BÀI THỰC HÀNH SỐ 3

HP: HPC VỚI MPI

NỘI DUNG:

- Cài đặt và thiết lập môi trường VS C/C++.NET với MPI
- Thực hiện các bài toán với thuật toán song song trong môi trường bộ nhớ phân tán (MPI)

YÊU CẦU: Viết báo cáo nộp cuối buổi với nội dung

- Liệt kê các công việc đã thực hiện (gạch đầu dòng)
- Kết quả chạy các chương trình ví dụ và giải thích kết quả
- So sánh thời gian xử lý giữa chạy chương trình tuần tự và song song (với mọi ví dụ và bài tập)

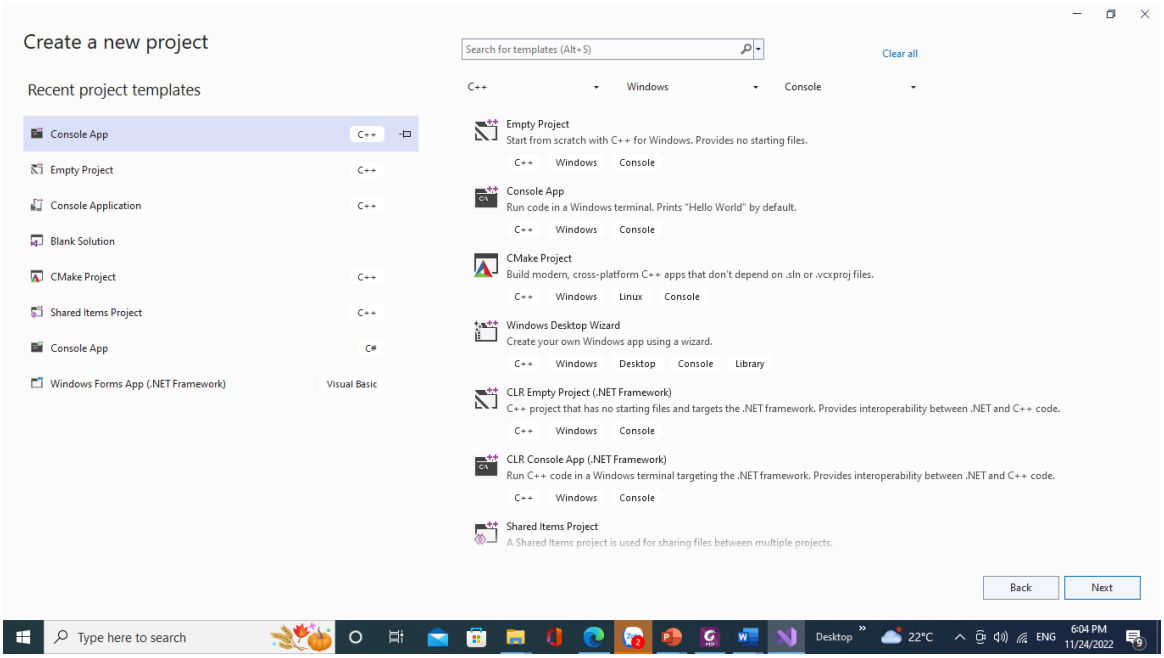
PHẦN 1: CÀI ĐẶT THIẾT LẬP MÔI TRƯỜNG VS C/C++.NET VỚI MPI

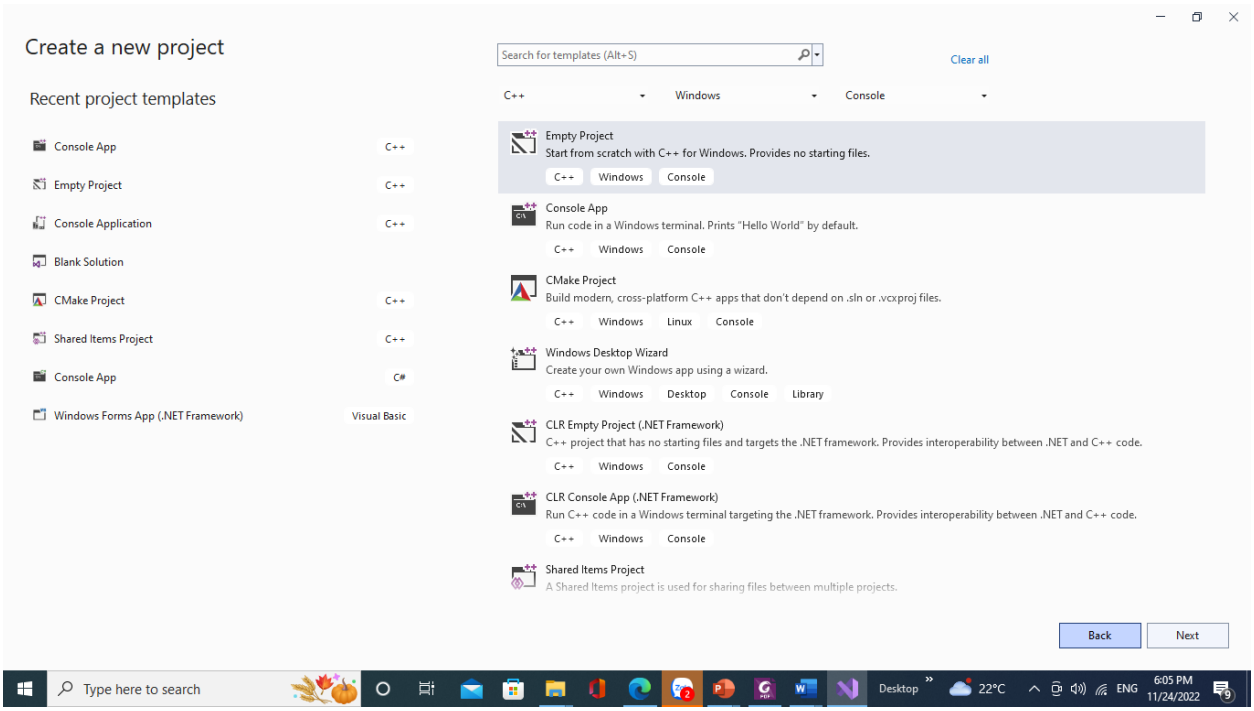
Bước 1: Cài đặt thư viện MPI cho VS 2019 (trên máy đã cài sẵn VS 2019)

