

Міністерство Освіти і Науки України
Національний Університет “Львівська Політехніка”



Кафедра ЕОМ

**ОЗНАЙОМЛЕННЯ З ТЕХНОЛОГІЄЮ ПАРАЛЕЛЬНОГО ПРОГРАМУВАННЯ
ЗАСОБАМИ МРІ**

Методичні вказівки

до лабораторної роботи з курсу “Паралельні та розподілені обчислення”
для студентів спеціальності 123. “Комп’ютерна інженерія”

Затверджено
на засіданні кафедри
”Електронні обчислювальні машини”
Протокол № _ від 2024 року

Львів – 2024

Ознайомлення з технологією паралельного програмування засобами MPI: Методичні вказівки до лабораторної роботи з курсу “Паралельні та розподілені обчислення” для студентів спеціальності 123 - “Комп’ютерна інженерія”/ Укладачі: Є. Ваврук, О. Лашко – Львів: Національний університет “Львівська політехніка”, 2024, 16 с.

Укладачі: Є. Ваврук, к.т.н., доцент.
О. Лашко, ст..викл.

Відповідальний за випуск: Дунець Р. Б., професор, завідувач кафедри

Рецензенти: Попович Р.Б., к. т. н, доцент
Юрчак І.Ю., к. т. н, доцент

МЕТА РОБОТИ

Вивчити основні поняття та визначення MPI. Набути навиків розробки паралельних програм з використанням MPI.

ТЕОРЕТИЧНІ ВІДОМОСТІ:

У обчислювальних системах з розподіленою пам'яттю процесори працюють незалежно один від одного. Для організації паралельних обчислень в таких умовах необхідно мати можливість розподіляти обчислювальне навантаження і організувати інформаційну взаємодію (передачу даних) між процесорами. Одним з способів взаємодії між паралельними процесами є передача повідомлень між ними, що відображено в самій назві технології MPI (message passing interface) – інтерфейс передачі повідомлень.

Для розподілу обчислень між процесорами необхідно проаналізувати алгоритм розв'язку задачі, виділити інформаційно незалежні фрагменти обчислень, виконати їх програмну реалізацію і розмістити отримані частини програми на різних процесорах. В MPI використовуються простіший підхід - для виконання завдання розробляється одна програма, яка запускається одночасно на виконання на всіх наявних процесорах. При цьому для того, щоб уникнути ідентичності обчислень на різних процесорах, можна, по-перше, підставляти різні дані для програми на різних процесорах, а по-друге, використовувати наявні в MPI засоби для ідентифікації процесора, на якому виконується програма (тим самим надається можливість організувати відмінності в обчисленнях залежно від використовуваного програмою процесора).

Інтерфейс MPI підтримує реалізацію програм для MIMD систем (Multiple Instructions Multiple Data), проте відлагодження таких програм є не тривіальною задачею. Тому на практиці для написання програм в більшості випадків застосовується SPMD (single program multiple processes) модель паралельного програмування - "одна програма безліч процесів".

1. MPI: основні поняття і визначення

MPI - це стандарт, якому повинні задовольняти засоби організації передачі повідомлень. Крім того MPI - це програмні засоби, які забезпечують можливість передачі повідомлень і при цьому відповідають всім вимогам стандарту MPI. Згідно стандарту, ці програмні засоби повинні бути організовані у вигляді бібліотек програмних функцій (бібліотеки MPI) і повинні бути доступні для найширше використовуваних алгоритмічних мов C і Fortran. Подібну "подвійність" MPI слід враховувати при використанні термінології. Як правило, аббревіатура MPI застосовується при згадці стандарту, а поєднання "бібліотека MPI" указує на ту або іншу програмну реалізацію стандарту. Оскільки достатньо часто скорочено позначення MPI використовується і для бібліотек MPI, тому для правильної інтерпретації терміну, слід враховувати контекст.

1.1. Поняття паралельної програми. Під паралельною програмою в MPI розуміється множина одночасно виконуваних процесів. Процеси можуть виконуватися як на різних процесорах, так і на одному процесорі можуть виконуватися і декілька процесів (в цьому випадку їх виконання здійснюється в режимі розділення часу). У граничному випадку для виконання паралельної програми може використовуватися один процесор - як правило, такий спосіб застосовується для початкової перевірки правильності паралельної програми.

Кожен процес паралельної програми породжується на основі копії одного і того ж програмного коду (модель SPMP). Даний програмний код, представлений у вигляді виконуваної програми, повинен бути доступний у момент запуску паралельної програми на всіх використовуваних процесорах. Початковий програмний код для виконуваної програми розробляється на алгоритмічних мовах C або Fortran із застосуванням тієї або іншої реалізації бібліотеки MPI.

Кількість процесів і використовуваних процесорів визначається у момент запуску паралельної програми засобами середовища виконання MPI-програм і в ході обчислень не може змінюватися без застосування спеціальних засобів динамічного породження процесів і управління ними, згідно з стандартом MPI версії 2.0. Всі процеси програми послідовно перенумеровані від 0 до $p-1$, де p є загальна кількість процесів. Номер процесу іменується рангом процесу.

1.2. Операції передачі даних. Основу MPI складають операції передачі повідомлень. Серед передбачених у складі MPI функцій розрізняються парні (point-to-point) операції між двома процесами і колективні (collective) комунікаційні дії для одночасної взаємодії декількох процесів.

Для виконання парних операцій можуть використовуватися різні режими передачі (синхронний, блокуючий і ін.). Повний розгляд можливих режимів передачі розглядається нижче.

