

3.2Контрольні питання

1. Як пересилається повідомлення під час колективного обміну?
2. Чим характеризуються колективні обміни?
3. Як називають процесс, що виконує широкомовне розсилання?
4. Назвіть спеціальну порожню групу MPI?
5. Що робить підпрограма MPI_Comm_create?
6. Яких параметрів набуває функція широкомовного розсилання?
7. За допомогою якої функції відбувається знищення групи?

4 РОЗРАХУНКОВО-ГРАФІЧНА РОБОТА №4

ПОХІДНІ ТИПИ В MPI

Мета. - ознайомитися зі створенням і використанням похідних типів в MPI..

4.1Теоретичні відомості. Опис похідних типів MPI

Похідні типи даних створюються під час виконання програми. Створення типу - двоступеневий процес, який складається з двох кроків:

1) конструювання типу;
2) реєстрація типу.

Після завершення роботи з похідним типом, він анулюється. При цьому всі похідні типи залишаються і можуть використовуватися далі, поки і вони не будуть знищені.

Похідні типи даних створюються з базових типів за допомогою підпрограм-конструкторів. Операції створення похідних типів можуть застосовуватися рекурсивно.

Порядок елементів у похідному типі може відрізнятися від початкового. Один елемент даних може з'являтися в новому типі багаторазово. Елементи можуть і розташовуватися з розривами, і перекриватися між собою. Послідовність пар (тип, зсув) називається картою типу.

Підпрограма MPI_Type_struct є найбільш загальним конструктором типу в MPI - програміст може використовувати повний опис кожного елемента типу. Якщо дані, що пересилаються, містять підмножину елементів масиву, така детальна інформація не потрібна, оскільки у всіх елементів один і той же базовий тип. MPI містить три конструктори, які можна використовувати в такій ситуації: MPI_Type_contiguous,

MPI_Type_vector і

MPI_Type_indexed. Перший з них створює похідний тип, елементи якого є безперервно розташованими елементами масиву. Другої створює тип, елементи якого розташовані на однакових відстанях один від одного, а третій створює тип, що містить довільні елементи.

4.2Порядок виконання роботи

Перевірити роботу підпрограм-конструкторів:

Конструктор векторного типу

int MPI_Type_vector (int count, int blocklen, int stride, MPI_Datatype oldtype, MPI_Datatype * newtype)

Вхідні параметри:

- count -кількість блоків (невід'ємне ціле значення);
- blocklen - довжина кожного блоку (кількість елементів, невід'ємне

ціле);

- stride - кількість елементів, розташованих між початком попереднього і початком наступного блоку («гребінка»);
- oldtype - базовий тип.

Вихідний параметр:

- newtype - ідентифікатор нового типу, який призначається програмістом.

Вихідні дані однотипні.

Конструктор структурного типу

int MPI_Type_struct (int count, int blocklengths [], MPI_Aint indices [], MPI_Datatype oldtypes [], MPI_Datatype * newtype)

Вхідні параметри:

- count - задає кількість елементів у похідному типі, а також довжину масивів oldtypes, indices і blocklengths;
- blocklengths - кількість елементів у кожному блоці (масив);
- indices - зміщення кожного блоку в байтах (масив);
- oldtypes - тип елементів у кожному блоці (масив).

Вихідний параметр:

- newtype - ідентифікатор похідного типу.

MPI_Aint - являє собою скалярний тип, довжина якого має розмір, однаковий з покажчиком.

Конструктор індексованого типу

int MPI_Type_indexed (int count, int blocklens [], int indices [], MPI_Datatype oldtype, MPI_Datatype * newtype) Вхідні параметри:

- count - кількість блоків, одночасно довжина масивів indices і blocklens;
- blocklens - кількість елементів у кожному блоці;
- indices - зміщення кожного блоку, що задається в кількості осередків базового типу (цілочисельний масив);
- oldtype - базовий тип. Вихідний параметр:
- newtype - ідентифікатор похідного типу.

Конструктор типу даних з безперервним розташуванням елементів int MPI_Type_contiguous (int count, MPI_Datatype oldtype, MPI_Datatype * newtype) Вхідні параметри:

- count - лічильник повторень;
- oldtype - базовий тип Вихідний параметр:
- newtype - ідентифікатор нового типу.