- Bộ cài đặt gồm 2 file: msmpisetup.exe, msmpisdk.msi
- Tiến hành cài đặt 2 tệp

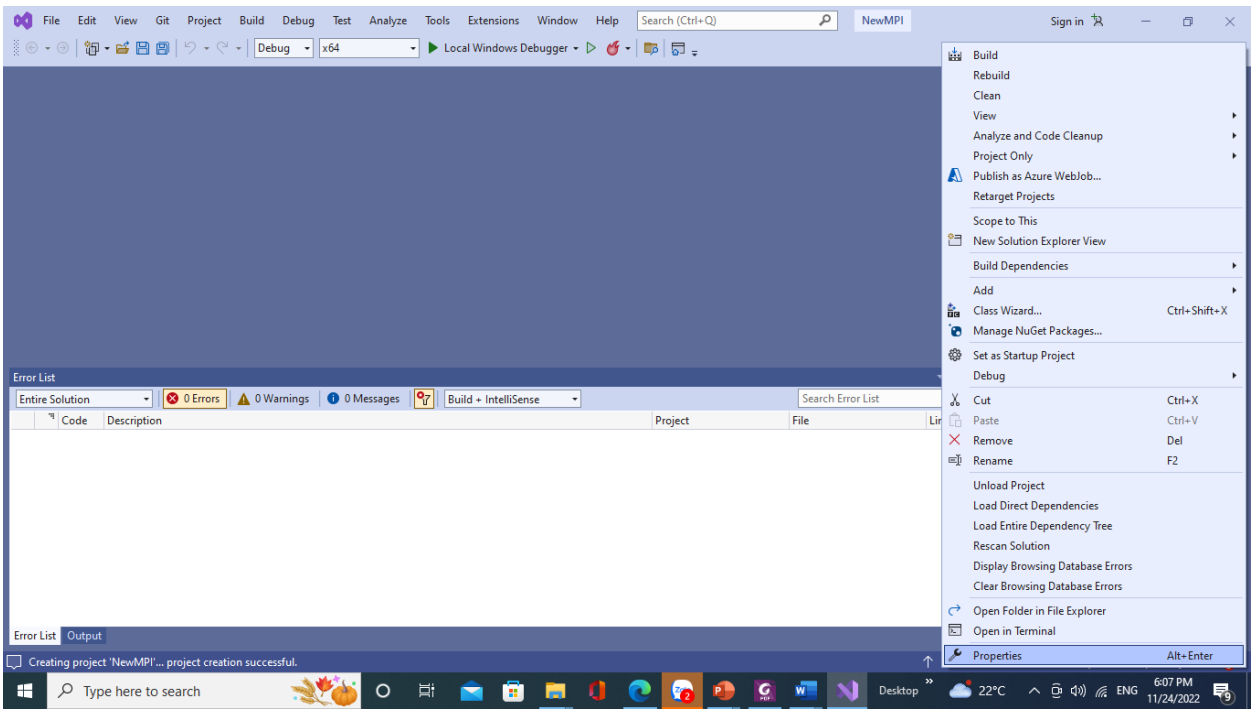
Bước 2: Cấu hình MPI trong VS 2019

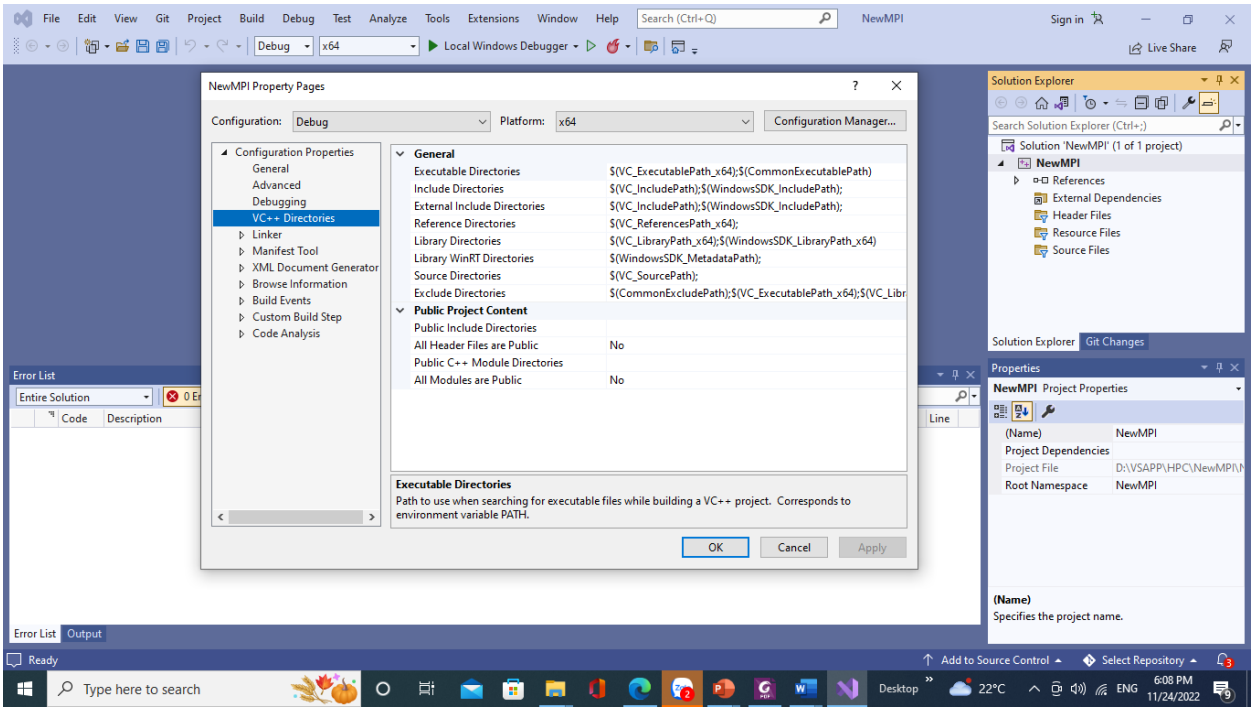
Bước 21: Trong môi trường VS 2019, mở dự án mới