1.3. Поняття комунікаторів. Процеси паралельної програми об'єднуються в групи. Іншим важливим поняттям MPI, що описує набір процесів, є поняття комунікатора. Під комунікатором в MPI розуміється спеціально створений службовий об'єкт, який об'єднує групу процесів і ряд додаткових параметрів (контекст), використовуваних при виконанні операцій передачі даних.

Парні операції передачі даних можуть бути виконані між будь-якими процесами одного і того ж комунікатора, а в колективних операціях беруть участь всі процеси комунікатора. Як результат, вказання використовуваного комунікатора є обов'язковим для операцій передачі даних в MPI.

Логічна топологія ліній зв'язку між процесами має структуру повного графа (незалежно від наявності реальних фізичних каналів зв'язку між процесорами).

У MPI є можливість представлення множини процесів у вигляді ґраток довільної розмірності. Крім того, в MPI є засоби і для формування логічних (віртуальних) топологій будь-якого необхідного типу.

Під час обчислень можуть створюватися нові і видалятися існуючі групи процесів і комунікаторів. Один і той же процес може належати різним групам і комунікаторам. Всі наявні в паралельній програмі процеси входять до складу конструйованого за замовчуванням комунікатора з ідентифікатором MPI_COMM_WORLD.

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

1.4. Типи даних. При виконанні операцій передачі повідомлень для вказівки передаваних або отримуваних даних у функціях MPI необхідно вказувати тип даних, що пересилаються. MPI містить великий набір базових типів даних, багато в чому співпадаючих з типами даних в алгоритмічних мовах C і Fortran. Крім того, в MPI є можливості створення нових похідних типів даних для точнішого і коротшого опису вмісту повідомлень, що пересилаються.

2. Введення в розробку паралельних програм з використанням MPI

Мінімально необхідний набір функцій MPI, достатній для розробки порівняно простих паралельних програм.

2.1. Ініціалізація і завершення MPI-програм. Першою функцією MPI, що викликається, повинна бути функція

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

argc - вказівник на кількість параметрів командного рядка;

argv - параметри командного рядка.

яка використовується для ініціалізації середовища виконання MPI-програми. Параметрами функції є кількість аргументів в командному рядку і адреса вказівника на масив параметрів командного рядка.

Останньою функцією MPI, що викликається, обов'язково повинна бути функція:

int MPI_Finalize(void).

Структура паралельної програми з використанням MPI повинна мати такий вигляд:

```
#include "mpi.h"
int main(int argc, char *argv[])
{
    <програмний код без використання функцій MPI>
    MPI_Init(&argc, &argv);
    <програмний код з використанням функцій MPI>
    MPI_Finalize();
    <програмний код без використання функцій MPI>
    return 0;
}
```

Варто зазначити:

1. Файл *mpi.h* містить визначення іменованих констант, прототипів функцій і типів даних бібліотеки MPI.
2. Функції *MPI_Init* і *MPI_Finalize* є обов'язковими і повинні бути виконані (і лише один раз) кожним процесом паралельної програми.
3. Перед викликом *MPI_Init* може бути використана функція *MPI_Initialized* для визначення того, чи був раніше виконаний виклик *MPI_Init*, а після виклику *MPI_Finalize* - *MPI_Finalized* аналогічного призначення.

Розглянуті приклади функцій дають представлення синтаксису іменування функцій в MPI. Імені функції передують префікс MPI; далі одне або декілька слів назви; перше слово в імені функції починається із заголовного символу; слова розділяються знаком підкреслення. Назви функцій MPI, як правило, пояснюють призначення виконуваних функцією дій.

2.2. Визначення кількості і рангу процесів. Визначення кількості процесів у виконуваний паралельній програмі здійснюється за допомогою функції:

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

comm - комунікатор, розмір якого визначається;
size - визначена кількість процесів в комунікаторі.

Для визначення рангу процесу використовується функція:

*int MPI_Comm_Rank(MPI_Comm comm, int *rank), де*

comm - комунікатор, в якому визначається ранг процесу;
rank - ранг процесу в комунікаторі.

Як правило, виклик функцій *Mpi_comm_size* і *Mpi_comm_rank* виконується відразу після *Mpi_init* для отримання загальної кількості процесів і рангу поточного процесу:

```

#include "mpi.h"
int main(int argc, char *argv[])
{
    int ProcNum, ProcRank;
    <програмний код без використання функцій MPI>
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
    MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
    <програмний код з використанням функцій MPI>
    MPI_Finalize();
    <програмний код без використання функцій MPI>
    return 0;
}

```

Варто зазначити:

1. Комунікатор `MPI_COMM_WORLD`, як наголошувалося раніше, створюється за замовчуванням і представляє всі процеси виконуваної паралельної програми.
2. Ранг, що отримується за допомогою функції `MPI_Comm_rank`, є рангом процесу, що виконав виклик цієї функції, тобто змінна `ProcRank` прийме різні значення у різних процесів.

2.3. Передача повідомлень. Для передачі повідомлення процес-відправник повинен виконати функцію:

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

buf - адреса буфера пам'яті, в якому розташовуються дані повідомлення, що відправляється;

count - кількість елементів даних в повідомленні;

type - тип елементів даних повідомлення, що пересилається;

dest - ранг процесу, якому відправляється повідомлення;

tag - значення-тег, що використовується для ідентифікації повідомлення;

comm - комунікатор, в рамках якого виконується передача даних.

Для вказання типу даних, що пересилаються, в MPI є ряд базових типів, повний список яких наведений в таблиці

Базові типи даних MPI для алгоритмічної мови C

Тип даних MPI	Тип даних C
MPI_BYTE	
MPI_CHAR	signed char
MPI_DOUBLE	Double
MPI_FLOAT	Float
MPI_INT	Int
MPI_LONG	Long
MPI_LONG_DOUBLE	long double
MPI_PACKED	
MPI_SHORT	Short
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED	unsigned int

MPI_UNSIGNED_LONG	unsigned long
MPI_UNSIGNED_SHORT	unsigned short

Варто зазначити:

1. Повідомлення, що відправляється, визначається шляхом задання блоку пам'яті (буфера), в якому це повідомлення розташовується. Використовувана для цього тріада (*buf, count, type*) входить до складу параметрів практично всіх функцій передачі даних.

