**МІНІСТЕРСТВО ОСВІТИ І НАУКИ,**
**МОЛОДІ ТА СПОРТУ УКРАЇНИ**
**ПРИКАРПАТСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**
**ІМЕНІ ВАСИЛЯ СТЕФАНИКА**

**І. М. Лазарович**

# ПАРАЛЕЛЬНІ ОБЧИСЛЮВАЛЬНІ СЕРЕДОВИЩА

## ЛАБОРАТОРНИЙ ПРАКТИКУМ

Івано-Франківськ
2014

**УДК 004.4’2**
**ББК 32.973.2-018**
Л17

*Рекомендовано до друку вченою радою факультету математики та інформатики Прикарпатського національного університету імені Василя Стефаника, ( протокол № 3 від 30 жовтня 2014 р.)*

***Рецензенти:***

***Козленко М.І.*** – кандидат технічних наук, доцент кафедри кафедри інформаційних технологій Прикарпатського національного університету імені Василя Стефаника;

***Соломко А.В.*** – кандидат фізико-математичних наук, доцент кафедри математичного і функціонального аналізу факультету математики та інформатики Прикарпатського національного університету імені Василя Стефаника

**Лазарович І. М.**

Л17 Паралельні обчислювальні середовища. Лабораторний практикум/ І. М. Лазарович. – Івано-Франківськ : Видавництво Прикарпатського національного університету імені Василя Стефаника, 2014. – 65 с.

Лабораторний практикум складається із шести лабораторних робіт, які містять теоретичні відомості до виконання роботи, варіанти завдань для виконання, вимоги до оформлення звітів та контрольні питання для самоперевірки. Теоретичні відомості містять відповідні зображення екранних форм середовища розробки, що полегшує виконання роботи і засвоєння матеріалу. Тематика робіт відповідає робочій програмі з дисципліни «Паралельні обчислюючі середовища» для студентів 5-го курсу спеціальності «інформатика». Практикум орієнтований на студентів, що мають мінімальні навики роботи із сучасними засобами розробки програмного забезпечення, зокрема базові поняття мови С++.

**УДК 004.4’2 ББК 32.973.2-018**

**ЗМІСТ**

# ПЕРЕДМОВА

Сучасні задачі моделювання складних систем у автомобілебудуванні, нафто- газовидобуванні, медицині, генній інженерії, у виробничих, хімічних та фізичних процесах часто є настільки ресурсоємкими, що навіть найсучасніші персональні ЕОМ не здатні вирішити їх в прийнятні терміни. Витрати, пов’язані з обчисленнями, істотно збільшуються із збільшенням їх точності. Дослідження характеризуються великими вимогами до об’єму пам’яті, підвищеними вимогами до організації процесу введення-введення.

Інша тенденція інформаційних технологій – швидке зростання продуктивності мереж, зокрема мережі Internet, зниження собівартості передавання інформації та одночасне зростання надійності передавання даних. Це обумовлює появу програмних додатків, які використовують розподілені обчислювальні ресурси.

Тому протягом останніх років в галузі інформаційних технологій дедалі частіше застосовуються методи паралельних та розподілених обчислень та GRID-технології. Розвиток цього напряму пояснюється як активним впровадженням провідними світовими виробниками обчислювальної техніки на основі багатоядерних процесорів чи багатопроцесорних систем, так і перспективою централізації обчислень.

Метою лабораторного практикуму з дисципліни «Паралельні обчислюючі середовища» є вивчення студентами спеціальності «Інформатика» сучасних інформаційних технологій використання колективних GRID-обчислень та методів паралельного і розподіленого програмування. Практикум складається із шести робіт, до кожної з них наведені теоретичні відомості для підготовки та виконання завдань. індивідуальні завдання та контрольні питання для самоперевірки.

# Лабораторна робота №1

**Тема робота:** GRID-технології

**Мета роботи:** ознайомитися з принципами організації розподілених обчислень і актуальними обчислювально складними проектами

## Теоретичні відомості

Термін Ґрід (англ. Grid) означає узгоджене, відкрите і стандартизоване середовище, яке забезпечує гнучкий, безпечний, скоординований розподіл ресурсів у рамках віртуальної організації. В рамках обчислювальних технологій Grid — це тип паралельно розподіленої системи, яка дозволяє розділення, вибір та накопичення географічно розподілених «автономних» ресурсів в реальному часі в залежності від їх придатності, можливості, роботи, ціни та вимогам якості-обслуговування користувачів.

GRID - система координує розрізнені ресурси. Ресурси не мають загального центру управління, а GRID-система займається координацією їх використання, наприклад, балансування навантаження. Тому проста система управління ресурсами кластера не є системою GRID, тому що здійснюється централізоване управління усіма вузлами даного кластера, маючи до них повний доступ. GRID-системи мають лише обмежений доступ до ресурсів, які залежать від політики того адміністративного домену, в якій цей ресурс знаходиться.

GRID-система будується на базі стандартних та відкритих протоколів, сервісів та інтерфейсів. Не маючи стандартних протоколів, неможливо легко та швидко підключати нові ресурси до GRID -системи, розробляти нові види сервісів.

Додамо ще декілька властивостей, якими взагалі володіє GRID-система:

- гнучкість: можливість забезпечення розподіленого доступу до будь-яких ресурсів;

– масштабування: працездатність GRID-системи при значному збільшенні чи зменшенні її складу;
– гнучка й потужна підсистема безпеки: стійкість до атак зловмисників, забезпечення конфіденційності;
– можливість контролю над ресурсами: використання локальних та глобальних політик і квот;
– гарантії якості обслуговування;
– можливість одночасної, скоординованої роботи з декількома ресурсами.

Хоча сама технологія GRID не прив’язана до визначених ресурсів, найбільш часто реалізації GRID-систем забезпечують роботу з наступними типами ресурсів:

– обчислювальні ресурси – окремі комп’ютери та кластери;
– сховища даних – диски та дискові масиви, стрічки, системи масового сховища даних;
– мережні ресурси;
– програмне забезпечення – будь-яке спеціалізоване ПО.

Технологія GRID включає в себе лише найбільш загальні та універсальні аспекти, однакові для будь-якої системи (архітектура, протоколи, інтерфейси, сервіси). Використовуючи цю технологію та заповнюючи її конкретним змістом, можливо реалізувати ту чи іншу GRID-систему, направлену задля рішення того чи іншого класу прикладних задач.

У рамках конкретної GRID-системи, звичайно, можливо організувати паралельні обчислення з використанням існуючих технологій (PVM, MPI), так як GRID-систему можливо розглядати як деякий метакомп’ютер, маючий велику кількість обчислювальних вузлів. Однак технологія GRID не є технологією паралельних обчислень, в її задачі входить лише координація використання ресурсів.

Прикладом платформи для організації GRID є некомерційне відкрите програмне забезпечення BOINC (англ. Berkeley Open Infrastructure for Network Computing), яке використовується для організації обчислень на основі добровільно наданих користувачами проекту обчислювальних ресурсів власних комп'ютерів. Спочатку BOINC розроблявся для найбільшого проекту добровільних обчислень — SETI@home, але згодом розробники з Каліфорнійського університету в Берклі зробили платформу доступною для сторонніх проектів. На сьогоднішній день BOINC є універсальною платформою для проектів у галузі математики, молекулярної біології, медицини, астрофізики та кліматології. BOINC дає дослідникам можливість задіяти величезні обчислювальні потужності персональних комп'ютерів з усього світу.

Складовими BOINC є серверне і клієнтське ПЗ. Серверна частина в основному являє собою набір PHP-скриптів і необхідна організаторам проектів для спільного управління проектом: реєстрація учасників, розподіл завдань для обробки, отримання результатів, управління базами даних проекту. Для користувачів поняття BOINC частіше використовується в контексті поняття BOINC-клієнт — універсальний клієнт для роботи з різними (BOINC-сумісними) проектами розподілених обчислень. BOINC-клієнт дозволяє брати участь одночасно в декількох проектах з допомогою однієї загальної програми управління (boinc або boinc.exe).

Для візуалізації процесу управління BOINC-клієнтом можна використовувати або офіційну програму-менеджер, що поставляється за замовчуванням (boincmgr або boincmgr.exe), або скористатися «неофіційною» програмою для моніторингу та управління BOINC-клієнтом. Слід зазначити, що власне BOINC-клієнт в академічному розумінні не має користувацького інтерфейсу як такого, а являє собою сервіс, що запускається разом із системою і управляється по протоколу TCP/IP.

Слід відмітити розвиток GRID-технологій і в Україні. В 2004 р., коли в нашій країні було встановлено перші грід-сервери, вітчизняна академічна грід-

інфраструктура значно розширилася, перетворившись в Український національний грід. У травні 2011 р. він став повноправним учасником колаборації NorduGrid, завдяки чому українські фахівці можуть впливати на розвиток європейських грід-технологій і долучатися до спільних міжнародних проектів у цій сфері.

## Завдання для виконання

1. Встановити клієнт BOINC для відповідної операційної системи, використовуючи посилання http://boincstats.com/page/download.php.

2. Вибрати проект згідно варіанту (таблиця 1). Ознайомитися із змістом і статусом проекту використовуючи посилання http://boincstats.com

3. Додати вибраний проект в клієнт (Головне меню → Сервіс → Додати проект). Дочекатися завершення завантаження даних проекту і початку розрахунку (для деяких проектів, наприклад World Community Grid, потрібна попередня реєстрація на сайті). Далі обчислення можуть бути продовжені у фоновому режимі, не заважаючи роботі користувача.

4. По можливості продовжити роботу проекту на домашньому комп'ютері. При підключенні до проекту вказати вже існуючі облікові дані(e-mail і пароль).

5. Якщо проект підтримує можливість виконання розрахунків з використанням відеокарти і вашому комп'ютері встановлена відеокарта, що відповідає вимогам проекту(за об'ємом відеопам'яті, підтримці технології nVidia CUDA, ATI STREAM або OpenCL), рекомендується дозволити її використання. При правильному використанні обчислювальних ресурсів відеокарти можливе отримання значно більшої кількості балів(у 10 і більше разів) в порівнянні з процесором за той же період часу

Таблиця 1. Варіанти завдань для GRID-обчислень

| № варіанту | Назва проекту | № варіанту | Назва проекту |
|---|---|---|---|
| 1 | LHC@Home | 16 | MilkyWay@home, |

| | | | |
|---|---|---|---|
| 2 | Einstein@Home | 17 | GPUGRID, |
| 3 | Rosetta@home | 18 | SETI@home, |
| 4 | Cosmology@Home | 19 | Collatz conjecture |
| 5 | WCG | 20 | **FreeHAL@home** |
| 6 | Climate Prediction | 21 | **BURP** |
| 7 | QMC@Home | 22 | **Hydrogen@Home** |
| 8 | Mersenne@home | 23 | **Enigma@Home** |
| 9 | RNA World | 24 | **Malaria Control** |
| 10 | PrimeGrid | 25 | **DNA@Home** |
| 11 | EDGeS@Home | 26 | **QMC@Home** |
| 12 | POEM@Home | 27 | **NumberFields** |
| 13 | Docking@Home | 28 | **theSkyNet POGS** |
| 14 | ABC@home | 29 | **LHC Test4Theory** |
| 15 | IBERCIVIS | 30 | **LHC@Home** |

6. За участь в проекті нараховуються відповідні бали. За нарахуванням балів рекомендується періодично стежити, оскільки можливі перебої в роботі серверів проектів, оновлення версії розрахункових модулів. Рекомендований час участі в проекті від 4 годин і більше.

7. При з’ясуванні об’єму витікаючого трафіку, щоб не пропустити момент відпрравки результуючих даних завершеного завдання на сервер, в головному меню BOINC Manager ’а можна встановити перемикач, що забороняє мережеву активність(Головне меню → Управління → Network Activity Suspended), потім дочекатися завершення розрахунку завдання, подивитися розмір файлів, очікуючих відправки(закладка Передача), після чого дозволити мережеву активність(Головне меню → Управління → Підключений до інтернету), виділити файли для відправки і натиснути кнопку Повторити. У цей момент почнеться відправка наявних файлів і, можливо, отримання нових завдань. Для вимірювання об’єму вхідного трафіку можна зняти screenshot вікна BOINC Manager ’а, натиснувши Print Screen, вставити його будь-який графічний

редактор (Photoshop, Paint) і потім проаналізувати (інакше дані можуть бути завантажені занадто швидко)

8. Оформити звіт із вказанням наступної інформації:

8.1. Короткий опис проекту;

8.2. Навести конфігурацію комп'ютера і характеристики проекту відповідно до таблиць 2 і 3.