Bước 22: Trong dự án mới, mở cửa sổ properties của dự án



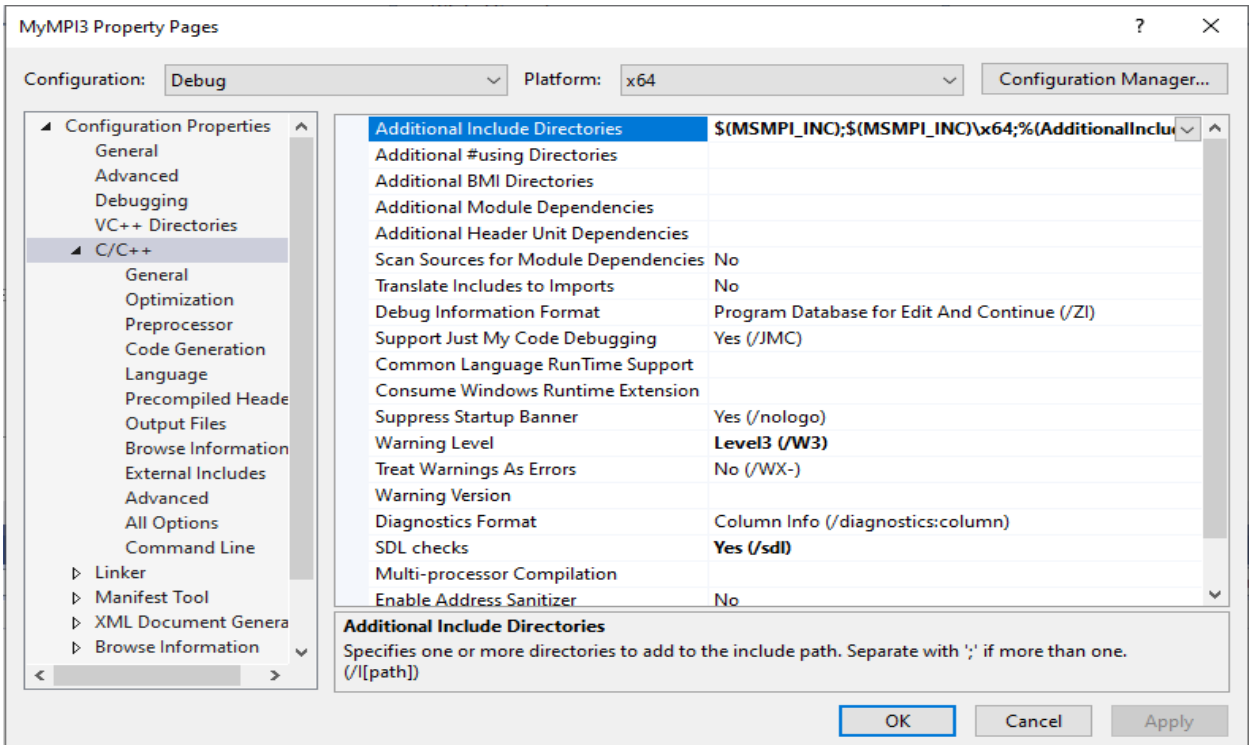


Chọn mục VC++ Directories. Thêm