Конструктор індексованого типу з блоками постійного розміру int MPI_Type_create_indexed_block (int count, int blocklength, int displacements [], MPI_Datatype oldtype, MPI_Datatype * newtype) Вхідні параметри:

- count - кількість блоків і розмір масивів indices і blocklens;
- blocklength - кількість елементів у кожному блоці;
- displacements - зміщення кожного блоку в одиницях довжини типу oldtype (цілочисельний масив);
- oldtype - базовий тип.

Вихідний параметр:

- newtype - ідентифікатор похідного типу.

Конструктор типу даних, відповідного підмасиву багатовимірного масиву

int MPI_Type_create_subarray (int ndims, int * sizes, int * subsizes, int * starts, int order, MPI_Datatype oldtype, MPI_Datatype * newtype)

Вхідні параметри:

- ndims - розмірність масиву;
- sizes - кількість елементів типу oldtype в кожному вимірюванні повного масиву;
- subsizes - кількість елементів типу oldtype в кожному вимірюванні підмасивів;
- starts - стартові координати підмасивів у кожному вимірі;
- order - прапорець, що задає переупорядкування;
- oldtype - базовий тип.

Вихідний параметр:

- newtype - новий тип.

Реєстрація похідного типу datatype, сконструйованого програмістом,

int MPI_Type_commit (MPI_Datatype * datatype)

Видалення похідного типу datatype

int MPI Type free (MPI_Datatype * datatype)

Базові типи даних не можуть бути видалені. Визначення розміру типу datatype в байтах (обсяг пам'яті, який займає один елемент цього типу)

int MPI_Type_size (MPI_Datatype datatype, int * size)

Вихідний параметр - розмір size.

Визначення кількості елементів даних в одному об'єкті типу datatype (його екстент)

int MPI Type extent (MPI_Datatype datatype, MPI_Aint *
extent)

Вихідний параметр - extent.

Зсуви можуть надаватися відносно базової адреси, значення якої міститься в константі MPI_BOTTOM.

Визначення адреси (address) за заданим положенням (location) int MPI_Address (void * location, MPI_Aint * address) Може використовуватися в програмах мовами C і FORTRAN. У C вона зазвичай повертає ту ж саму адресу, що й оператор &, хоча іноді це не так.

Визначення фактичних параметрів, використаних для створення похідного типу

int MPI_Type_get_contents (MPI_Datatype datatype, int max_integers, int max_addresses, int max_datatypes, int * integers, MPI_Aint * addresses, MPI_Datatype * datatypes) Вхідні параметри:

- datatype - ідентифікатор типу;
- max_integers - кількість елементів у масиві integers;
- max_addresses - кількість елементів у масиві addresses;
- max_datatypes - кількість елементів у масиві datatypes. Вихідні параметри:
- integers - містить цілочисельні аргументи, використані для конструювання зазначеного типу;
- addresses - містить аргументи address, використані для конструювання зазначеного типу;
- datatypes - містить аргументи datatype, використані для конструювання зазначеного типу.

Визначення нижньої межі типу даних datatype

int MPI_Type_lb (MPI_Datatype datatype, MPI_Aint Misplacement)

Вихідний параметр:

- displacement - зміщення (у байтах) нижньої межі відносно джерела.

Визначення верхньої межі типу

int MPI_Type_ub (MPI_Datatype datatype, MPI_Aint Misplacement)

Упаковка даних

int MPI_Pack (void * inbuf, int incount, MPI_Datatype datatype, void * outbuf, int outcount, int * position, MPI_Comm comm)

Для виклику incount елементів зазначеного типу вибираються з вхідного буфера та упаковуються у вихідному буфері, починаючи з положення position. Вхідні параметри:

- inbuf - початкова адреса вхідного буфера;
- incount - кількість вхідних даних;
- datatype -тип кожного вхідного елемента даних;
- outcount - розмір вихідного буфера в байтах;
- position - поточне положення в буфері в байтах;
- comm - комунікатор для упакованого повідомлення. Вихідний параметр:
- outbuf - стартова адреса вихідного буфера.