Таблиця 2. Апаратна конфігурація комп'ютера

| Параметр | Значення |
|---|---|
| Процесор | |
| Модель | Intel Core 2 Duo 6300 |
| CPUID | 006F6h |
| Тактова частота | 1,86 ГГц |
| Число ядер | 2 |
| Організація і об'єм кеш-пам'яті | 64+64 КБ L1, 2 МБ L2 |
| Підтримка технологій | співпроцесор(FPU), векторні розширення(MMX, SSE, SSE2, SSE3, SSSE3) |
| Пам'ять | |
| Тип | DDR3 |
| Об'єм | 4 ГБ(2+2) |
| Відеокарта | |
| Модель | nVidia GeForce 450 GTS |
| Можливості для розрахунку | CUDA, OpenCL, PhysX |
| Тип відеопам'яті | GDDR5 |
| Об'єм відеопам'яті | 1 ГБ |
| Ширина шини | 128 біт |
| Число скалярних процесорів | 192 |
| Частота ядра | 783 Мгц |
| Частота пам'яті | 902 Мгц |
| Частота шейдерного блоку | 1566 Мгц |

Таблиця 3. Технічні характеристики проекту

| | |
|---|---|
| Технічні характеристики проекту | |
| Проект | QMC@Home |
| Середній час розрахунку | 06:35:00 |
| Можливість використання мно-гоядерности | ні |
| Використовуваний об'єм оперативної пам'яті | 3 МБ |
| Використовуваний об'єм диска | 9,38 МБ |
| Об'єм трафіку (разо-, що входить вый при підключенні проекту) | 8 МБ |
| Об'єм трафіку расчет-, що входить ного завдання | 100 КБ |
| Об'єм витікаючого трафіку рас-парного завдання | 2,5 КБ |
| Наявність графічного інтерфейсу | ні |
| Deadline | 5днів |
| Можливість використання ви-деокарты | ні |
| Середній об'єм обчислень в день (для проекту) | 8,1 TFLOPs |
| Особистий вклад в обчислення у рамках проекту | 1 723 810 CS, 1 488 PFLOPs, 8 609 ГГц-днів, 23,6 ГГц-років |
| Місце серед усіх учасників(для усіх проектів) | 3275 з 2 469 079 |
| Середня кількість очок за годину роботи | 38,1 CS/година |
| Особистий вклад (платформа в це-лом) | 51 688 268 CS, 45 324 PFLOPs, 262 294 ГГц-днів, 718 ГГц-років |

8.3. Навести технічні screenshot-и характеристик виконаного проекту:

- мінімальний і максимальний час обчислення завдання на вашому комп'ютері (BOINC Manager → Завдання → Інформація);

- можливість використання багатоядерності (по завантаженню процесора, з використанням утиліт Process Explorer(переважно) або Диспетчер завдань Windows);

- об'єм вхідного трафіку при завантаженні завдання, об'єм вихідного трафіку при відправці результатів обчислень (BOINC Manager → закладка Передача);

- особистий вклад в загальний об’єм обчислень, виражений в числі операцій з плаваючою крапкою, за даними особистого сертифікату (з сайту проекту).

9. Зробити висновки про виконану роботу.

10. Оформити звіт з роботи, який включає тему, мету, завдання, хід роботи відповідно до пунктів 1-9 з використанням screenshot-ів і коротких пояснень, висновки.

**Контрольні питання**

1. Що таке GRID, які є його види?
2. Для чого використовується платформа BOINC?
3. Що потрібно для участі в проекті BOINC?
4. Які засоби обчислень може використовувати BOINC?
5. Які переваги і недоліки GRID-технологій?
6. Які є альтернативи GRID-системам?
7. Які властивості GRID-систем?
8. Який механізм взаємодії вузлів у GRID-системах?

# Лабораторна робота №2

**Тема робота:** Організація кластерних обчислень з використанням технології MPI

**Мета роботи:** ознайомитися з принципами функціонування технології MPI та налаштування вузлів кластера в середовищі Windows для виконання виконання паралельних програм.

## Теоретичні відомості

Клаастер — це декілька незалежних обчислювальних машин, що використовуються спільно і працюють як одна система для вирішення тих чи інших задач, наприклад, для підвищення продуктивності, забезпечення надійності, спрощення адміністрування тощо. Обчислювальний кластер потрібен для збільшення швидкості обрахунків за допомогою паралельних обчислень.

Однією із найпоширеніших сучасних технологій організації паралельних обчислень є MPI. MPI (Message Passing Interface) — інтерфейс обміну повідомленнями (інформацією) між одночасно працюючими обчислювальними процесами. Він широко використовується для створення паралельних програм для обчислювальних систем з розподіленою пам’яттю (кластерів).

MPICH — найвідоміша реалізація MPI, створена в Арагонській національній лабораторії (США). Існують версії цієї бібліотеки для усіх популярних операційних систем. До того ж, вона безкоштовна. Перераховані чинники роблять MPICH ідеальним варіантом для того, щоб почати практичне освоєння MPI.

В роботі буде використовуватись MPICH2. Двійка в назві — це не версія програмного забезпечення, а номер того стандарту MPI, який реалізований у

бібліотеці. MPICH2 відповідає стандарту MPI 2.0, звідси і назва. Мета створення MPICH2 наступна:

- надати реалізацію MPI, яка ефективно підтримує різні обчислювальні і комунікаційні платформи, включаючи загальнодоступні кластери (настільні системи, системи із загальною пам’яттю, багатоядерна архітектура), високошвидкісні мережі (Ethernet 10 ГБіт/з, InfiniBand, Myrinet, Quadrics) і ексклюзивні обчислювальні системи (Blue Gene, Cray, SiCortex).
- зробити можливими передові дослідження технології MPI за допомогою легко розширюваної модульної структури для створення похідних реалізацій.

В якості кластера можна використовувати комп’ютерну мережу кавчального класу. За відсутності комп’ютерної мережі можна запускати усі обчислювальні процеси і на одному комп’ютері. Якщо комп’ютер одноядерний, то, природно, приросту швидкодії ви не отримаєте, — тільки уповільнення. Якщо багатоядерний – то можна завантажити всі обчислювальні ядра.

В якості середовища розробки доцільно використовувати Visual Studio 2008 або 2010 (можливе використання безкоштовної версії Express). Для зручності викладу створену програму, що використовує MPI, і призначену для запуску на декількох обчислювальних вузлах, називатимемо MPI -програмою.

MPICH для Windows складається з наступних компонентів:

- Менеджер процесів smpd.exe, який є системною службою (сервісне застосування). Менеджер процесів веде список обчислювальних вузлів системи, і запускає на цих вузлах MPI-програми, надаючи їм необхідну інформацію для роботи і обміну повідомленнями.
- заголовні файли (.h) і бібліотеки компіляції (.lib), необхідні для розробки MPI -програм.
- бібліотеки виконання (.dll), необхідні для роботи MPI -програм.
- Додаткові утиліти (.exe), необхідні для налаштування MPICH і запуску MPI -програм.

Усі компоненти, окрім бібліотек виконання, встановлюються по замовчуванню в теку C :\Program Files\MPICH2; dll-бібліотеки встановлюються в C:\Windows\System32.

Менеджер процесів є основним компонентом, який має бути встановлений і налагоджений на усіх комп'ютерах мережі (бібліотеки часу виконання можна, в крайньому випадку, копіювати разом з MPI -программой). Інші файли потрібно для розробки MPI -программ і налаштування деякого «головного» комп'ютера, з якого здійснюватиметься їх запуск.

Менеджер працює у фоновому режимі і чекає запитів до нього з мережі з боку «головного» менеджера процесів (за умовчанням використовується мережевий порт 8676). Щоб якось узабезпечити себе від хакерів і вірусів, менеджер вимагає пароль при зверненні до нього. Коли один менеджер процесів звертається до іншого менеджера процесів, він передає йому свій пароль. Звідси витікає, що треба вказувати один і той же пароль при установці MPICH на комп'ютери мережі.

Запуск MPI -програми робиться таким чином (рисунок 1) :

1. Користувач за допомогою програми Mpirun (чи Mpiexec, при використанні MPICH2 під Windows) вказує ім'я виконуваного файлу MPI -програми і необхідне число процесів. Крім того, можна вказати ім'я користувача і пароль : процеси MPI -програми запускатимуться від імені цього користувача.

2. Mpirun передає відомості про запуск локальному менеджерові процесів, у якого є список доступних обчислювальних вузлів.

3. Менеджер процесів звертається до обчислювальних вузлів за списком, передаючи запущеним на них менеджерам процесів вказівки по запуску MPI -програми.

4. Менеджери процесів запускають на обчислювальних вузлах декілька копій MPI-програми (можливо, по декілька копій на кожному вузлі), передаючи програмам необхідну інформацію для зв'язку один з одним.

Дуже важливим моментом є те, що перед запуском MPI -програма не копіюється автоматично на обчислювальні вузли кластера. Замість цього менеджер процесів передає вузлам шлях до виконуваного файлу програми точно в тому вигляді, в якому користувач вказав цей шлях програмі Mpirun. Це означає, що якщо ви, наприклад, запускаєте програму C :.exe, то усі менеджери процесів на обчислювальних вузлах намагатимуться запустити файл C :.exe. Якщо хоч би на одному з вузлів такого файлу не виявиться, станеться помилка запуску MPI -програми.

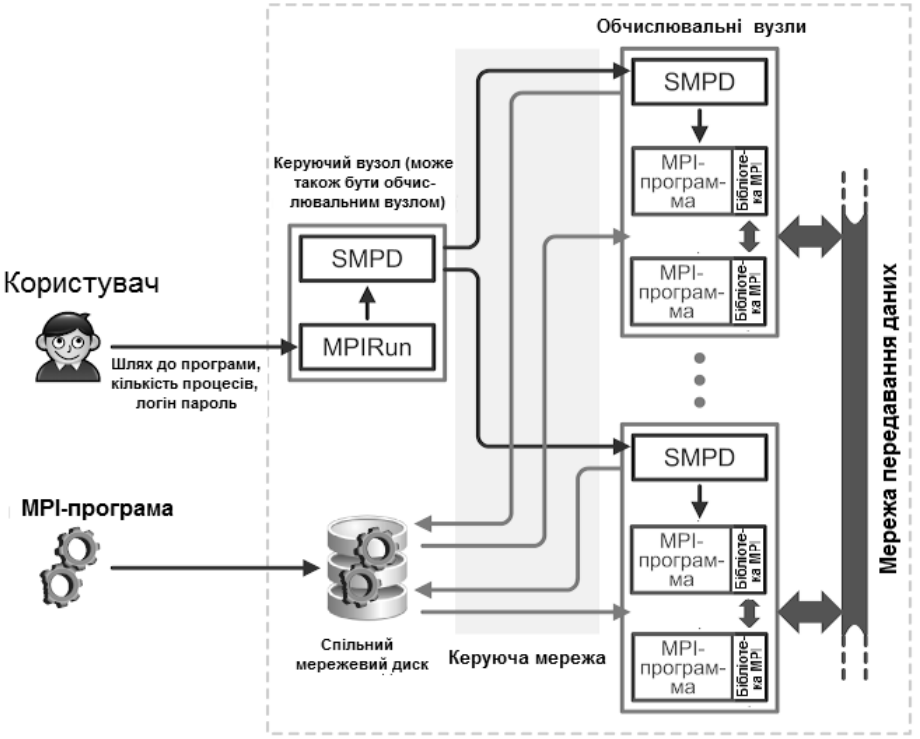



Рис. 1  Схема роботи MPICH на кластері.

Щоб кожного разу не копіювати вручну програму і усі необхідні для її роботи файли на обчислювальні вузли кластера, зазвичай використовують загальний мережевий ресурс. В цьому випадку користувач копіює програму і додаткові файли на мережевий ресурс, видимий усіма вузлами кластера, і вказує шлях до файлу програми на цьому ресурсі. Додатковою зручністю такого підходу є те, що за наявності можливості запису на загальний мережевий

ресурс запущені копії програми можуть записувати туди результати своєї роботи.

Робота MPI -програми відбувається таким чином:

1. Програма запускається і ініціалізує бібліотеку часу виконання MPICH шляхом виклику функції MPI_Init.
2. Бібліотека отримує від менеджера процесів інформацію про кількість і місце розташування інших процесів програми, і встановлює з ними зв’язок.
3. Після цього запущені копії програми можуть обмінюватися один з одним інформацією за допомогою бібліотеки MPICH. З точки зору операційної системи бібліотека є частиною програми (працює в тому ж процесі), тому можна вважати, що запущені копії MPI -програми обмінюються даними безпосередньо один з одним, як будь-які інші застосування, передавальні дані по мережі.
4. Консольне уведення-виведення усіх процесів MPI -програми перенаправляється на консоль, на якій запущена Mpirun. Наскільки я розумію, перенаправленням введення-виводу займаються менеджери процесів, оскільки саме вони запустили копії MPI -програми, і тому можуть отримати доступ до потоків введення-виведення програм.
5. Перед завершенням усі процеси викликають функцію MPI_Finalize, яка коректно завершує передачу і прийом усіх повідомлень, і відключає MPICH.

Усі описані вище принципи діють, навіть якщо ви запускаєте MPI -програму на одному комп’ютері.

## Завдання для виконання

Перший етап роботи полягає у встановленні MPICH на вузлах кластера. Для цього треба виконати наступні пункти:

1. Виконати інсталятор mpich2-1.0.7-win32-ia32.msi , використовуючи права привілеями адміністратора. Для цього потрібно зайти в меню Пуск → Програми → Стандартні програму «Командний рядок», натисніть на неї правою кнопкою миші, і виберіть пункт «Запуск від імені адміністратора» (Рисунок 2). Підтвердіть свої наміри і введіть пароль, якщо необхідно.

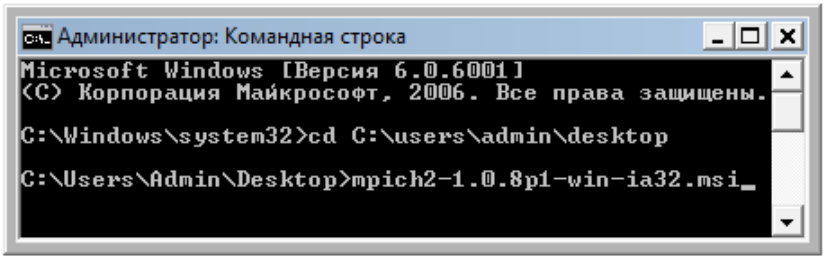
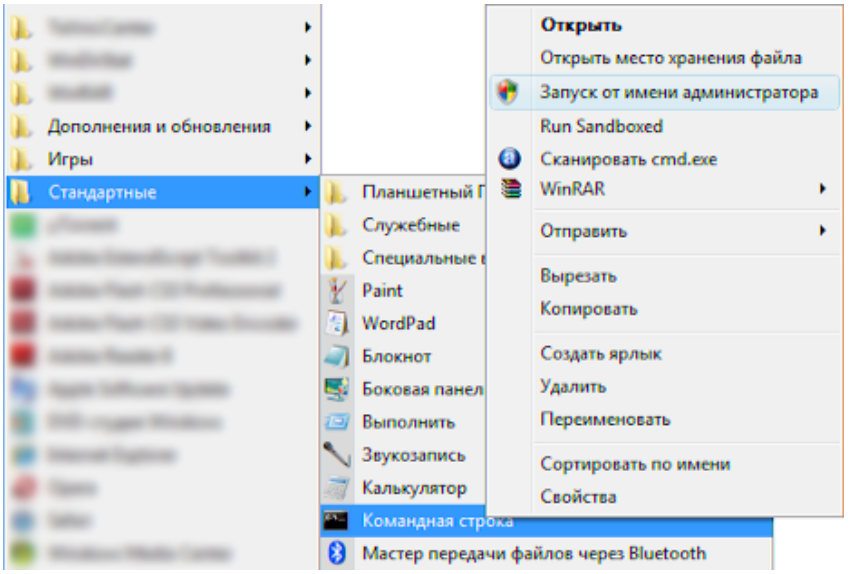



Рисунок 3. Запуск інсталятора з командного рядка

2. Ввести в командному рядку повний шлях до програми інсталяції і натиснути Enter (Рисунок 3).