2. Процеси, між якими виконується передача даних обов'язково повинні належати комунікатору, що вказується у функції *MPI_Send*.

3. Параметр *tag* використовується тільки при необхідності розрізнення передаваних повідомлень, інакше, як значення параметра, може бути використане довільне додатнє ціле число.

Відразу після завершення виконання функції *MPI_Send* процес-відправник може почати повторно використовувати буфер пам'яті, в якому розташовувалося повідомлення, що відправлялося. Також необхідно пам'ятати, що у момент завершення функції *MPI_Send* стан самого повідомлення, що пересилається, може бути абсолютно різним: повідомлення може розташовуватися в процесі-відправнику, може знаходитися в стані передачі, може зберігатися в процесі-одержувачі або ж може бути прийнято процесом-одержувачем за допомогою функції *MPI_Recv*. Тим самим, завершення функції *MPI_Send* лише означає, що операція передачі почала виконуватися і пересилка повідомлення рано чи пізно буде виконана.

2.4. Прийом повідомлень. Для прийому повідомлення процес-одержувач повинен виконати функцію:

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

buf, count, type - буфер пам'яті для прийому повідомлення, призначення кожного окремого параметра відповідає опису в *MPI_Send*.

source - ранг процесу, від якого повинен бути виконаний прийом повідомлення.

tag - тег повідомлення, яке повинне бути прийняте для процесу.

comm - комунікатор, в рамках якого виконується передача даних.

status - вказівник на структуру даних з інформацією про результат виконання операції прийому даних.

Варто зазначити:

1. Буфер пам'яті повинен бути достатнім для прийому повідомлення. При браку обсягу пам'яті частина повідомлення буде втрачена і в кодї завершення функції буде зафіксована помилка переповнення. З іншого боку, повідомлення, що приймається, може бути коротшим від розміру приймального буфера. У такому разі зміняться тільки області буфера, використані прийнятим повідомленням.

2. Типи елементів повідомлення, що передаються і приймаються, повинні співпадати.

3. При необхідності прийому повідомлення від будь-якого процесу-відправника для параметра *source* може бути вказане значення *MPI_ANY_SOURCE* (на відміну від функції передачі *MPI_Send*, яка посилає повідомлення строго певного адресата).

4. При необхідності прийому повідомлення з будь-яким тегом для параметра *tag* може бути вказане значення *MPI_ANY_TAG* (знову-таки, при використанні функції *MPI_Send* повинне бути вказане конкретне значення тега).

5. На відміну від параметрів "процес-одержувач" і "тег", параметр "комунікатор" не має значення, що означає "будь-який комунікатор".

6. Параметр `status` дозволяє визначити ряд характеристик прийнятого повідомлення.
7. `Status.MPI_SOURCE` - ранг процесу - відправника прийнятого повідомлення.
8. `Status.MPI_TAG` - тег прийнятого повідомлення.

Значення змінної `status` дозволяє визначити кількість елементів даних в прийнятому повідомленні за допомогою функції:

*int MPI_Get_count(MPI_Status *status, MPI_Datatype type, int *count), де*

status - статус операції `MPI_Recv`;

type - тип прийнятих даних;

count - кількість елементів даних в повідомленні.

Виклик функції `Mpi_Recv` не зобов'язаний бути узгодженим з часом виклику відповідної функції передачі повідомлення `MPI_Send` - прийом повідомлення може бути ініційований до моменту, в момент чи після моменту початку відправки повідомлення.

Після закінчення виконання функції `MPI_Recv` в заданому буфері пам'яті буде розташоване прийняте повідомлення. Принциповий момент полягає в тому, що функція `MPI_Recv` є блокуючою для процесу-одержувача, тобто його виконання припиняється до завершення роботи функції. Таким чином, якщо, по якихось причинах очікуване для прийому повідомлення буде відсутнє, виконання паралельної програми буде заблоковано.

3. Інсталяція та зв'язування бібліотеки MPI

Для виконання лабораторних робіт можна використовувати **Microsoft MPI**: <https://learn.microsoft.com/uk-ua/message-passing-interface/microsoft-mpi>.

Останню версію **Microsoft MPI v10.1.3** (станом на 12.02.2024) можна взяти тут - <https://www.microsoft.com/en-us/download/details.aspx?id=105289>.

Необхідно встановити програму для запуску MPI програм і бібліотеку:

- `msmpisdk.msi` - software development kit (SDK) — бібліотека;
- `msmpisetup.exe` — програма для запуску MPI програм.