Розпакування даних

int MPI_Unpack (void * inbuf, int insize, int * position, void * outbuf, int outcount, MPI_Datatype datatype, MPI_Comm comm)

Вхідні параметри:

- inbuf - стартова адреса вхідного буфера;
- insize - розмір вхідного буфера в байтах;
- position - поточне положення в байтах;
- outcount - кількість даних, які мають бути розпаковані;
- datatype - тип кожного вихідного елемента даних;

- comm - комунікатор для упаковуваного повідомлення.

Вихідний параметр:

- outbuf - стартова адреса вихідного буфера.

Визначення обсягу пам 'яті size (у байтах), необхідного для розпакування повідомлення

int MPI_Pack_size (int incount, MPI_Datatype datatype, MPI_Comm comm, int * size) Вхідні параметри:

- incount - аргумент count, використаний для упакування;
- datatype - тип упакованих даних;
- comm - комунікатор.

4.3Завдання для самостійної роботи

У програмі мовою С задаються типи членів похідного типу, потім кількість елементів кожного типу. Після цього обчислюються адреси членів типу indata і визначаються зміщення трьох членів похідного типу відносно адреси першого, для якого зсув дорівнює 0. Потім визначається похідний тип. Аргументи підпрограм MPI_Type_struct і MPI_Type_commit, а також деякі інші фрагменти пропущені. Додати ці фрагменти, відкомпілювати і запустити програму.

#include "mpi.h" #include <stdio.h>

struct newtype {

Float a; float b; int n;

};

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

int myrank;

MPI_Datatype NEW_MESSAGE_TYPE; int block_lengths[3] ; MPI_Aint displacements[3];

MPI_Aint addresses[4]; MPI_Datatype typelist[3]; int blocks_number; struct newtype indata; int tag = 0; MPI_Status status;

MPI_Comm_rank(MPI_COMM_WORLD, &myrank); typelist[0] = MPI_FLOAT; typelist[1] = MPI_FLOAT; typelist[2] = MPI_INT;

block_lengths[0] = block_lengths[1] = block_lengths[2] = 1; MPI_Address(&indata, &addresses[0]); MPI_Address(&(indata.a), &addresses[1]); MPI_Address(&(indata.b), &addresses[2]); MPI_Address(&(indata.n), &addresses[3]); displacements[0] = addresses[1] - addresses[0]; displacements[1] = addresses[2] - addresses[0]; displacements[2] = addresses[3] - addresses[0]; blocks_number = 3; MPI_Type_struct(...); MPI_Type_commit(...);

if (myrank == 0)

{

indata.a = 3.14159; indata.b = 2.71828; indata.n = 2002;

```
MPI_Send(&indata, 1,NEW_MESSAGE_TYPE, 1, tag, MPI_COMM_WORLD);
printf("Process %i send: %f %f %i\n", myrank, indata.a, indata.b, indata.n);
}
else {
MPI_Recv(&indata, 1, NEW_MESSAGE_TYPE, 0,
tag, MPI_COMM_WORLD, &status); printf("Process %i received: %f %f %i, status %s\n", myrank, indata.a, indata.b, indata.n,
status.MPI_ERROR); }
MPI_Finalize(); return 0;
}
```

4.1Вимоги до звіту по РГР

Звіт повинен містити:

- Титульну сторінку з даними про виконавця і перевіряючого.
- Тему і мету роботи.
-Лістинг програми.
-Результати застосування створеної функції до тестового зображення.
-Висновки за результатами лабораторної роботи.

Звіт повинен бути оформлений відповідно до вимог СОКР/

4.2Контрольні питання

1. Як створюються похідні типи в MPI?
2. Як характеризуються похідні типи MPI?
3. Що таке карта типу?
4. Які конструктори використовуються, якщо дані, що пересилаються, містять підмножину елементів масиву?
5. Який тип створює конструктор MPI_Type_contiguous?
6. У якій константі містяться значення базової адреси?
7. Яка функція використовується для упакування даних?