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).

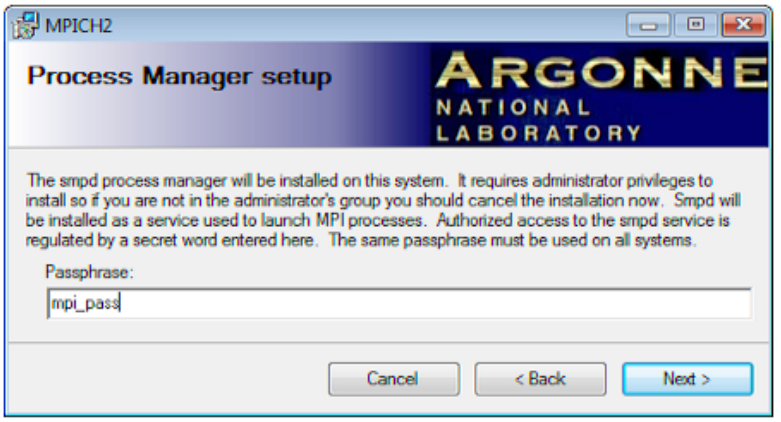



Рисунок 4. Вказівка пароля для доступу до менеджера процесів

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

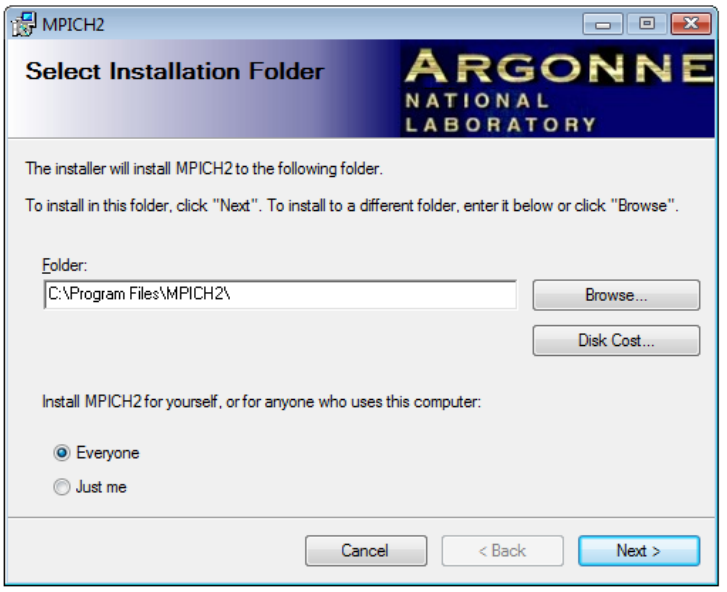



Рисунок 5. Вказівка шляху установки

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. «MPICH

2 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.

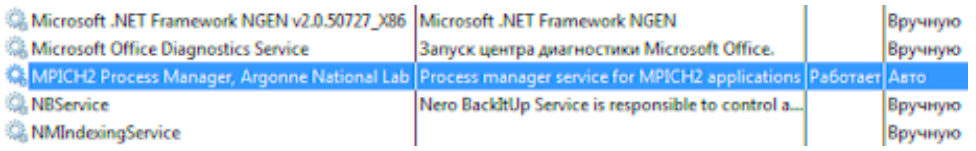



Рисунок 6. Служба «MPICH 2 Process Manager» в списку служб

Далі слід перевірити, чи дозволений доступ в мережу для MPICH. В менюв 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 7).

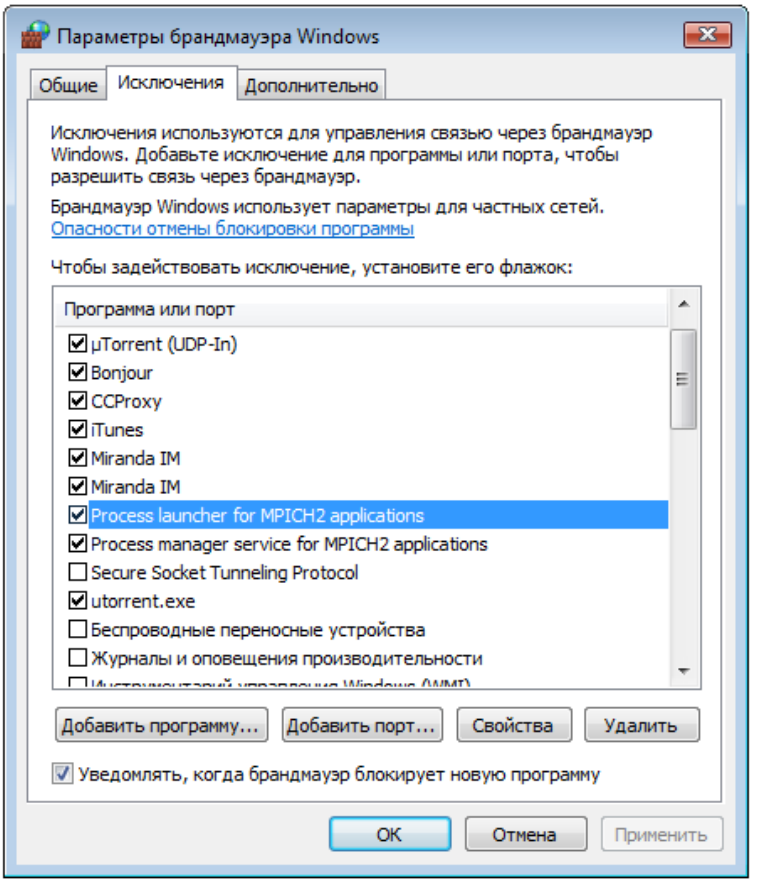



Рисунок 7. Програми MPICH в списку виключень брандмауера

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».

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

1. Необхідно **створити 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 8.

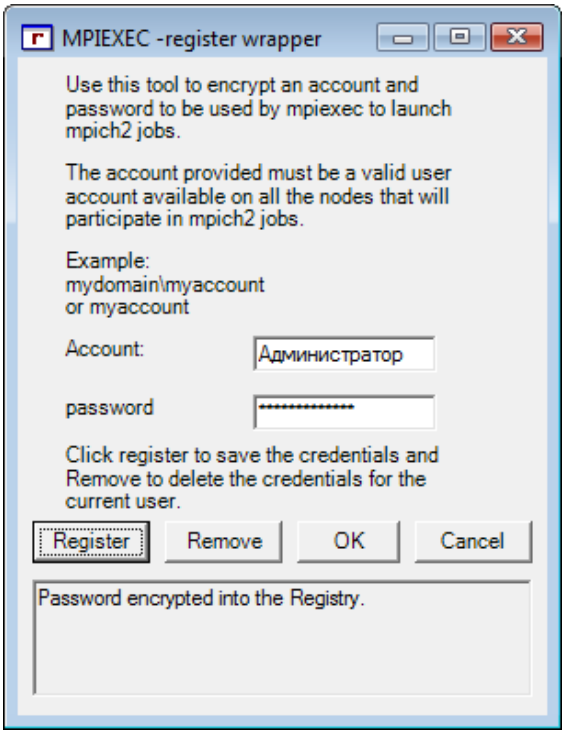



Рисунок 8. Програма Wmpiregister

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».

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. В цьому випадку зверніться за допомогою до викладача.

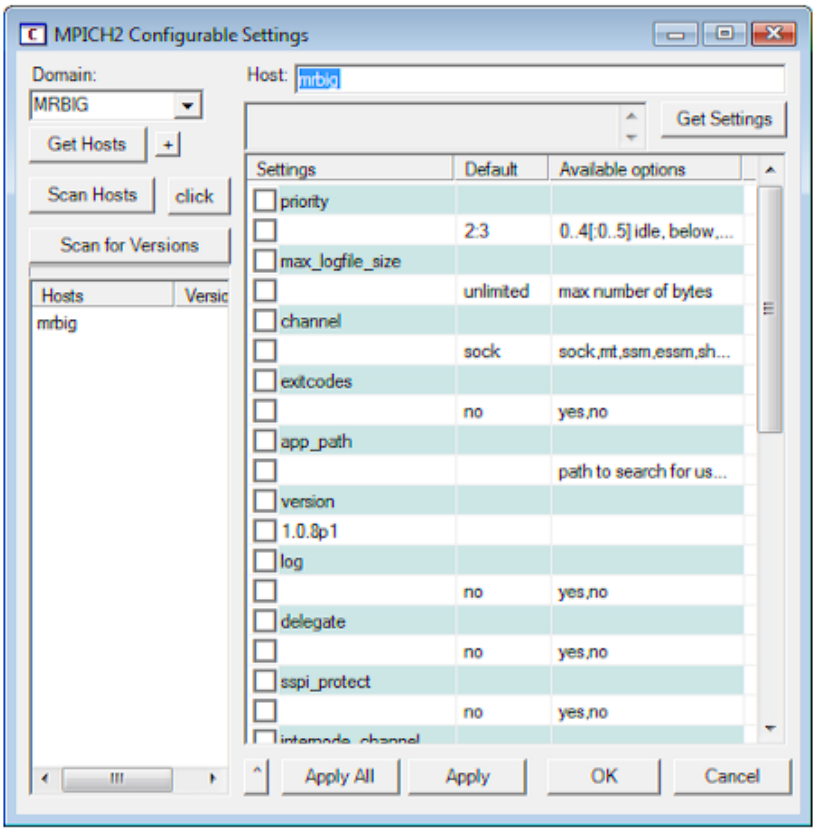



Рисунок 9. Програма Wmpiconfig

Програма Wmpiconfig призначена для налаштування менеджерів процесів на поточному комп’ютері і інших комп’ютерах мережі. Для цього вона під’єднується до менеджерів процесів на вибраних комп’ютерах, читає наявні у них налаштування, і 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 - программа.

- 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 галочки ліворуч.
- Кнопка «Apply» застосовує виділені галочкою 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 10), і натиснути кнопку «Apply».

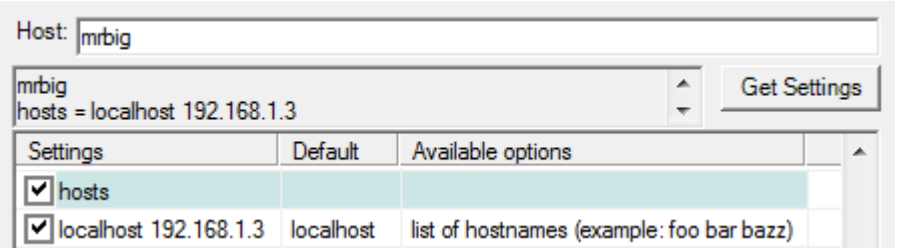



Рисунок 10. Вказуємо список доступних обчислювальних вузлів

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».

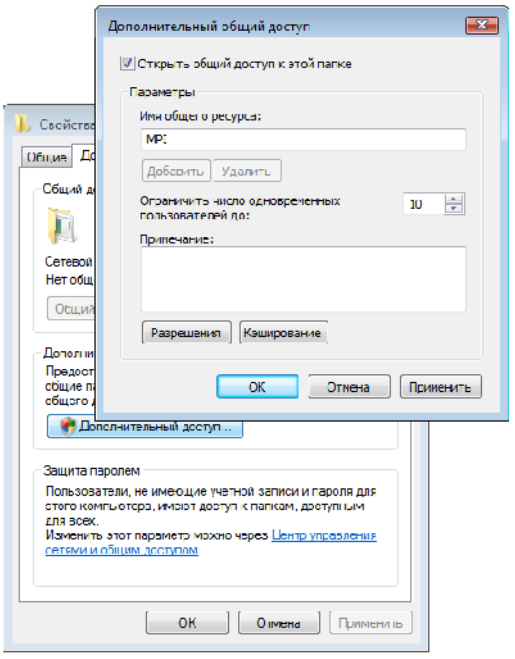



Рисунок 11. Вікно властивостей папки

7. У вікні, що з’явилося, виберіть вкладку «Доступ» (Рисунок 11), в якій натисніть кнопку «Додатковий доступ». У вікні, що з’явилося, «Додатковий загальний доступ» поставте галочку «Відкрити загальний доступ до цієї теки», і встановіть «число одночасних користувачів» таким, щоб воно перевищувало кількість комп’ютерів мережі, призначених для запуску MPI –програм.