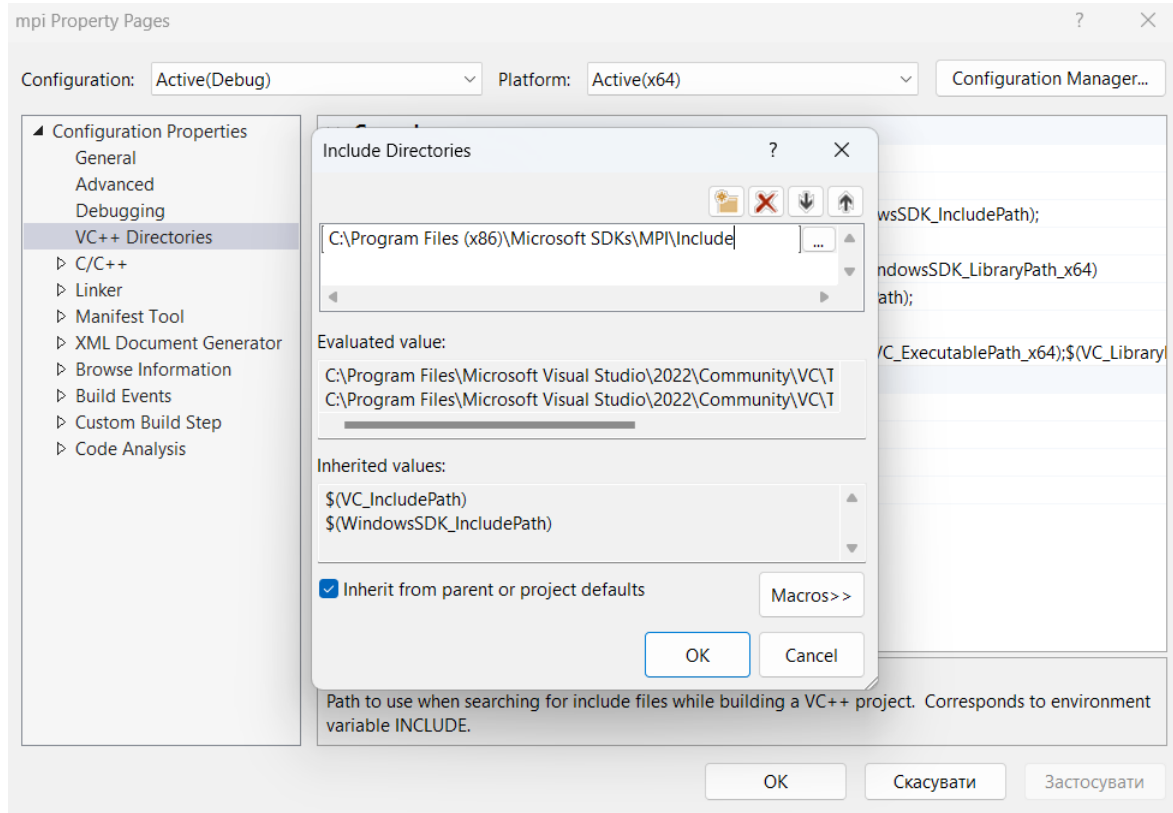
Довідкову інформацію (**MPI Reference**) можна глянути тут - <https://learn.microsoft.com/uk-ua/message-passing-interface/mpi-reference>

Як налаштувати середовище Microsoft Visual Studio і запустити MPI програму (How to compile and run a simple MS-MPI program) можна глянути тут - <https://learn.microsoft.com/en-us/archive/blogs/windowshpc/how-to-compile-and-run-a-simple-ms-mpi-program>

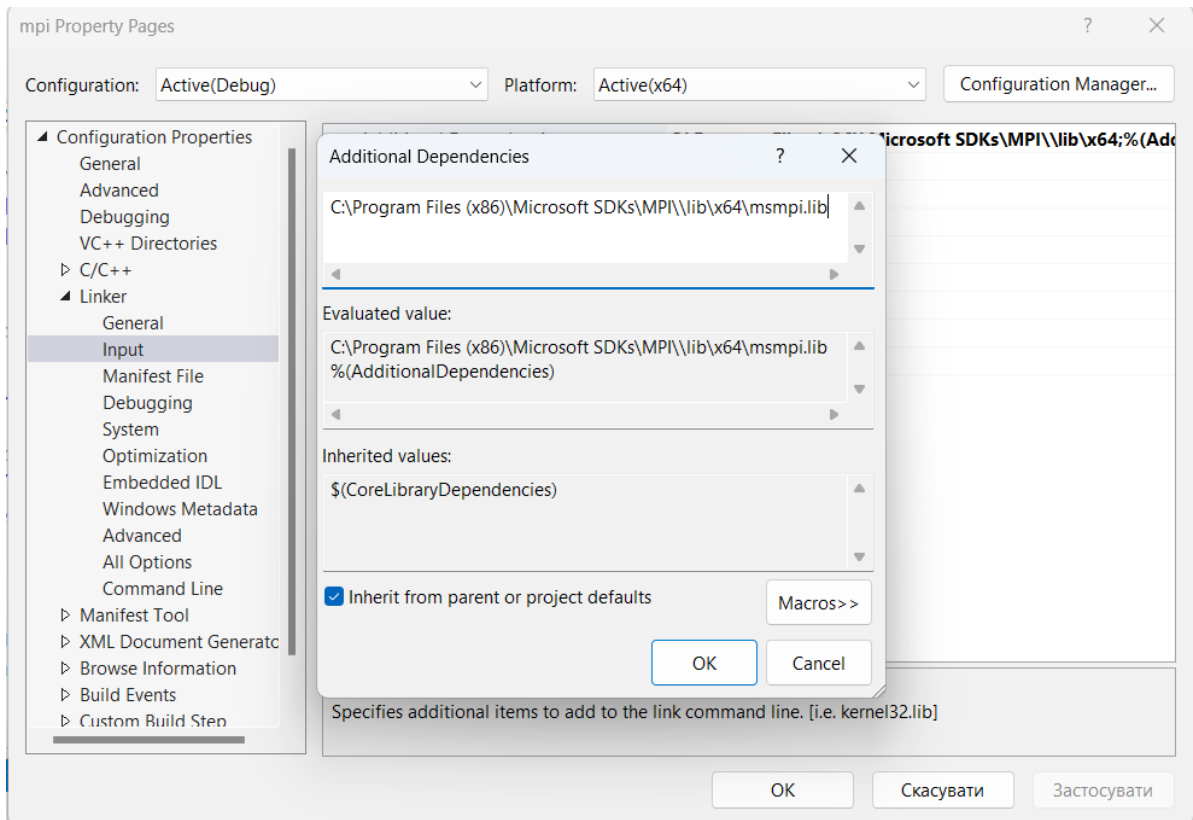
Послідовність дій наступна:

1. Створюємо порожній проект у середовищі Microsoft Visual Studio.