các đường dẫn như sau:

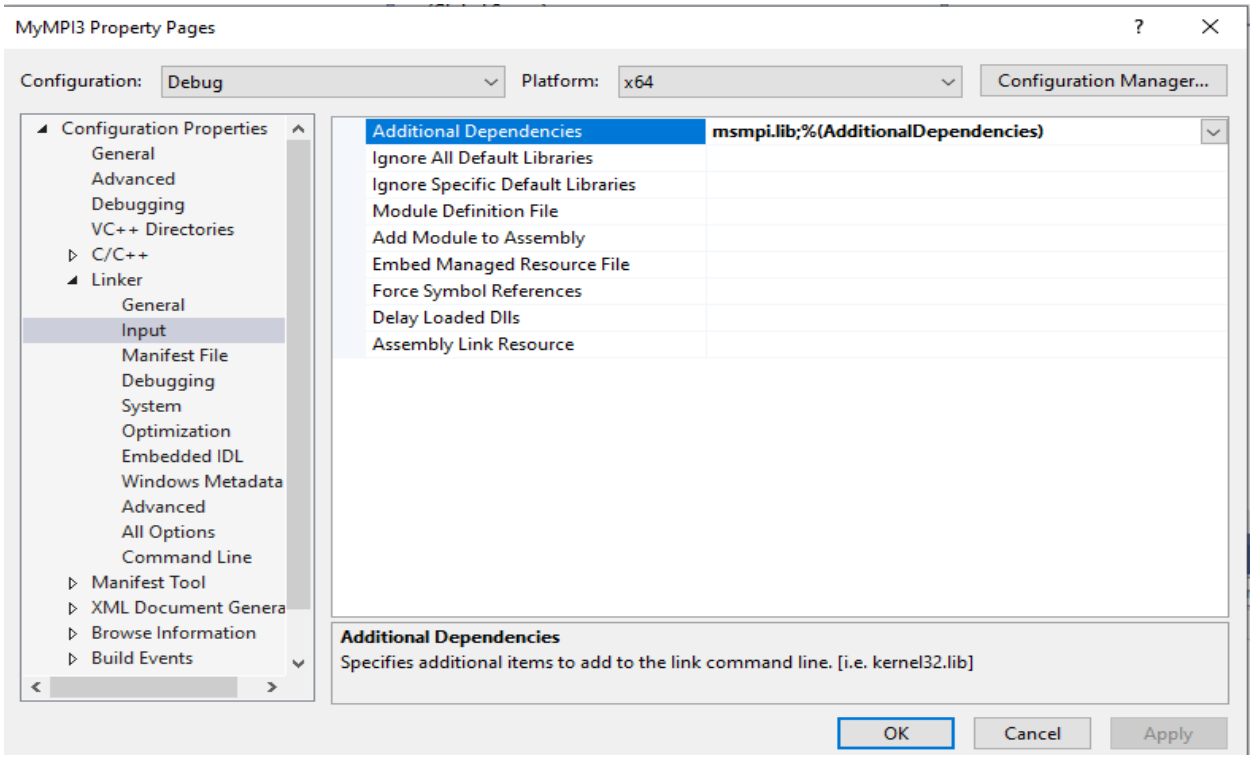
- Thêm vào **Additional Include Directories** (hình H1)