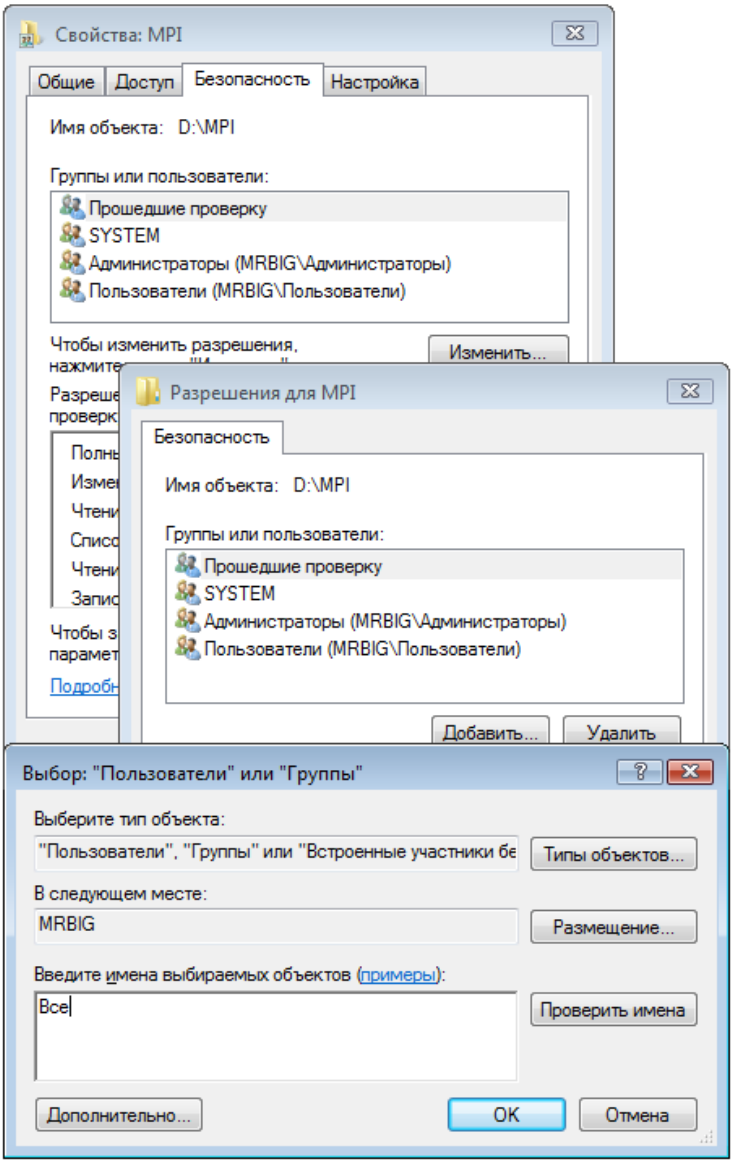



Рисунок 12. Додавання дозволів для доступу до теки

8. Натисніть «ОК» 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 «ОК» в двох вікнах. У вкладці «Безпека» у верхньому списку повинна додатися рядок «Все», при виборі якої в списку дозволів повинні стояти галочки 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»).

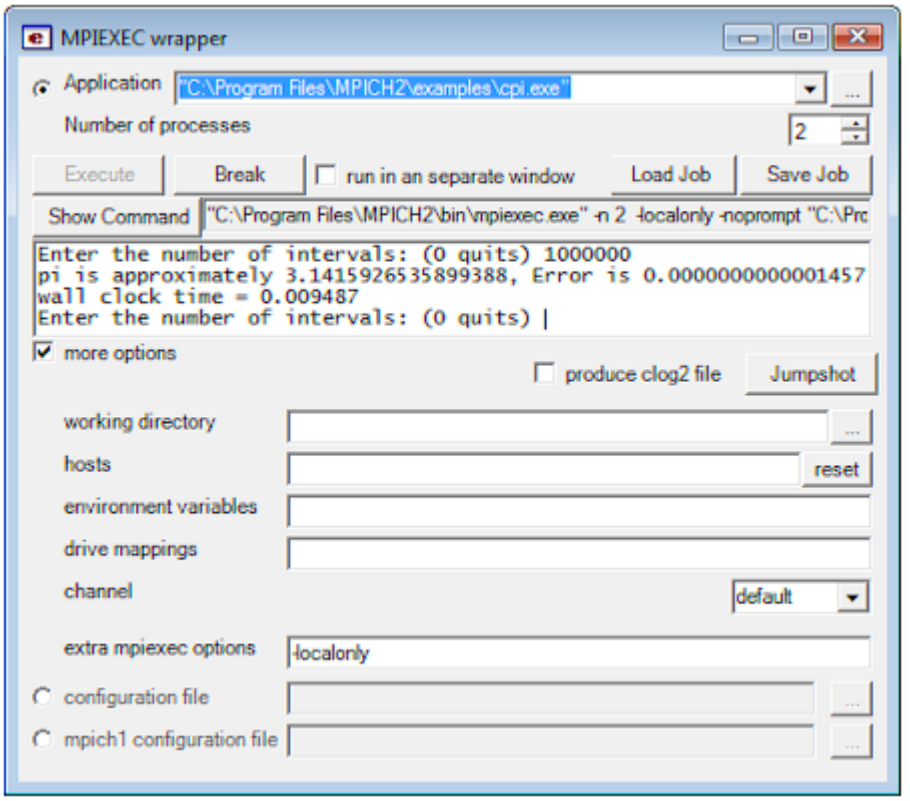



Рисунок 13. Програма Wmpiexec

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.

Разом з MPICH поставляється зразок MPI-програми: C:\Program Files\MPICH2\examples\cpi.exe (початковий код також є). Це проста програма, що приблизно обчислює значення числа Пі шляхом чисельного обчислення наступного інтеграла :

$$\pi = \int_0^1 \frac{4}{1+x^2}\,dx. \qquad (1)$$

10. Спочатку потрібно запустити **один, два, і чотири процеси на одному комп'ютері**. Щоб MPICH не розподіляв процеси, що запускалися, між наявними вузлами, відключимо роботу з мережею; для цього існує ключ командного рядка - localonly. При його додаванні менеджер процесів не використовується. Це дуже корисно, якщо з яких-небудь причин MPICH не вдається настроїти. Введіть цей ключ в поле «extra mpiexec options» (Рисунок 13).
11. У полі «Application» введіть шлях до програми cpi.exe. Оскільки програма запускається на одному комп'ютері, шлях можна вводити локальний. «Number of processes» введіть рівним 1, і натисніть кнопку «Execute». Програма запуститься і запитає число інтервалів для чисельної інтеграції. Введіть 100 000 000, і натисніть Ctrl+Enter . Програма порахує число Пі, і виведе час, витрачений на обчислення .

12. Проведіть обчислення з 100 мільйонами інтервалів ще кілька разів, щоб визначити мінімальний час. Після цього введіть 0, і натисніть Ctrl+Enter. Програма завершиться.
13. Далі слід запустити **MPI -програму на двох чи більше комп'ютерах**. Для цього скопіюйте програму ісрі.exe на загальний мережевий диск ереконайтеся, що програма видно з усіх комп'ютерів по одній і тій же адресі. Тепер найголовніше. Оскільки екземпляри MPI -програми працюють і обмінюються даними по мережі незалежно від менеджера процесів, їх теж треба внести в список виключень брандмауера Windows. Додайте ісрі.exe (прямо з мережевого ресурсу) в список виключень брандмауера на усіх комп'ютерах мережі (Рисунок 7). Відключення брандмауера на усіх комп'ютерах — не завжди допомогає.
14. Виберіть число процесів і натисніть кнопку Execute.
15. Оформити звіт до роботи, який включає тему, мету, хід роботи з наведеними screenshot-ами (Wmpiregister, Wmpiconfig, Wmpiexec для варіантів запуску 1, 2, 4 процеси в режимі localhost, та 1, 2, 4 процеси із залученням двох чи більше комп'ютерів ) і короткими пояснення до них, висновки.

**Контрольні запитання**

1. Що таке стандарт MPI?
2. Для чого необхідно виконати файл mpich2-1.0.7-win32-ia32.msi ?
3. Які налаштування проекту з MPI необхіно виконати?
4. Як виконати компілювання проекту?
5. Які утіліти є в пакеті MPI, їх призначення?
6. Як інсталювати SMPD?
7. Призначення і послідовність роботи з Wmpiconfig.
8. Призначення і послідовність роботи з Wmpiexec.

# Лабораторна робота №3

**Тема робота:** Створення простих MPI-програм в середовищі Visual Studio

**Мета роботи:** ознайомитися з методикою налаштування компілятора Visual Studio для роботи з MPICH та вивчення операцій попарного і колективного обміну повідомленнями в MPI.

## Теоретичні відомості

MPI - message passing interface - бібліотека функцій, та пакет утиліт, призначених для підтримки роботи паралельних процесів в термінах передачі повідомлень.

Номер процесу - ціле додатнє число, що є унікальним атрибутом кожного процесу.

Атрибути повідомлення - номер процесу-відправника, номер процесу-одержувача і ідентифікатор повідомлення. Для них заведена структура MPI_Status, що містить три поля: MPI_Source (номер процесу відправника), MPI_Tag (ідентифікатор повідомлення), MPI_Error (код помилки); можуть бути і додаткові поля.

Ідентифікатор повідомлення (msgtag) - атрибут повідомлення, що є цілим додатнім числом, що лежить в діапазоні від 0 до 32767.

Процеси об’єднуються в групи, можуть бути вкладені групи. Усередині групи всі процеси перенумеровані. З кожною групою асоційований свій комунікатор. Тому при здійсненні пересилки необхідно вказати ідентифікатор групи, усередині якої проводиться ця пересилка. Всі процеси містяться в групі з визначеним ідентифікатором MPI_COMM_WORLD.

При описі процедур MPI користуватимемося словом OUT для позначення "вихідних параметрів", тобто таких параметрів, через які процедура повертає результати.

Загальні процедури MPI

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

MPI_Init - ініціалізація паралельної частини додатку. Реальна ініціалізація для кожного додатку виконується не більше одного разу, а якщо MPI вже ініціалізував, то ніякі дії не виконуються і відбувається негайне повернення з підпрограми. Всі MPI-процедури, що залишилися, можуть бути викликані тільки після виклику MPI_Init.

Повертає: у разі успішного виконання - MPI_SUCCESS, інакше - код помилки. (Те ж саме повертає і вся решта функцій)

**int MPI_Finalize( void )**

MPI_Finalize - завершення паралельної частини додатку. Всі подальші звернення до будь-яких MPI-процедур, у тому числі до MPI_Init, заборонені. До моменту виклику MPI_Finalize деяким процесом всі дії, що вимагають його участі в обміні повідомленнями, повинні бути завершені.

Складний тип аргументів MPI_Init передбачений для того, щоб передавати всім процесам аргументи main:

```
int main(int argc, char** argv)
{
    MPI_Init(&argc &argv);
       ...
    MPI_Finalize();
}
```

**int MPI_Comm_size( MPI_Comm comm, int* size)**

Визначення загального числа паралельних процесів в групі comm.

comm - ідентифікатор групи

OUT size - розмір групи (OUT означає, шо цей параметр є вихідним результатом функції )

**int MPI_Comm_rank( MPI_Comm comm, int* rank)**

Визначення номера процесу в групі comm. Значення, що повертається за адресою &rank, лежить в діапазоні від 0 до size_of_group-1.

comm - ідентифікатор групи

OUT rank - номер викликаючого процесу в групі comm

**double MPI_Wtime(void)**

Функція повертає астрономічний час в секундах (дійсне число), що пройшов з деякого моменту у минулому. Гарантується, що цей момент не буде змінений за час існування процесу.

Для реалізації процедур попарного обміну повідомленнями використовуються функції MPI_Send (для передавання) і MPI_Recv (для приймання), які мають наступний формат:

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

де

- `buf` – адреса буфера пам’яті, де розміщується повідомлення;
- `count` – кількість елементів у повідомленні,
- `type` – тип елементів у повідомленні,
- `dest` – номер процесу отримувача,
- `tag` – тег повідомлення, використовується для ідентифікації повідомлення. Це число від 0 до 255.
- `comm` - група процесів, в рамках якої відбувається передавання.

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

де

- `buf` – адреса буфера пам’яті, куди прийматиметься повідомлення;
- `count` – кількість елементів у повідомленні,
- `type` – тип елементів у повідомленні,
- `dest` – номер процесу, від якого отримується повідомлення,

- `tag` – тег повідомлення, використовується для ідентифікації повідомлення. Значення тегу повинно співпадати при відправці і прийманні.
- `comm` - група процесів, в рамках якої відбувається передавання.

При заданні номера процесу і тегу при прийманні можна використовувати константи відповідно MPI_ANY_SOURCE та MPI_ANY_TAG, що означають прийняти повідомлення з від довільного процесу та з будь-яким тегом.