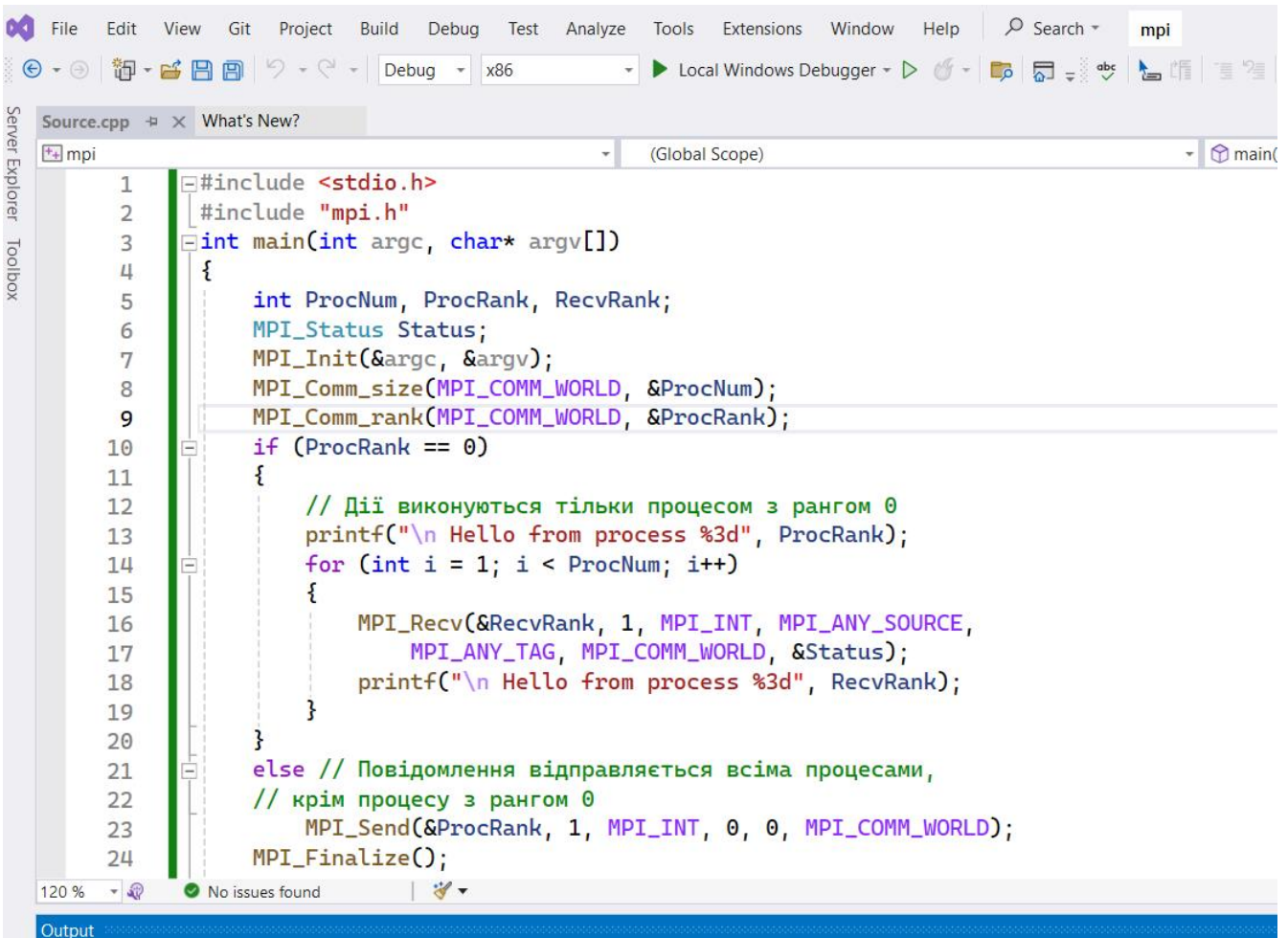
2. Додаємо шлях до заголовних файлів.



3. Додаємо шлях до .dll бібліотеки



4. Створюємо .cpp файл і копіюємо туди тестову програму.