РЕКОМЕНДОВАНА ЛИТЕРАТУРА

1. Андрюс Г. Р. Основи многопоточного, параллельного и распределенного программирования / Г. Р. Зндрюс. - М.: Вильямс, 2003. - 512 с.
2. Дорошенко А. Е. Математические модели и методи организации вьісокопроизводительньїх параллельних вичислений А. Е Дорошенко. -К.: Наукова думка, 2000. - 177 с.
3. Корнеев В. В. Параллельньїе вьічислительньїе системи / В. В. Корнеев. - М.: Нолидж, 1999. - 320 с.
4. Гергель В.П. Основи параллельних вичислений для многопроцессорних вичислительних систем / В.П. Гергель, Р.Г. Стронгин. Учебное пособие. - Нижний Новгород: Изд-во ННГУ им. Н.И. Лобачевского, 2003. -184 с.
5. Crichlow J. M. An Introduction to Distributed and Parallel Computing / J. M. Crichlow. - Prentice Hall, 1997. - 209 p.
6. Злементи параллельного программирования / В. А. Вольковский, В. Е. Котов, А. Г. Марчук, Н. Н. Миренков. - М.: Радио й связь, 1983. - 240 с.
7. Бурова И.Г. Алгоритми параллельних вичислений и программирование / И.Г. Бурова, Ю.К. Демьянович. Курс лекций. - Изд-во С. -Пб. ун-та, 2007. - 206 с.
8. Воеводин В.В. Параллельньїе вичисления / В.В. Воеводин, Вл.В. Воеводин. - СПб.: БХВ-Петербург, 2002. - 608 с.
9. Программирование многопроцессорних вичислительних систем. Ростов-на-Дону // А. А. Букатов, В. Н. Дацюк, А. И. Жегуло. Издательство ООО «ЦВВР», 2003. - 208 с.
10. Synchronization of Parallel Programs / Andre J., Herman D., Verjus J.-P. Oxford: North Oxford Academic Publishing Company Limited, 1985. - 110 p.
11. Parallel Computing. Architectures, Algorithms and Applications / Bischof C., Bucker M., Gibbon P., Joubert G.R., Lippert T., Mohr B., Peters F.
12. (eds.) OS Press, 2008. - 825 p.
13. Pllana Sabri. Programming multicore and many-core computing ystems/ Sabri Pllana, Fatos Xhafa.Wiley, 2017. - 528 p.
14. Рихтер Дж. Создание ^ффективних WIN32-приложений с учетом специфики 64-разрядной версии Windows / Дж. Рихтер. Пер. с англ.: 4-е изд. — СПб.: Питер; М.: Русская Редакция, 2001. - 752 с.
15. Качко Е.Г. Параллельное программирование / Е.Г. Качко. Харьков: Форт, 2011. - 528 с.

16. Корнеев В.Д. Параллельное программирование в MPI / В.Д. Корнеев. 2-е изд., испр. - Новосибирск: Изд-во ИВМиМГ СО РАН, 2002. - 215 с.
17. Лазарович І.М. Паралельні обчислювальні середовища. Лабораторний практикум/ І. М. Лазарович. - Івано-Франківськ: Видавництво Прикарпатського національного університету імені Василя Стефаника, 2014. - 65 с.
18. Антонов А.С. Параллельное программирование с использованием технологии OpenMP / А.С. Антонов. Учебное пособие. - М.: Изд-во МГУ, 2009. - 77 с.
19. Таненбаум ^. Распределенние системи. Принципи и парадигми / З. Таненбаум, М. ван Стеен. - СПб.: Питер, 2003. - 877 с.
20. Радченко Г.И. Распределенние вичислительние системи / Г.И. Радченко. - Челябинск: Фотохудожник, 2012. - 184 с.
21. Foster I. The Grid. Blueprint for a new computing infrastructure / I. Foster, C. Kesselman. San Francisco: Morgan Kaufman, 1999. - 677 p.
22. Інформаційні ресурси мережі Інтернет
23. Інформаційно-аналітичні матеріали з паралельних обчислень (http://www.parallel.ru).
24. T0P500. Рейтинг 500 найпотужніших відомих суперкомп'ютерних систем (http://www.top500.org/).
25. Обчислювальний кластер Київського національного університету ім. Т. Г. Шевченка (http://cluster.univ.kiev.ua/).
26. Основна
27. Аксак Н. Г. Паралельні та розподілені обчислення: пірдуч. / Н. Г. Аксак, О. Г. Руденко, А. М. Гуржій. - Х. :Компанія СМІТ, 2009.
28. Антонов А. С. Введение в параллельньїе вьічисления : учебное пособие / А. С. Антонов. - М . : Изд-во МГУ, 2002. - 69 с.
29. Богачев К. Ю. Основи параллельного программирования / К. Ю. Богачев. - М . : БИНОМ. Лаборатория знаний, 2003. - 342 с.
30. Гергель В. П. Лекции по параллельньїм вьічислениям : учебное пособие / В. П. Гергель, В. А. Фурсов. - Самара . : изд-во Самар. гос. азрокосм. ун-та, 2009. - 164 с.
31. Воеводин В. В. Параллельньїе вичисления / В. В. Воєводин, Вл. В. Воеводин - СПб. : БХВ-Петербург, 2002. - 600 с.
32. Жуков І. Паралельні та розподілені обчислення / І. Жуков, О. Корочкін.К. Корнійчук, 2005. - 226 с.
33. Кузьменко Б. В. Технологія розподілених систем та паралельних обчислень : навчальний посібник. Частина 1 / Б. В. Кузьменко, О. А. Чайковська. - Київ : Видавничий центр КНУКІМ, 2011. - 161 с.
34. Немнюгин С. А. Средства программирования для многопроцессорньїх вьічислительньїх систем : учебное пособие / С. . Немнюгин. - Санкт- Петербург : изд-во Санкт-Петербургского государственного университета, 2007. - 88 с.