Приклад використання функцій:

```
MPI_Send(&ProcRank,1,MPI_INT,0,0,MPI_COMM_WORLD);
MPI_Recv(&RecvRank, 1, MPI_INT, MPI_ANY_SOURCE,
         MPI_ANY_TAG, MPI_COMM_WORLD, &Status);
```

## Завдання для виконання

### 1. Налаштування проекту для виконання

### MPI -програми в Visual Studio

У Visual Studio 2010 налаштувань 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 настроїти Visual Studio, щоб він знаходив заголовні файли і .lib -библиотеки 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:

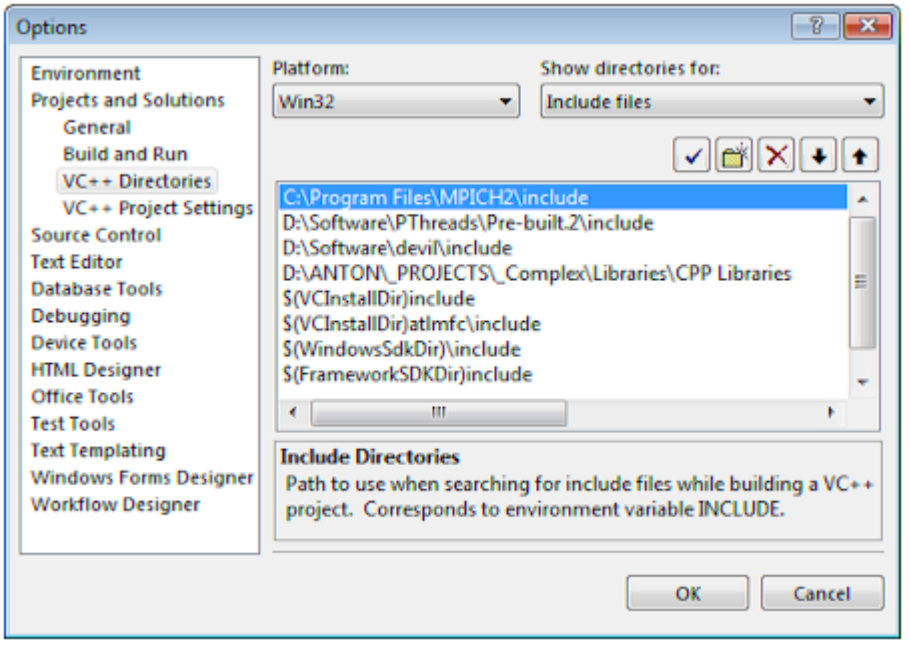



Рисунок 14. Налаштування шляху до заголовних файлів MPICH

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

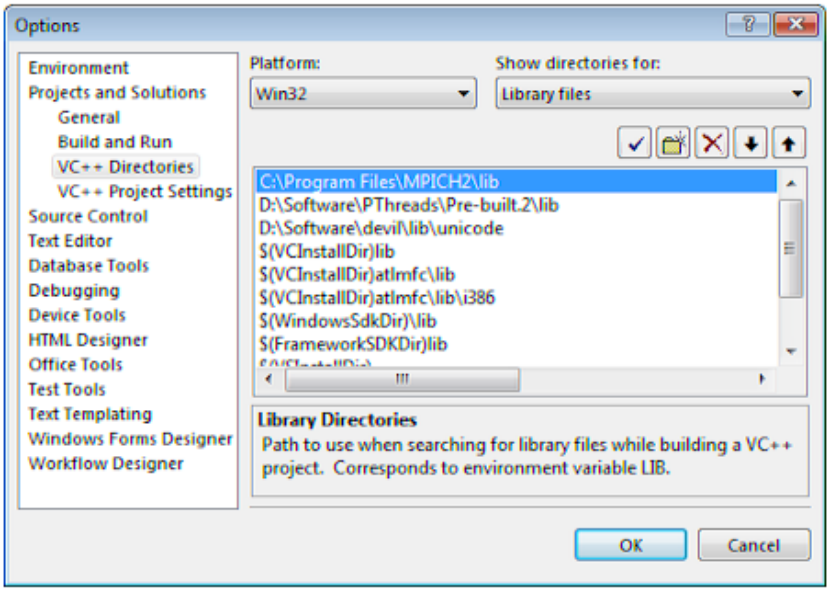



Рисунок 15. Налаштування шляху до бібліотечних файлів MPICH

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)

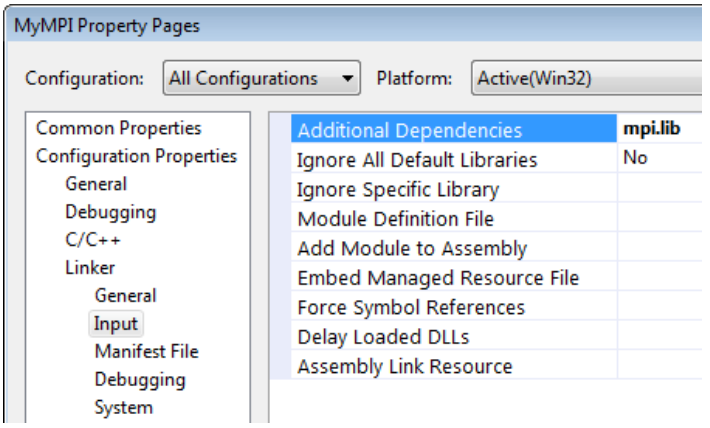



Рисунок 16. Додавання mpi.lib до програми

Тепер створіть консольний проект відповідно до рисунків 17,18.

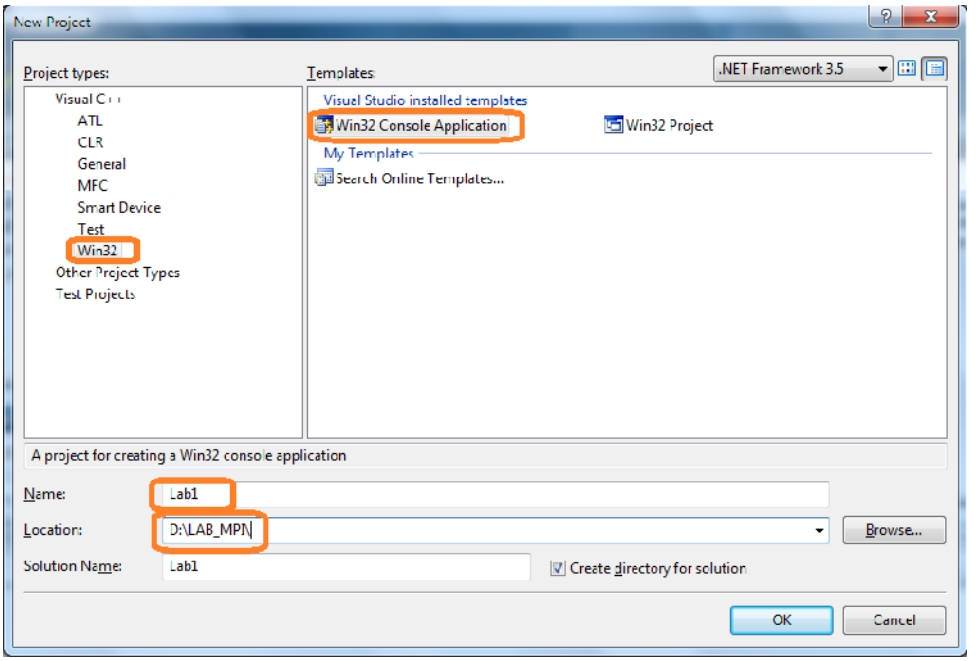
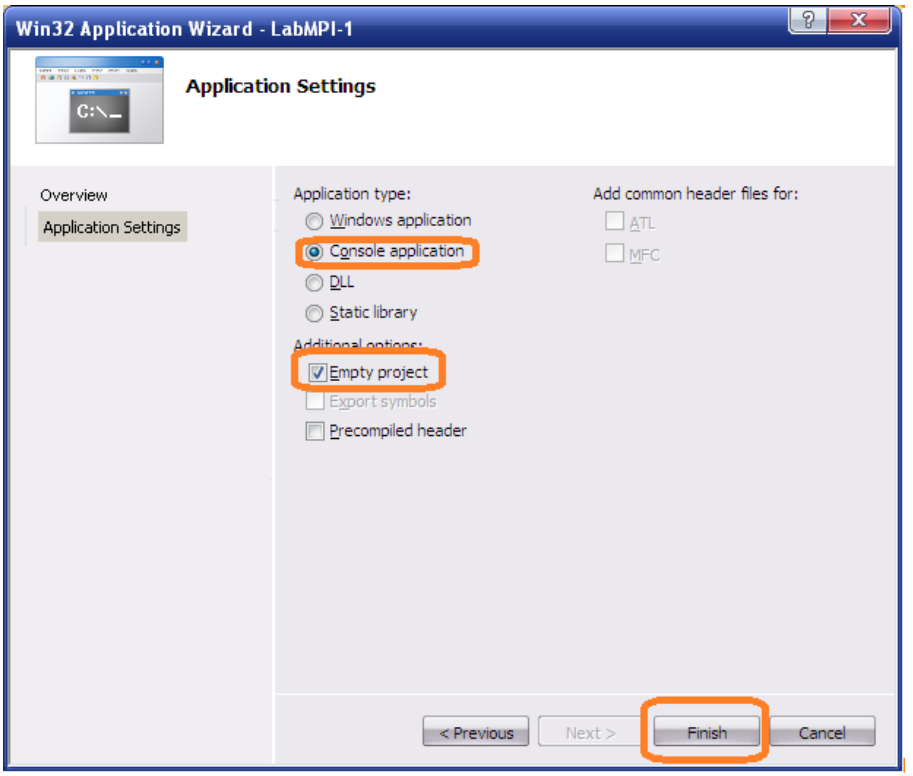



Рисунок 17. Створення нового проекту




Срисунок 18. Задання властивостей створюваного проекту

**2. Створення простої MPI -програми**

Розглянемо як приклад програму, що виводить на екран повідомлення «Hello» від кожного з процесів.

```
#include <stdio.h>
#include <string.h>
#include "mpi.h"
/* Довжина буфера повідомлень */
#define BUF_LEN 256
int main (int argc, char *argv[])
{
  int my_rank;        /* ранг поточного процесу */
  int p;              /* загальне число процесів */
```

```
int source;                /* ранг відправника */
int dest;                  /* ранг одержувача */
int tag = 0;               /* тег повідомлень */
char message[BUF_LEN];     /* буфер для повідомлення */
MPI_Status status;         /* інформація про отримане
                              повідомлення */
/* Почати роботу з MPI */
MPI_Init (&argc, &argv);
/* Отримати номер поточного процесу в групі всіх
процесів */
MPI_Comm_rank (MPI_COMM_WORLD, &my_rank);
/* Отримати загальну кількість запущених процесів */
MPI_Comm_size (MPI_COMM_WORLD, &p);
/* Посилаємо повідомлення процесу 0, який їх виводить
  на екран */
if (my_rank ! = 0)
   {
   /* Створюємо повідомлення */
   sprintf(message,"Hello from process %d!",my_rank);
   /* Відправляємо його процесу 0 */
   dest = 0;
   MPI_Send (message, strlen (message) + 1, MPI_CHAR,
   dest, tag, MPI_COMM_WORLD);
   }
else
 {
   /* В процесі 0: одержуємо повідомлення від процесу
   1...,p-1 і виводимо його на екран */
   for (source = 1; source < p; source++)
    {
```

```
        MPI_Recv (message, BUF_LEN, MPI_CHAR, source, tag,
        MPI_COMM_WORLD &status);
        printf ("%s\n", message);
        }
    }
  /* Закінчуємо роботу з MPI */
   MPI_Finalize ();
   return 0;
}
```

Використовуючи теоретичні відомості створити MPI–програму, яка працює з множиною процесів (від 2 до 6), і використовуючи функції попарного передавання даних передає і виводить в нульовому процесі прізвище автора і номер процесу, від якого надійшло повідомлення.

Оформити звіт до роботи, який включає тему, мету, хід роботи з наведеними screenshot-ами (лістинг програми, Wmpiexec для варіантів запуску 2, 4, 6 процесів) і короткими пояснення до них, висновки.

## Контрольні запитання

1. Який мінімальний набір засобів є достатнім для організації паралельних обчислень в системах з розподіленою пам’яттю?
2. Що слід розуміти під паралельною програмою?
3. В чому відмінність понять процесу і процесора?
4. Який мінімальний набір функцій MPI дозволяє почати розробку паралельних програм?
5. Як описуються передавані повідомлення?
6. Як можна організувати прийом повідомлень від конкретних процесів?
7. Як визначити час виконання MPI програми?

## Лабораторна робота №4

**Тема роботи:** Вивчення колективного обміну повідомленнями засобами MPI

**Мета роботи:** ознайомитися з функціями колективного обміну та розробити програму з їх використанням.

## Теоретичні відомості

Всі описувані нижче можливості MPI є надмірними в тому значенні, що будь-яка програма може бути написана без їх використання. Дійсно, всякий колективний обмін повідомленнями може бути замінений на відповідний цикл попарних обмінів. Проте це приведе до значного зниження швидкості роботи програми, оскільки, по-перше, найпростіше рішення (один з процесів розсилає дані всім іншим) не годиться (в кожний момент часу працюють тільки два процеси), а, по-друге, будь-яке рішення (наприклад розсилка за принципом бінарного дерева: перший процес посилає дані другому, потім перший і другий посилають дані третьому і четвертому відповідно, і т. д.) не враховуватиме специфіку конкретної обчислювальної установки.

Повідомлення, які посилаються за допомогою функцій колективного обміну, не можуть бути отримані за допомогою функцій попарного обміну і навпаки. Нагадаємо також, що в перших версіях MPI всі функції колективного обміну були синхронними і це наклало відбиток на їх синтаксис.

Для синхронізації процесів можна використовувати функцію MPI_Barrier. Її прототип:

```
int MPI_Barrier (MPI_Comm comm);
```

де вхідний параметр

comm – комунікатор.

Функція MPI_Barrier блокує процес що її викликав до тих пір, поки всі процеси в групі з ідентифікатором comm (комунікаторі) не викличуть цю функцію.

Для розсилки даних з одного процесу всім іншим в його групі можна використовувати функцію MPI_Bcast. Її прототип:

```
int MPI_Bcast (void *buf, int count MPI_Datatype
    datatype, int root, MPI_Comm comm);
```

де вхідні параметри:

buf

– в процесі з номером root – адреса буфера з даними, які посилаються (вхідний параметр),

– в решті процесів групи comm – адреса буфера з даними, де слід розмістити отримане повідомлення (вихідний параметр),

count

– в процесі з номером root – число елементів (типу datatype) в буфері,

– в решті процесів групи comm – максимальне число елементів (типу datatype) в буфері

datatype – тип даних кожного з елементів буфера,

root – ранг (номер) відправника в комунікаторі (групі) comm,

comm – комунікатор,

і вихідний параметр (результат):

buf – в процесах групи comm з номерами, відмінними від root, – отримане повідомлення.

Ця функція повинна бути викликана у всіх процесах групи comm з однаковими значеннями для аргументів root і comm. Також в більшості реалізацій MPI потрібно, щоб значення аргументів count і datatype були однаковими у всіх процесах групи comm. Ця функція розсилає повідомлення buf з процесу з номером root всім процесам групи comm. Оскільки всі процеси

викликають цю функцію одночасно і з однаковими аргументами, то, на відміну від функцій MPI_Send і MPI_Recv, поля tag і status відсутні.