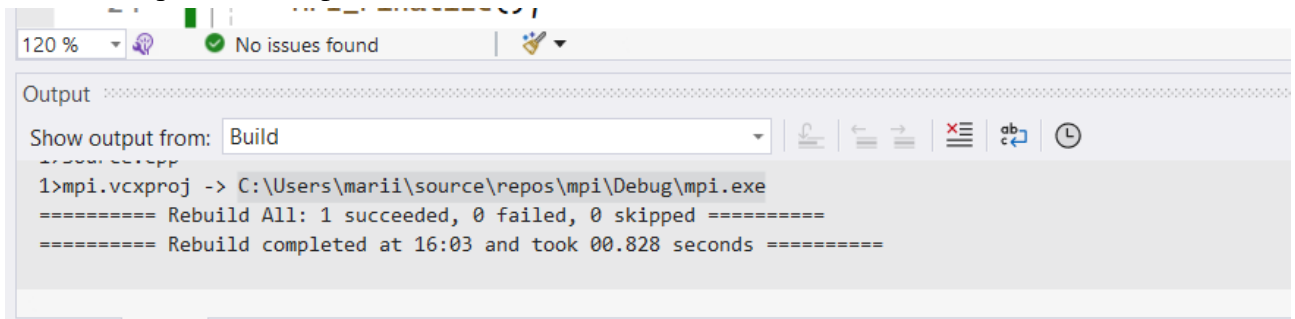


The screenshot shows the Visual Studio IDE with the 'Source.cpp' file open. The code is an MPI program that prints 'Hello from process' for each rank. The code is as follows:

```
1 #include <stdio.h>
2 #include "mpi.h"
3 int main(int argc, char* argv[])
4 {
5     int ProcNum, ProcRank, RecvRank;
6     MPI_Status Status;
7     MPI_Init(&argc, &argv);
8     MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
9     MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
10    if (ProcRank == 0)
11    {
12        // Дії виконуються тільки процесом з рангом 0
13        printf("\n Hello from process %3d", ProcRank);
14        for (int i = 1; i < ProcNum; i++)
15        {
16            MPI_Recv(&RecvRank, 1, MPI_INT, MPI_ANY_SOURCE,
17                    MPI_ANY_TAG, MPI_COMM_WORLD, &Status);
18            printf("\n Hello from process %3d", RecvRank);
19        }
20    }
21    else // Повідомлення відправляється всіма процесами,
22          // крім процесу з рангом 0
23        MPI_Send(&ProcRank, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
24    MPI_Finalize();
25 }
```

The interface includes a menu bar (File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help), a toolbar with icons for file operations and debugging, and a status bar at the bottom showing '120 %' zoom and 'No issues found'.

5. Створюємо .exe файл.

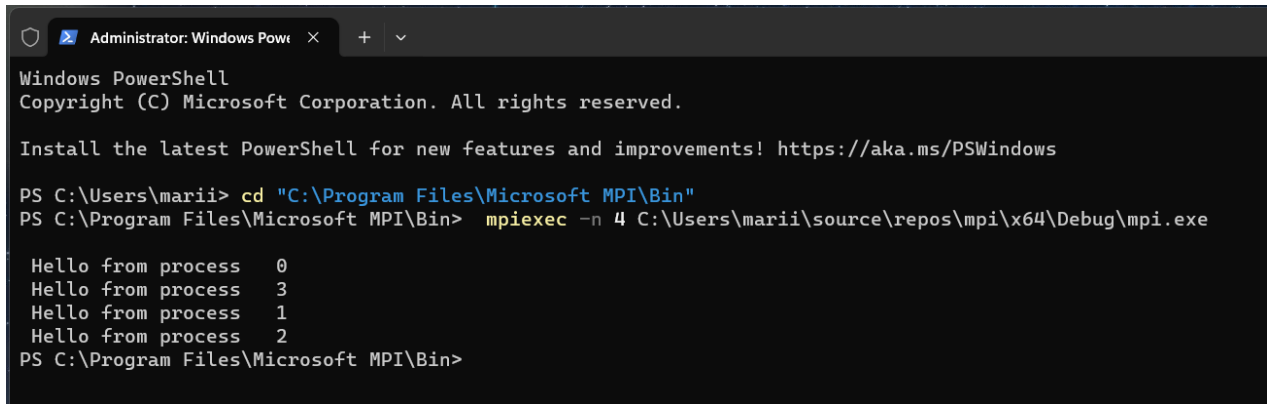


The screenshot shows the 'Output' window in Visual Studio. The 'Show output from:' dropdown is set to 'Build'. The output text is as follows:

```
1>mpi.vcxproj -> C:\Users\marii\source\repos\mpi\Debug\mpi.exe
===== Rebuild All: 1 succeeded, 0 failed, 0 skipped =====
===== Rebuild completed at 16:03 and took 00.828 seconds =====
```

The output window also features a toolbar with icons for clearing, previous, next, and other actions.

6. Далі запускаємо MPI програму на виконання з командного рядка.