35. Зндрюс Г. Основьі многопоточного, параллельного и распределенного программирования / Г. Зндрюс ; пер. с англ. - М. : Изд. Дом «Вильямс», 2003.
36. Ясько М. М. Навчальний посібник до вивчення курсів «Паралельна обробка даних» та «Мови обчислень та кластерні системи» / М. М. Ясько. - ВВ ДНУ, 2010. - 76 с.
37. Додаткова
38. Багатоядерні процесори: мікроархітектура та особливості застосування : посібник / М. М. Барченко, І. Б. Березовська. - Львів : Ліга Прес, 2009. - 176 с.
39. Бройнль Т. Паралельне програмування. Початковий курс : навч. посіб. - К. : Вища шк., 1997. - 358 с.
40. Богачев К. Ю. Основи параллельного программирования. - М. : БИНОМ. Лаборатория знаний, 2003. - 342 с.
41. Дерев'янченко О. В. ПАРКС-JAVA система для паралельних обчислень на комп'ютерних мережах : навчальний посібник для студентів факультету кібернетики / О. В. Дерев'янченко. - Київ. - 2011. - 60 с.
42. Дорошенко А. Ю. Алгеброалгоритмічні основи програмування / А. Ю. Дорошенко, Г. С. Фінін, Г. О. Цейтлін. - К. : Наук. думка. - 2004. - 457 с.
43. Жуков І. А. Паралельні та розподілені обчислення / І. А. Жуков, О. В. Корочків. - К. : Корнійчук, 2005. - 224 с.
44. Кавун С. В. Архітектура комп'ютерів. Особливості використання комп'ютерів в ІС : навчальний посібник / С. В. Кавун, І. В. Сорбат. - Харків : Вид. ХНЕУ, 2010. - 256 с.
45. Мельник А. О. Архітектура комп'ютера / А. О. Мельник. -Луцьк : 2008. - 470 с.
46. Рихтер Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.0 на язьіке C# / Дж. Рихтер, - [3-е изд.] - СПб. : Питер, 2012. - 928 с.
47. Quinn M. J. Parallel Programming in C with MPI and OpenMP. McGrawHill / M. J. Quinn. - 2004. - 544 p.
48. Інформаційні ресурси
49. http: //uk.wikipedia. org/wiki/Кластер_(Інформатика)
50. www. ci. ru/inform 10_99/p_08_9. htm «Кластери на ОС Linux как системи високой доступности».
51. www.cluster.linux-ekb.info/build.php «Варианти построения кластера».
52. www.frmb.org/occtutor.html «Учебник по Occam».
53. MPI https://www.youtube.com/watch?v=xeTOUQb96Bg&list=PLY0WmHiV80hvBowBcNtj0wI6cRsiOFx4_&index=1&t=209s
54. Visual Studio https://www.youtube.com/watch?v=4ozCfih75Uk&t=2s