Часто вимагається виконати в якомусь значенні зворотню до виконуваної функцією MPI_Bcast операцію – переслати дані одному з процесів групи від всієї решти процесів цієї групи; при цьому часто потрібно отримати не самі ці дані, а деяку функцію від них, наприклад, суму всіх даних. Для цього можна використовувати функцію MPI_Reduce. Її прототип:

```
int MPI_Reduce (void *sendbuf, void *recvbuf,  int
    count, MPI_Datatype datatype, MPI_Op op, int
    root, MPI_Comm comm);
```

де вхідні параметри:

sendbuf – адреса буфера з даними

count – число елементів (типу datatype) в буфері

datatype – тип даних кожного з елементів буфера

op – ідентифікатор операції (типу MPI_Op), яку потрібно здійснити над пересланими даними для отримання результату в буфері recvbuf; див. в табл. 4 можливі значення цього аргументу,

root – ранг (номер) одержувача в комунікаторі (групі) comm,

comm – комунікатор,

і вихідний параметр (результат):

recvbuf – покажчик на буфер, де вимагається отримати результат; використовується тільки в процесі з номером root в групі comm.

Ця функція повинна бути викликана у всіх процесах групи comm з однаковими значеннями для аргументів root, comm, count, datatype, op.

Якщо вимагається, щоб результат, отриманий за допомогою функції MPI_Reduce, став відомий не тільки одному процесу (з номером root в групі comm), а всім процесам групи, то можна використовувати MPI_Bcast, але більш ефективним рішенням є функція MPI_Allreduce. Її прототип:

```
int MPI_Allreduce (void *sendbuf, void *recvbuf, int
```

```
count, MPI_Datatype datatype, MPI_Op op, MPI_Comm
comm);
```

де вхідні параметри:

sendbuf – адреса буфера з даними,

count – число елементів (типу datatype) в буфері,

datatype – тип даних кожного з елементів буфера,

op – ідентифікатор операції (типу MPI_Op), яку потрібно здійснити над пересланими даними для отримання результату в буфері recvbuf; див. в табл. 4 можливі значення цього аргументу,

comm – комунікатор,

і вихідний параметр (результат):

recvbuf – покажчик на буфер, де вимагається отримати результат.

Функція аналогічна MPI_Reduce, але результат утворюється в буфері recvbuf у всіх процесах групи comm, тому немає аргументу root.

Таблиця 4. Операції над даними в MPI

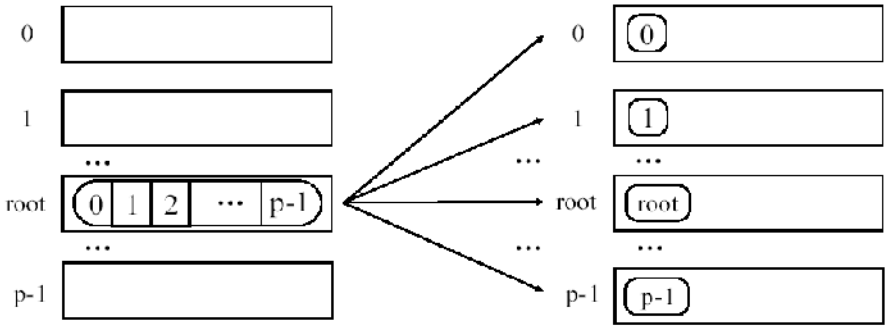
| Операція MPI | Значення |
|---|---|
| MPI_MAX | максимум |
| MPI_MIN | мінімум |
| MPI_SUM | сума |
| MPI_PROD | множення |
| MPI_LAND | логічне «і» |
| MPI_BAND | побітове «і» |
| MPI_LOR | логічне «або» |
| MPI_BOR | побітове «або» |
| MPI_LXOR | логічне «виключне або» |
| MPI_BXOR | побітове «виключне або» |
| MPI_MAXLOC | максимум і його позиція |
| MPI_MINLOC | мінімум і його позиція |

Окрім функцій колективного обміну даних в MPI, а саме функції широкомовної розсилки MPI_Bcast та функції збору даних від всіх процесорів MPI_Reduce, які відповідно передають всім процесам одні й ті ж самі дані, або отримують від всіх процесів дані і утворюють одне число, в MPI передбачені функції колективного передавання даних, які дозволяють обмінюватися різними даними.

Узагальнена операція передачі даних від одного процесу всім процесам (розподіл даних) відрізняється від широкомовної розсилки тим, що процес передає процесам дані, що відрізняються (рис. 18). Виконання даної операції може бути забезпечене за допомогою функції:

int **MPI_Scatter**(void *sbuf, int scount, MPI_Datatype stype, void *rbuf, int rcount, MPI_Datatype rtype, int root, MPI_Comm comm), де:

- sbuf, scount, stype - параметри передаваного повідомлення (scount визначає кількість елементів, що передаються на кожен процес);
- rbuf, rcount, rtype - параметри повідомлення, що приймається в процесах;
- root – номер процесу, що виконує розсилку даних;
- comm – група процесів(комунікатор), в рамках якого виконується передача даних.




а) до виконання б)після виконання

Рисунок 18. Демонстрація роботи функції **MPI_Scatter.**

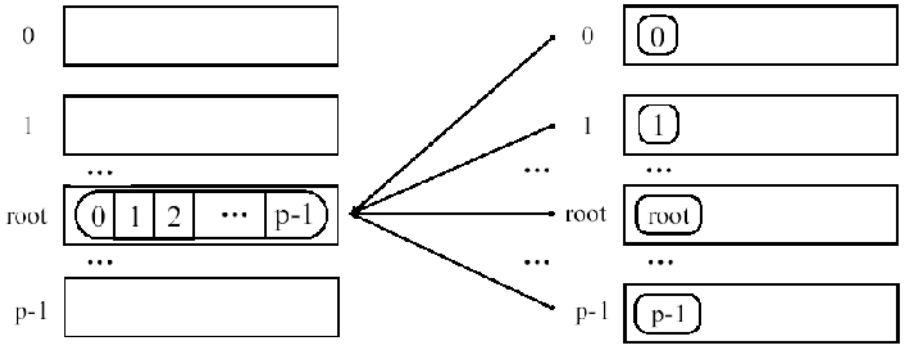
При виклику цієї функції процес з номером *root* проведе передачу даних всім іншим процесам в комунікаторі. Кожному процесу буде відправлено *scount* елементів. Процес з номером 0 отримає блок даних із *sbuf* з елементів із індексами від 0 до *scount-1*, процесу з номером 1 буде відправлений блок з елементів із індексами від *scount* до *2* scount-1* і т.д. Тим самим, загальний розмір повідомлення, що відправляється, повинен бути рівний *scount * p* елементів, де *p* є кількість процесів в комунікаторі *comm*.

Слід зазначити, оскільки функція *MPI_Scatter* визначає колективну операцію, виклик цієї функції при виконанні розсилки даних повинен бути забезпечений в кожному процесі комунікатора. Відзначимо також, що функція *MPI_Scatter* передає всім процесам повідомлення однакового розміру.

Операція узагальненої передачі даних від всіх процесорів одному процесу (*збір даних*) є зворотною до процедури розподілу даних (рис. 19). Для виконання цієї операції в MPI призначена функція:

int **MPI_Gather**(void *sbuf, int scount, MPI_Datatype stype, void *rbuf, int rcount,
MPI_Datatype rtype, int root, MPI_Comm comm), де:

- **sbuf**, **scount**, **stype** - параметри передаваного повідомлення;
- **rbuf**, **rcount**, **rtype** - параметри повідомлення, що приймається;
- **root** – ранг процесу, що виконує збір даних;
- **comm** - коммуникатор, в рамках якого виконується передача даних.




**процеси** **процеси**

а) до виконання б)після виконання

Рисунок 19 Демонстрація роботи функції **MPI_Gather.**

При виконанні функції *MPI_Gather* кожен процес в комунікаторі передає дані з буфера *sbuf* на процес з номером *root*. Процес з номером *root* збирає всі отримувані дані в буфері *rbuf* (розміщення даних в буфері здійснюється відповідно до номерів процесів-відправників повідомлень). Для того, щоб розмістити всі дані, що поступають, розмір буфера *rbuf* повинен бути рівний *scount* * *p* елементів, де *p* є кількість процесів в комунікаторі *comm*. Функція *MPI_Gather* також визначає колективну операцію, і її виклик при виконанні збору даних повинен бути забезпечений в кожному процесі коммуникатора.

Слід зазначити, що при використанні функції *MPI_Gather* збір даних здійснюється тільки на одному процесі. Для отримання всіх зібраних даних на кожному з процесів комунікатора необхідно використовувати *функцію збору і розсилки*:

int **MPI_Allgather**(void *sbuf, int scount, MPI_Datatype stype, void *rbuf, int rcount, MPI_Datatype rtype, MPI_Comm comm),

де параметри аналогічні, як і в попередній функції.

## Завдання для виконання

Задано одномірний масив $A_i$, де $0 \le i \le 10000$ відповідно до задач з табл.5. Необхідно написати програму, орієнтовану на мультипроцесорне виконання, кожен з яких виконує:

- генерацію свого фрагмента вектора $A_i$;
- обробку фрагмента відповідно до завдання Z1;
- передавання результату обробки процесу з номером 0.

Нульовий процес повинен обробити отримані від інших процесів дані відповідно до завдання Z2 і вивести їх на екран.

Конкретизація завдань згідно варіанту наведена в таблиці 6.

Необхідно передбачити визначення часу виконання програми. Кожен варіант програми виконати на одному, двох, трьох і чотирьох вузлах. Отримані часові показники використати для побудови двох графіків залежності часу виконання програми від кількості вузлів. По отриманих графіках зробити відповідні висновки.

У звіті до роботи обов’язково навести:

- тему і мету роботи;
- повністю сформульоване завдання відповідно варіанту;
- програму використанням операцій тільки попарного обміну;
- програму з використанням операцій колективного обміну.
- графік залежності часу виконання програми від кількості вузлів (від 1 до 4-ох);

Таблиця 5 Задачі для виконання

| № задачі | $A_i$ | Z1 | Z2 |
|---|---|---|---|
| 1 | $\frac{i \cdot \cos(i \cdot 2)}{3}$ | Знайти добуток найменшого додатного і найбільшого від’ємного елементів | Знайти суму отриманих від процесів значень |
| 2 | $\frac{\lvert\sin(i)\rvert}{5 \cdot (i+1)}$ | Знайти найбільший елемент | Знайти максимальне серед отриманих від процесів значень |
| 3 | $\frac{i^2 + 3i + 2}{\sqrt{i+1}}$ | Знайти півсуму найменшого і найбільшого елементів | Знайти добуток отриманих від процесів значень |
| 4 | $\frac{0.3\sqrt{i}}{\cos(i)+5}$ | Знайти різницю найменшого і найбільшого елементів | Знайти мінімальне серед отриманих від процесів значень |
| 5 | $\frac{\lg\sqrt{i+5}}{\tan(2*i)+10}$ | Знайти найменший елемент | Знайти максимальне серед отриманих від процесів значень |
| 6 | $\frac{\sqrt{i} \cdot \sin(i \cdot 5)}{\cos(i+5)}$ | Знайти півсуму найменшого від’ємного і найбільшого від’ємного елементів | Знайти суму отриманих від процесів значень |

| № задачі | $A_i$ | Z1 | Z2 |
|---|---|---|---|
| 7 | $\frac{\cos^2(i+1)}{\lg(i+1)}$ | Знайти найбільший від'ємний елемент | Знайти максимальне серед отриманих від процесів значень |
| 8 | $\frac{i^3+5i^2+6}{\lvert i\cdot\cos(i)\rvert}$ | Знайти добуток найменшого і найбільшого елементів | Знайти добуток отриманих від процесів значень |
| 9 | $\frac{0.1\cdot\log(i+11)}{\cos\ (i)+2}$ | Знайти різницю найменшого і найбільшого додатніх елементів | Знайти мінімальне серед отриманих від процесів значень |
| 10 | $\frac{\cos(\sqrt{i+5})}{\arctan(2*i)+1}$ | Знайти найменший додатній елемент | Знайти максимальне серед отриманих від процесів значень |

Таблиця 6 Варіанти завдань

| № варіанту | № задачі | Функції колективного обміну |
|---|---|---|
| 1-10 | 1-10 | MPI_Bcast, MPI_Reduce |
| 11-20 | 1-10 | MPI_Scatter, MPI_Gather |
| 21-30 | 1-10 | MPI_Scatter, MPI_Reduce |

## Контрольні запитання

1. Якими параметрами описуються передавані повідомлення у колективних функціях обміну?
2. Чим відрізняються функції колективного обміну MPI_Bcast та MPI_Reduce від функцій MPI_Scatter та MPI_Gather?
3. В чому функціональна різниця між функціями MPI_Gather MPI_Allgather?
4. Як визначити час виконання MPI програми?
5. Які можливі операції редукції даних?
6. Чим відрізняється колективне передавання інформації від попарного?

# Лабораторна робота №5

**Тема роботи:** Вивчення віртуальних топологій.

**Мета роботи:** Засвоєння основних принципів створення віртуальних топологій в MPI та роботи з ними.

## Теоретичні відомості

### 1. Поняття про віртуальні топології

Під топологією обчислювальної системи зазвичай розуміється структура вузлів мережі і ліній зв’язку між цими вузлами. Топологія може бути представлена у вигляді графа, в якому вершини – це процесори (процеси) системи, а дуги відповідають наявним лініям (каналам) зв’язку. Попарні операції передачі даних можуть бути виконані між будь-якими процесами одного і того ж комунікатора, а в колективні операції приймають всі процеси комунікатора. У цьому плані, логічна топологія ліній зв’язку між процесами в паралельній програмі має структуру повного графа (незалежно від наявності реальних фізичних каналів зв’язку між процесорами). Ствонення віртуальної топології породжує новий комунікатор, який представляє цю топологію. Комунікатор в MPI – це спеціально створений службовий об’єкт, що включає в своєму складі групу процесів і ряд додаткових параметрів (контекст), які використовуються при виконанні операцій передачі даних.