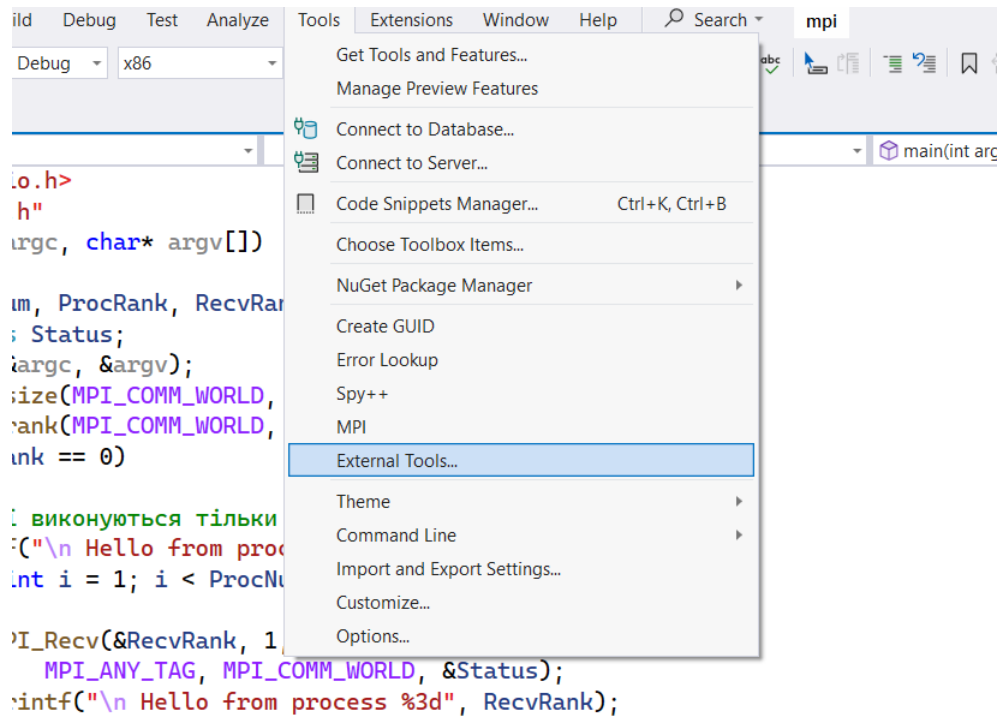
```
Administrator: Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

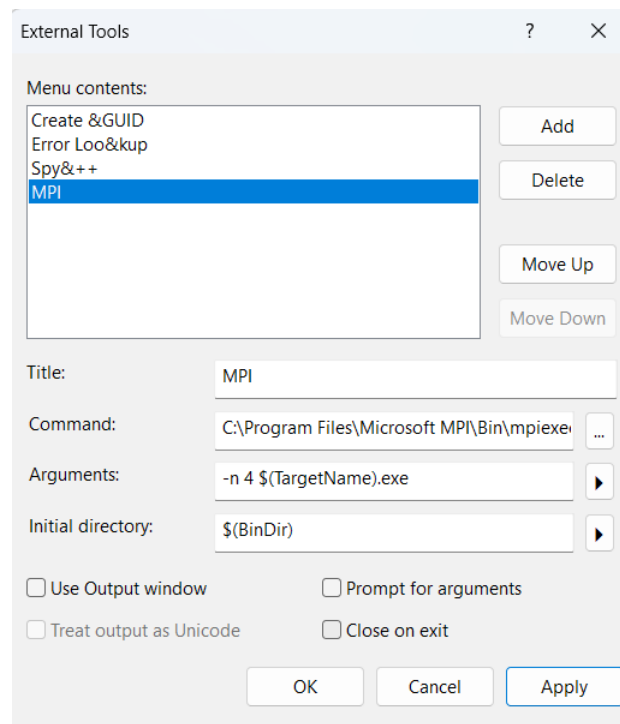
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\marii> cd "C:\Program Files\Microsoft MPI\Bin"
PS C:\Program Files\Microsoft MPI\Bin> mpiexec -n 4 C:\Users\marii\source\repos\mpi\x64\Debug\mpi.exe

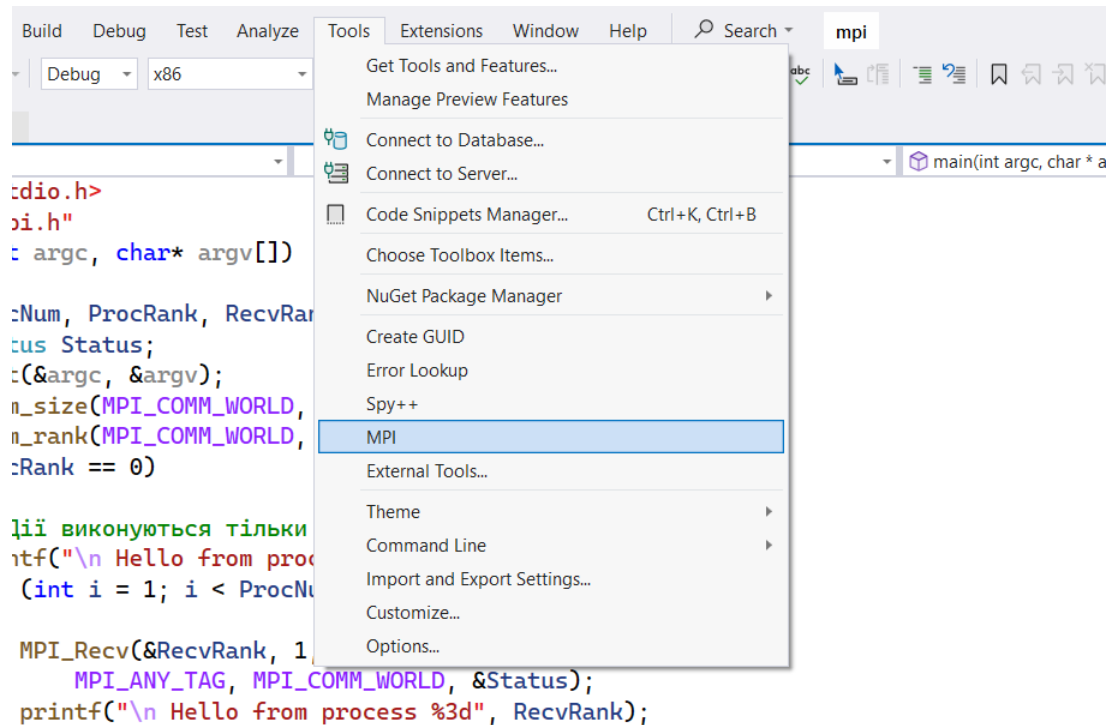
Hello from process 0
Hello from process 3
Hello from process 1
Hello from process 2
PS C:\Program Files\Microsoft MPI\Bin>
```

7. Для полегшення запуску MPI програм можна налаштувати зовнішні інструменти у Visual Studio.





Тепер для запуску програми достатньо вибрати Tools\MPI.



КОНТРОЛЬНІ ЗАПИТАННЯ

1. Яка модель організації обчислень застосовується для написання програм з використанням MPI?
2. Пояснити поняття паралельної програми в MPI.
3. Навести структуру паралельної програми з використанням MPI?
4. Як визначити кількість процесів у виконуваний паралельній програмі?
5. Які типи обміну повідомлень між процесами ви знаєте?