Зрозуміло, що фізична топологія системи є такою, що реалізовується апаратно і зміні не підлягає. Але, залишаючи незмінною фізичну основу, ми можемо організувати логічне представлення будь-якої необхідної віртуальної топології. Для цього достатньо, наприклад, сформувати той або інший механізм додаткової адресації процесів.

Використання віртуальних процесів може виявитися корисним через ряд різних причин. Віртуальна топологія, наприклад, може більше відповідати

наявній структурі ліній передачі даних. Використання віртуальних топологій може помітно спростити у ряді випадків і реалізацію паралельних алгоритмів.

У MPI підтримуються два види топологій - прямокутні решітки довільної розмірності (декартова топологія) і топологія графа будь-якого довільного вигляду. Слід зазначити, що наявні в MPI функції забезпечують лише отримання нових логічних систем адресації процесів, відповідних формованим віртуальним топологіям. Виконання ж всіх передавань повинно здійснюватися, як і раніше, за допомогою звичайних функцій передачі даних з використанням початкових номерів процесів.

Над топологіями можна виконувати різні операції. Декартові решітки можна розділяти на гіперплошщини, виділяючи деякі виміри. Дані можна зсувати вздовж вибраного виміру декартової решітки. Зсувом називають пересилку даних між процесами уздовж певного виміру. Уздовж вибраного виміру можуть бути організовані колективні обміни.

## 2. Декартові топології

Для того щоб створити структуру декартової решітки з комунікатором MPI_COMM_WORLD, необхідно задати наступні параметри:

- розмірність решітки (значення 2 відповідає плоскій двовимірній решітці);
- розмір решітки уздовж кожного виміру (розміри {10, 15}, наприклад, відповідають плосій прямокутній решітці, протяжність якої уздовж осі x складає 10 вузлів-процесів, а уздовж осі y - 15 вузлів);
- періодичність уздовж кожного виміру (решітка може бути періодичною, якщо процеси, що знаходяться на протилежних кінцях ряду і мають прямий зв’язок між собою).

MPI дає можливість системі оптимізувати відображення віртуальної топології процесів на фізичну за допомогою зміни порядку нумерації процесів в групі.

Функція MPI_Cart_create створює новий комунікатор, який представляє топологііію решітки. Це є колективною функція, вона повинна викликатись з усіх процесів комунікатора. Формат вункції:

int MPI_Cart_create(MPI_Comm oldcomm, int ndims, int *dims, int *periods, int reorder, MPI_Comm *cartcomm),

де - oldcomm - початковий комунікатор

- ndims - розмірність декартової решітки

- dims - масив довжини ndims, задає кількість процесів в кожному вимірі решітки

- periods - масив довжини ndims, визначає, чи є решітка періодичною уздовж відповідного виміру.

- reorder - параметр допустимості зміни нумерації процесів

- cartcomm – створюваний комунікатор з декартовою топологією.

Для визначення декартових координат процесу по його номеру можна скористатися функцією:

int MPI_Cart_coords(MPI_Comm comm, int rank, int ndims, int *coords)

де:

- comm – комунікатор з топологією решітки

- rank - номер процесу, для якого визначаються декартові координати

- ndims - розмірність решітки

- coords - декартові координати процесу, які визначаються функцією.

Зворотна дія – визначення номеру процесу по його декартових координатах – забезпечується за допомогою функції:

int MPI_Cart_rank(MPI_Comm comm, int *coords, int *rank)

де

- comm – комунікатор з топологією решітки

- coords - декартові координати процесу

- rank - номер процесу, що визначається функцією.

Між процесами, організованими в декартову решітку, можна виконувати два типи зсувів даних у групі з N процесів:

- циклічний зсув на J позицій уздовж ребра решітки. Дані від процесу K пересилаються процесу з номером (J + K) mod N;
- лінійний зсув на J позицій уздовж ребра решітки, коли дані в процесі K пересилаються процесу з номером J + K, якщо номер адресата знаходиться в межах між 0 і N.

Функція MPI_Cart_shift забезпечує отримання номерів процесів, з якими поточний процес (процес, що викликав функцію MPI_Cart_shift) повинен виконати обмін даними:

int MPI_Card_shift(MPI_Comm comm, int dir, int disp, int *source, int *dst),

де:

- comm – комунікатор з топологією решітки,
- dir - номер виміру, по якому виконується зсув.
- disp - величина зсуву (<0 – зсув до початку виміру)
- source – номер процесу, від якого повинні бути отримані дані
- dst - номер процесу якому повинні бути відправлені дані.

Слід зазначити, що функція MPI_Cart_shift тільки визначає номери процесів, між якими повинен бути виконаний обмін даними в ході операції зсуву. Безпосередня передачами даних може бути виконана, наприклад, за допомогою функції MPI_Sendrecv.

Для пояснення призначення параметрів функції MPI_Cart_create розглянемо приклад створення двомірної решітки 4x4, в якої рядки і стовпці мають кільцеву структуру(за останнім процесом слідує перший процес) :

```
// створення двомірної решетки4 x4
MPI_Comm GridComm;
int dims[2], periods[2], reorder = 1;
dims[0] = dims[1] = 4;
```

```
    periods[0] = periods[1] = 1;
    MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods,
reoreder, &GridComm);
```

Слід зазначити, що в силу кільцевої структури вимірів сформована у рамках прикладу топологія є тором.

### 3. Віртуальні топології графу

Для створення комунікатора з топологією типу граф в MPI призначена функція:

int MPI_Graph_create(MPI_Comm oldcomm, int nnodes, int *index, int *edges, int reorder, MPI_Comm *graphcomm),

де:

- oldcomm - початковий комунікатор
- nnodes - кількість вершин графу
- index - кількість вихідних дуг для кожної вершини
- edges - послідовний список дуг графу
- reorder - параметр допустимості зміни нумерації процесів
- cartcomm – створюваний комунікатор з топологією типу граф.

Наведемо ще дві корисні функції для роботи з топологіями графа. Кількість сусідніх процесів, в яких від процесу, що перевіряється, є вихідні дуги, можна визначити за допомогою функції:

int MPI_Graph_neighbors_count(MPI_Comm comm,int rank, int *nneighbors).

Отримання рангів сусідніх вершин забезпечується функцією:

int MPI_Graph_neighbors(MPI_Comm comm, int rank, int mneighbors, int *neighbors)

де mneighbors є розмір масиву neighbors.

Розглянемо приклад створення топології графу згідно з конфігурацією, що наведена на рисунку 20. Кількість процесів равна 5, порядки вершин (кількості вихідних дуг) набувають значень (4,1,1,1,1), а матриця інцидентності(номери вершин, для яких дуги є такими, що входять) має вигляд:

| Процеси | Лінії зв’язку |
|---|---|
| 0 | 1, 2, 3, 4 |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 0 |

Для створення топології з графом цього виду необхідно виконати наступний програмний код:

```
// створення топології типу зірка
int index[] = { 4,1,1,1,1 };
int edges[] = { 1,2,3,4,0,0,0,0 };
MPI_Comm StarComm;
MPI_Graph_create(MPI_COMM_WORLD, 5, index, edges, 1,
&StarComm);
```

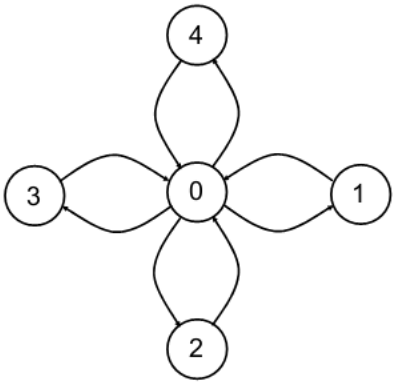



Рисунок 20. Приклад топологі для реалізації функції MPI_Graph_create

## Завдання для виконання роботи

Реалізувати засобами MPI віртуальну декартову топологію відповідно до варіанту, заданого в таблицях 7 та 8. В кожному процесі за допомогою функцій MPI_Card_coords та MPI_Cart_rank визначити координати та відповідні їм номери процесів. Переконатися, що кожній координаті відповідає один номер процесу. Побудувати таблицю відповідності номеру процесу та його координат.

Табл.7

| Варіант | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Решітка 2x3, періодична по першому виміру | Решітка 2x2x2 періодична по другому виміру | Решітка 3x2 періодична по всіх вимірах | Решітка 3x3 неперіо-дична | Гіперкуб третього порядку | Гіперкуб другого порядку | Гіперкуб 3-го порядку, періодичний по всіх вимірах | Гіперкуб другого порядку періодичний по всіх вимірах | Кільце на 8 вузлів | Кільце на 5 вузлів |

За допомогою функції MPI_Cart_shift виконати (циклічний) зсув заданого самостійно масиву із 10 цілих випадкових чисел. В звіті навести проведені транзакції.

Для варіантів 11-20 потрібно засобами MPI реалізувати віртуальну топологію графу згідно варіанту. Для кожної вершини графу визначити кількість і номери сусідніх вершин.

Табл.2

| Варіант | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| Зірка, n=6 | Решітка 2x3 з одним цикліч-ним звязком | Решітка 3x3 двома цикліч-ними звязками | Кільце на 6 вузлів з 3-ма хордальн ими звязками | Лінійка на 4 вузли, кожен з яких має 1 дочірний | Зірка на 5 вузлів, вершини якої зєднані в кільце | Гіпекуб третього порядку | Повнозв’ язна, n=6 | Дерево, n=7 | Кільце на 3 вузли, кожен має 2 дочірні. |

## Контрольні запитання

1. Що розуміється в MPI під комунікатором?
2. Для чого потрібно створення нових комунікаторів?
3. Що таке віртуальна топологія в MPI і для чого вона потрібна?
4. Які види топологій передбачені в MPI?
5. Для яких задач може виявитися корисним використання віртуальних топологій?
6. Які функції для роботами з віртуальними топологіями ви знаєте, їх параметри?

## Лабораторна робота №6

**Тема робота:** Паралельне множення матриці на вектор.

**Мета роботи:** Вивчення паралельних алгоритмів матрично-векторного множення та їх реалізація в середовищі MPI.

### Теоретичні відомості

Матриці і матричні операції широко використовуються при математичному моделюванні найрізноманітніших процесів, явищ і систем. Матричні обчислення складають основу багатьох наукових і інженерних розрахунків в області обчислювальної математики, фізики, економіки і ін. Будучи обчислювально-трудомісткими, матричні обчислення є класичною задачею для застосування паралельних обчислень.

Послідовні алгоритми матично-векторного множення використовують класичні математичні методи, а при паралельній реалізації алгоритмів однією із важливих задач є розподіл фрагментів матриці між процесорами та подальша їх обробка. Найбільш загальні і широко використовувані способи розділення матриць полягають в розбитті даних на *смуги* (по вертикалі або горизонталі).

В загальному випадку в результаті множення матриці *A* розмірності *n x m* і вектора *B*, що складається з *n* елементів, отримуємо вектор C розміру *m*, кожен *i*-тий елемент якого є результат скалярного множення *i*-того рядка матриці *A* (позначимо цей рядок $a_i$) і вектора *B:*

$$c_i = \sum_{j=1}^{n} a_{ij} b_j, \qquad 1 \le i \le m.$$

Послідовний алгоритм множення матриці на вектор може бути представлений наступним чином:

```
for (i = 0; i < m; i++)
{    c[i] = 0;
```

```
    for (j = 0; j < n; j++)
        {
            c[i] += a[i][j]*b[j];
          }
}
```

При виконанні паралельних алгоритмів множення матриці на вектор, окрім матриці *A* необхідно розділити ще вектор *B* і вектор результату C. Елементи векторів можна *продублювати*, тобто скопіювати всі елементи вектора на всі процесори, які утворюють багатопроцесорну обчислювальну систему, або *розділити* між процесорами. Розділювати матрицю A можна по рядках або по стовбцях.

Для прикладу розглянемо паралельний алгоритм множення матриці на вектор з розділенням по рядках. При такому способі розділення даних базовою підзадачею може бути вибрана операція скалярного множення одного рядка матриці на вектор.

Для виконання базової підзадачі скалярного добутку процесор повинен містити відповідний рядок матриці *A* і копію вектора *B*. Після завершення обчислень кожна базова підзадача визначає один з елементів вектора результату *C*.

Для об’єднання результатів розрахунку і отримання повного вектора *C* на кожному з процесорів обчислювальної системи необхідно виконати операцію узагальненого збору даних, в якій кожен процесор передає свій обчислений елемент вектора C решті всіх процесорів. Цей крок можна виконати, наприклад, з використанням функції *MPI_Allgather* з бібліотеки MPI. У загальному вигляді схема інформаційної взаємодії підзадач в ході виконуваних обчислень показана на рисунку 19.

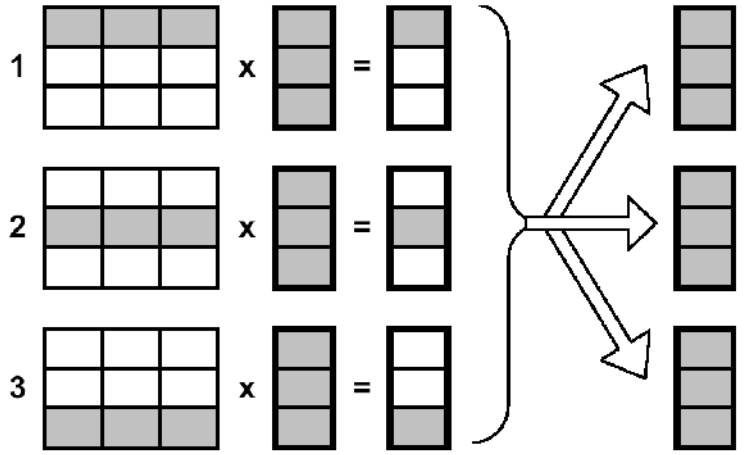



Рисунок 19. Організація обчислень при виконанні паралельного алгоритму множення матриці на вектор, при розділенні матриці по рядках.