ПОРЯДОК ВИКОНАННЯ РОБОТИ

1. Встановити та налаштувати бібліотеку MPI для розробки MPI програм у середовищі програмування Microsoft Visual Studio 2022.
2. Написати програму обміну повідомлень між процесами з використанням MPI. Кожен процес має визначати свій ранг та пересилати його значення головному процесу. Головний процес повинен виводити отримане значення на екран.
3. Виконати розроблену програму для різної кількості процесів.
4. Зробити висновок про порядок прийому повідомлень головним процесом та звернути увагу на його зміну від виклику до виклику розробленої програми.

ЗМІСТ ЗВІТУ ДО ЛАБОРАТОРНОЇ РОБОТИ

1. Структура паралельної програми з використанням MPI.
2. Завдання (кількість процесів) згідно варіанту.
3. Текст програми на C з використанням бібліотеки MPI.
4. Вікно консолі з результатами виконання програми (результат друку рангів процесів).
5. Висновок.

ЗАВДАННЯ

Варіант №	Кількість процесів 1	Кількість процесів 2
1	4	26
2	8	22
3	3	27
4	7	23
5	5	25
6	2	28
7	6	24
8	12	18
9	10	22
10	30	10
11	15	5
12	22	8
13	11	19
14	13	17

15	16	14
16	9	21
17	18	12
18	23	7
19	25	5
20	27	3
21	14	16
22	19	11
23	6	24
24	8	22
25	11	19
26	14	18
27	16	22
28	18	8
29	8	12
30	6	22

ЛІТЕРАТУРА

1. А. А. Букатов, В. Н. Дацюк, А. И. Жегуло. Программирование многопроцессорных вычислительных систем. Ростов-на-Дону. Издательство ООО «ЦВВР», 2003, 208 с.
2. Антонов А.С. Параллельное программирование с использованием технологии MPI: Учебное пособие. – М.: Издательство МГУ, 2004. – 71 с.
3. Є. Ваврук, О. Лашко. Організація паралельних обчислень: Навчальний посібник. – Львів: Національний університет “Львівська політехніка”, 2007. – 70 с.
4. www.parallel.ru
5. www.mcs.anl.gov/mpi/mpich

ПРИКЛАД ВИКОНАННЯ

Завдання:

Написати програму обміну повідомленнями між процесами з використанням MPI та запустити її для 4 процесів.

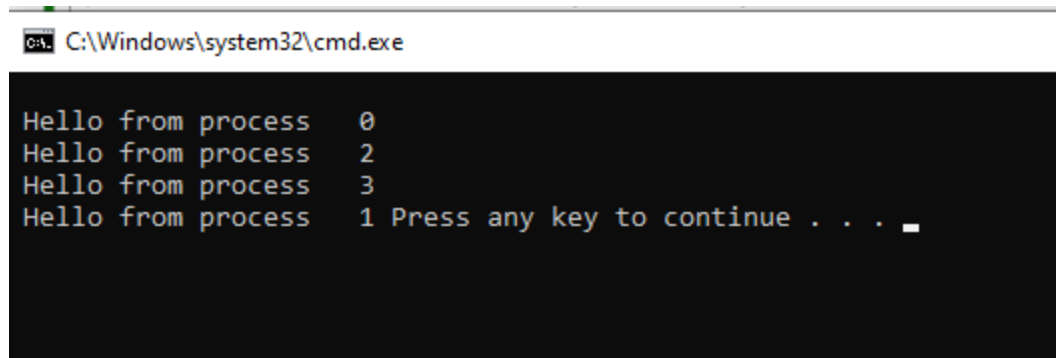
Текст програми:

```
#include <stdio.h>
#include "mpi.h"
int main(int argc, char* argv[])
{
    int ProcNum, ProcRank, RecvRank;
    MPI_Status Status;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
    MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
    if (ProcRank == 0)
    {
        // Дії виконуються тільки процесом з рангом 0
        printf("\n Hello from process %3d ", ProcRank);
        for (int i = 1; i < ProcNum; i++)
```

```

    {
        MPI_Recv(&RecvRank, 1, MPI_INT, MPI_ANY_SOURCE,
                MPI_ANY_TAG, MPI_COMM_WORLD, &Status);
        printf("\n Hello from process %3d ", RecvRank);
    }
}
else // Повідомлення відправляється всіма процесами,
     // крім процесу з рангом 0
    MPI_Send(&ProcRank, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
MPI_Finalize();
return 0;
}

```



```

C:\Windows\system32\cmd.exe
Hello from process 0
Hello from process 2
Hello from process 3
Hello from process 1 Press any key to continue . . .

```

Висновок

Порядок прийому повідомлень заздалегідь не визначений і залежить від умов виконання паралельної програми (більш того, цей порядок може змінюватися від запуску до запуску). Так, можливий варіант результатів друку процесу 0 для паралельної програми з чотирьох процесів наведено на рисунку вище.

НАВЧАЛЬНЕ ВИДАННЯ

ОЗНАЙОМЛЕННЯ З ТЕХНОЛОГІЄЮ ПАРАЛЕЛЬНОГО ПРОГРАМУВАННЯ ЗАСОБАМИ MPI

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторної роботи з дисципліни “Паралельні та розподілені обчислення”
для студентів для студентів базового напрямку 123- “Комп’ютерна інженерія”

Укладачі:

Баврук Євгеній Ярославович
Лашко Оксана Любомирівна

Редактор

Комп’ютерне верстання

Здано у видавництво . Підписано до друку
Формат 70х100/16. Папір офсетний. Друк на різнографі
Умовн. друк. арк. Обл.-вид. арк..
Тираж прим. Зам..

Видавництво Національного університету “Львівська політехніка”
Реєстраційне свідоцтво ДК №751 від 27.12.2001 р.

Поліграфічний центр Видавництва
Національного університету “Львівська політехніка”

Вул.. Ф. Колесси, 2. Львів, 79000