Якщо кількість процесорів є меншою за кількість рядків матриці, то необхідно розподілити матрицю так званими смугами і кожен процесор обчислить відповідний фрагмент результату.

Таким чином, паралельна програма реалізації матрично-векторного множення складатиметься з наступних етапів:

1. Ініціалізація елементів матриці і вектора та їх розподіл між процесорами (реалізується в процесі 0).
2. Обчислення блока результуючого вектора (реалізується в кожному процесорі).
3. Пересилання фрагментів вектора-результату на нульовий процесор, або в кожен процесор (реалізується в кожному процесорі).
4. Вивід перших 10 елементів рядків матриці, вектора та результату.
5. Оцінка і вивід часу виконання алгоритму.

Кожен з наведених пунктів (крім 5) доцільно виконати у вигляді окремої функції.

Однією із найскладніших функцій цього алгоритму є функція, яка реалізує розподіл рядків матриці по всіх процесорах. Нижче наведено лістинг цієї функції. Функція здійснює розсилку вектора *B* і розподіл рядків початкової

матриці *A* по процесах обчислювальної системи. Слід зазначити, що коли кількість рядків матриці *n* не є кратною числу процесорів *p*, об'єм даних, що пересилаються, для процесів може бути різним, і для передачі повідомлень необхідно використовувати функцію *MPI_Scatterv* бібліотеки MPI. У функції використано наступні аргументи:

`*pMatrix` – вказівник на матрицю A;

`*pProcRows` – вказівник на смугу рядків матриці A, яка надсилається процесу з номером Rank;

`*pVector` – вказівник на вектор B;

`RowSize` – кількість елементів у рядку матриці A;

`RowNum` – кількість рядків матриці A;

`ProcRank` – глобальна змінна – поточний номер процесу.

```
void DataDistribution (double *pMatrix, int RowSize, int
          RowNum, double *pProcRows, double *pVector )
{  int *pSendNum; //кількість елементів, що пересилаються процесу
   int *pSendInd; // індекс першого елемента даних, що пересилається
процесу
   // Розсилка вектора b всім процесам програми
  MPI_Bcast(pVector, RowSize, MPI_DOUBLE, 0,
MPI_COMM_WORLD);
   // Виділення пам'яті
  pSendNum = new int [ProcNum];
  pSendInd = new int [ProcNum];
   // Визначення набору рядків для кожного процесу
  int ProcDataSize = RowSize * (RowNum/ProcNum);
  pSendInd[0] = 0;
  for ( int i=1; i<ProcNum; i++ ) {
        pSendInd[i] = pSendInd[i-1] + ProcDataSize;
```

```
        pSendNum[i-1] = ProcDataSize; }
  pSendNum[ProcNum-1] = RowSize*RowNum -
pSendInd[ProcNum-1];
```
// Розсилка рядків матриці всім процесам програми
```
  MPI_Scatterv(pMatrix , pSendNum, pSendInd, MPI_DOUBLE,
            pProcRows,pSendNum[ProcRank], MPI_DOUBLE, 0,
            MPI_COMM_WORLD);
```
// Звільнення пам’яті
```
  delete [] pSendNum;
  delete [] pSendInd;
}
```

У вищенаведеному фрагменті використано функцію MPI_Scatterv, яка дозволяє посилати кожному процесу різну кількість елементів. Початок розташування елементів блоку, що посилається і-му процесу, задається в масиві зсувів displs, а число посиланих елементів – в масиві sendcounts. Формат функції наступний:

```
int MPI_Scatterv(void* sendbuf, int *sendcounts, int
        *displs, MPI_Datatype sendtype, void* recvbuf,
        int recvcount, MPI_Datatype recvtype, int root,
        MPI_Comm comm),
```

де sendbuf (IN)– адреса початку буфера посилки (використовується тільки в процесі-відправнику root);

sendcounts (IN)– цілочисельний масив (розмір рівний числу процесів в групі), що містить число елементів, які надсилаються кожному процесу;

displs(IN) – цілочисельний масив (розмір рівний числу процесів в групі), і-те значення визначає зсув відносно початку sendbuf для даних, що посилаються процесу і;

sendtype (IN)– тип посиланих елементів;

recvbuf (OUT)– адреса початку буфера прийому;

recvcount (IN)– число отримуваних елементів;

recvtype (IN)– тип отримуваних елементів;

root (IN)– номер процесу-відправника;

comm (IN)– комунікатор(група процесів).

Розглянемо тепер інший підхід до паралельного множення матриці на вектор, заснований на розділенні початкової матриці на неперервні набори (вертикальні смуги) стовпців.

При такому способі розділення даних в якості базової підзадачі може бути вибрана операція множення стовпця матриці A на відповідний елемент вектору b. Для організації обчислень в цьому випадку кожна базова підзадача i, $0 \le i < n$, повинна містити i -й стовпець матриці $A_i$ та *i*-ті елементи $b_i$ та $c_i$ векторів b і c.

Паралельний алгоритм множення матриці на вектор розпочинається з того, що кожна базова задача *i* виконує множення свого стовпця матриці A на елемент $b_i$, у результаті в кожній підзадачі виходить вектор c’(i) проміжних результатів. Далі для отримання елементів результуючого вектору *c* підзадачі повинні обмінятися своїми проміжними даними між собою (елемент j, $0 \le j < n$, часткового результату c’(i) підзадачі i, $0 \le i < n$, має бути переданий підзадачі j). Ця узагальнена передача даних (all - to - all communication або total exchange) є найбільш загальною комунікаційною процедурою і може бути реалізована за допомогою функції MPI_Alltoall бібліотеки MPI. Після виконання передачі даних кожна базова підзадача i, $0 \le i < n$, міститиме n часткових значень c’i (j), $0 \le j < n$,, після складання яких і визначається елемент ci вектору результату з (рисунок 20).

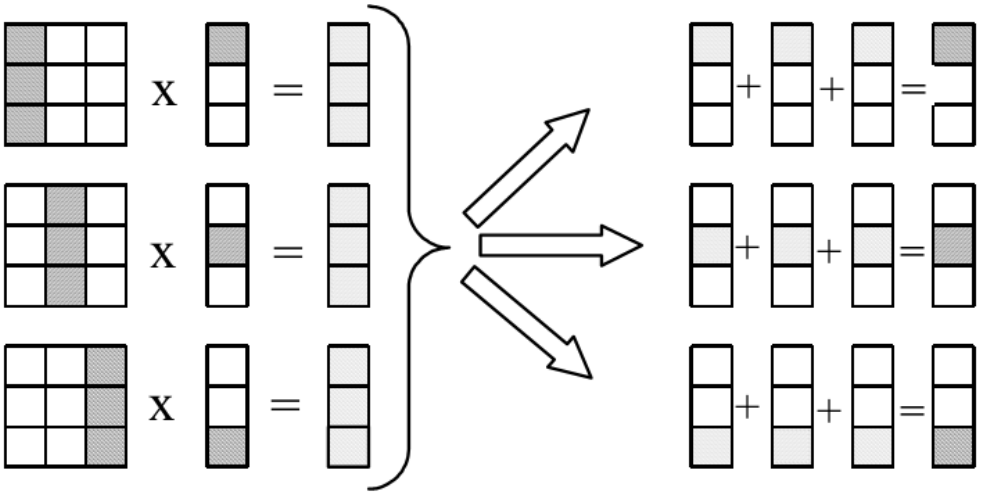


Рисунок 20. Організація обчислень при виконанні паралельного алгоритму множення

## Завдання для виконання роботи

Використовуючи вищенаведений алгоритм, реалізувати задачу матрично-векторного множення, використовуючи вхідні дані відповідно до завдань (n – розмірність квадратної матриці). Варіанти 1-10 виконують розбиття матриці по горизонтальних смужках, 11-20 – по вертикальних.

Завдання 1. $a_{ij} = \sin(i) + \cos(j)$; $b_i = \sin(i) * \cos(i)$; n=100;

Завдання 2. $a_{ij} = \sin(i + j)$; $b_i = \cos(i)$; n=90;

Завдання 3. $a_{ij} = j \cdot \sin(i)$; $b_i = i \cdot \sin(i)$; n=120;

Завдання 4. $a_{ij} = i\log(j)$; $b_i = \sqrt{i}$; n=80;

Завдання 5. $a_{ij} = j\tan(i)$; $b_i = i \cdot \sqrt{i \cdot \cos(i^2)}$; n=110;

Завдання 6. $a_{ij} = i \cdot \sin(j) - tg(i)$; $b_i = \sin(i^2)\sqrt{i}$; n=100;

Завдання 7. $a_{ij} = \dfrac{i \cdot \cos(2j)}{3}$; $b_i = \dfrac{\lg\sqrt{i+5}}{\tan(2*i)+10}$; n=60;

Завдання 8. $a_{ij} = \dfrac{|\sin(i)|}{5 \cdot (j+1)}$; $b_i = \dfrac{1}{\tan(i)+5}$; n=80;

Завдання 9. $a_{ij} = \frac{j^2 + 3j + 2}{\sqrt{i+1}}$; $b_i = \frac{\ln\sqrt{i} \cdot \tan(2*i)}{\sin(i)+3}$; n=50;

Завдання 10. $a_{ij} = \frac{0.3\sqrt{i}}{\cos(j)+5}$; $b_i = \frac{i}{2} \cdot \sqrt{\sin(i)}$; n=100;

Розроблену програму виконати почергово на 1, 2, 4, 6 вузлах. Отримані часові показники проаналізувати і використати для побудови графіка, на основі якого зробити відповідні висновки, які навести у звіті.

Також у звіті до роботи обов’язково навести:

- тему і мету роботи;
- повністю сформульоване завдання відповідно варіанту;
- програму;
- **графік** залежності часу обчислення від кількості вузлів;
- скріншот виконання програми на чотирьох процесорах.

## Контрольні запитання

1. Назвіть основні способи розподілу елементів матриці між процесорами обчислювальної системи.
2. У чому полягає постановка завдання множення матриці на вектор?
3. Чому при розробці паралельних алгоритмів множення матриці на вектор допустимо дублювати вектор-операнд на всі процесори?
4. Розкажіть як реалізований в роботі алгоритм множення.
5. Які інформаційні взаємодії виконуються для алгоритмів при стрічковій схемі розділення даних? У чому відмінність необхідних операцій передачі даним при розділенні матриці по рядкам і стовпцям?
6. Прокоментуйте всі MPI-функції, використані у програмі.

# ЛІТЕРАТУРА


1. Антонов А.С. Параллельное программирование с использованием технологии MPI: Учебное пособие. – М.: Изд-во МГУ, 2004. – 71с.
2. Богачёв К.Ю. Основы параллельного программирования /К.Ю. Богачёв. – М.:Бином. Лаборатория знаний, 2003. – 342с.
3. Воеводин В.В., Воеводин Вл. В. Параллельные вычисления. – СПб.: БХВ-Питербург, 2004. – 608с.
4. Глинський Я. М. С і С++ : навч. посіб. / Глинський Я. М., Анохін В. Ї., Ряжська В. А. – Л. : Деол, СПД Глинський, 2006. – 192с.
5. Клементьев И. П. Устинов В. А. Введение в Облачные вычисления / И. П. Клементьев, В. А. Устинов . – Издательство: УГУ, 2009. – 128с.
6. Лаптев В. С++ : Экспресс-курс / Валерий Лаптев. – С. Пб. : БХВ-Петербург, 2004. – 512 с.
7. Левин М.П. Параллельное программирование с использованием OpenMP/ М.П. Левин. – М.:Бином. Лаборатория знаний, 2003. – 342с.
8. Монахов Д.Н. Облачные Технологии. Теория и практика / Д.Н. Монахов, Н.В. Монахов , Г.Б. Прончев, Д.А.Кузьменков. – МАКС Пресс Москва, МГУ, 2013. – 128с.
9. Пауэрс Л., Снелл М. Microsoft Visual Studio 2008 / Л. Пауэрс, М. Снелл. – С. Пб. : БХВ-Петербург, 2009. – 1191 с.
10. Прата С. Язык программирования С++ : лекции и упражнения / Стивен Прата. – К. : ДиаСофт ЮП, 2005. – 1104 с.
11. Эндрюс Г.Р. Основы многопоточного, параллельного и распределенного программирования.: Пер. с англ. – М.: Издательский дом «Вильямс», 2003. – 512с.
12. Шилдт Г. Полный справочник по С++ / Герберт Шилдт ; [пер. с англ. – 4-е изд.].– М. : Вильямс, 2006. – 800 с.

## Інформаційні ресурси


1. Українська команда розподілених обчислень. [Електронний ресурс]. – Режим доступу: http://distributed.org.ua/
2. Кластерные вычисления. [Електронний ресурс]. – Режим доступу: http://www.intuit.ru/department/calculate/clusterexec/
3. Теория и практика параллельных вычислений. [Електронний ресурс]. – Режим доступу: http://www.intuit.ru/department/calculate/paralltp/
4. Параллельное программирование с использованием технологии MPI. [Електронний ресурс]. – Режим доступу: http://www.intuit.ru/department/se/mpitech/
5. Архитектура параллельных вычислительных систем. [Електронний ресурс]. – Режим доступу: http://www.intuit.ru/department/hardware/paralltech/
6. Параллельное программирование с использованием OpenMP. [Електронний ресурс]. – Режим доступу: http://www.intuit.ru/studies/courses/1112/232/info
7. Параллельное программирование с OpenMP. [Електронний ресурс]. – Режим доступу: http://www.intuit.ru/studies/courses/1111/295/info