  $(MSMPI_INC);$(MSMPI_INC)\x64
- Thêm vào **Additional Dependencie**s của Linker/Input thư viện msmpi.lib (hình H2), lưu ý thêm dấu ‘;’ vào sau msmpi.lib để phân tách với chuỗi khác.
- Thêm vào mục **Additional Library Directories** của Linker/General chuỗi (hình H3):

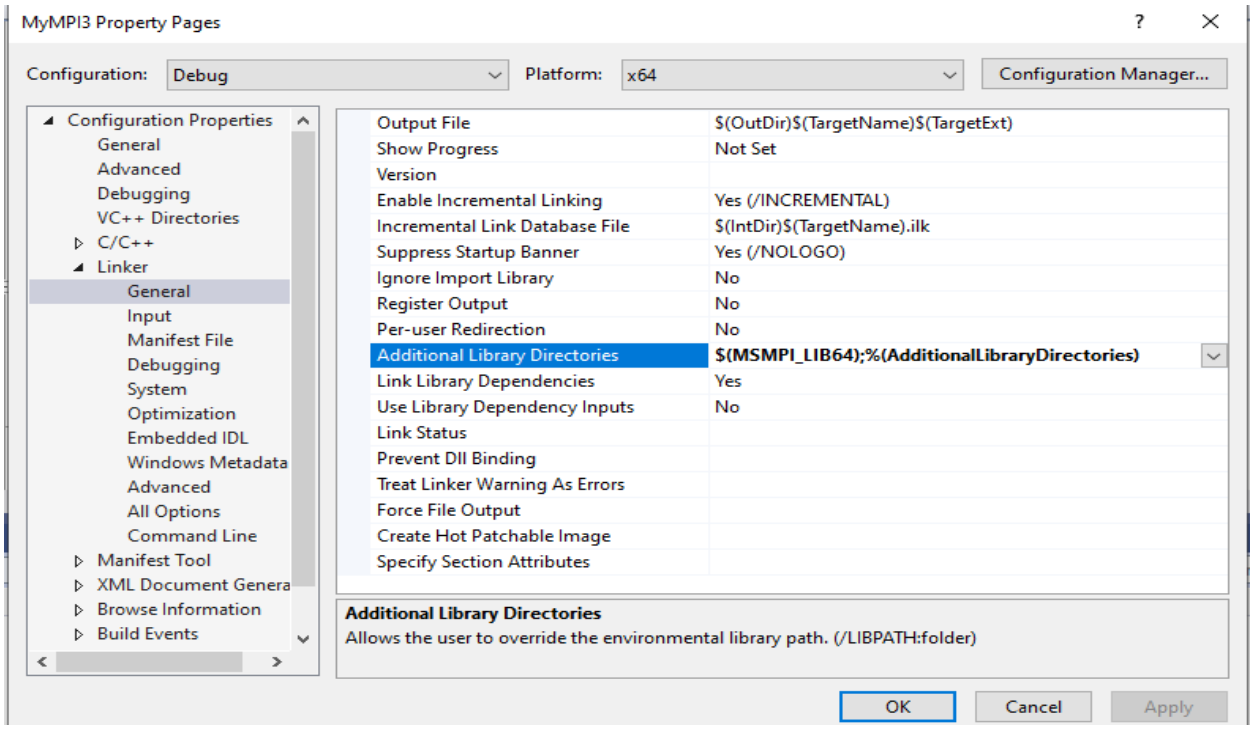
  $(MSMPI_LIB64)



Hình H1



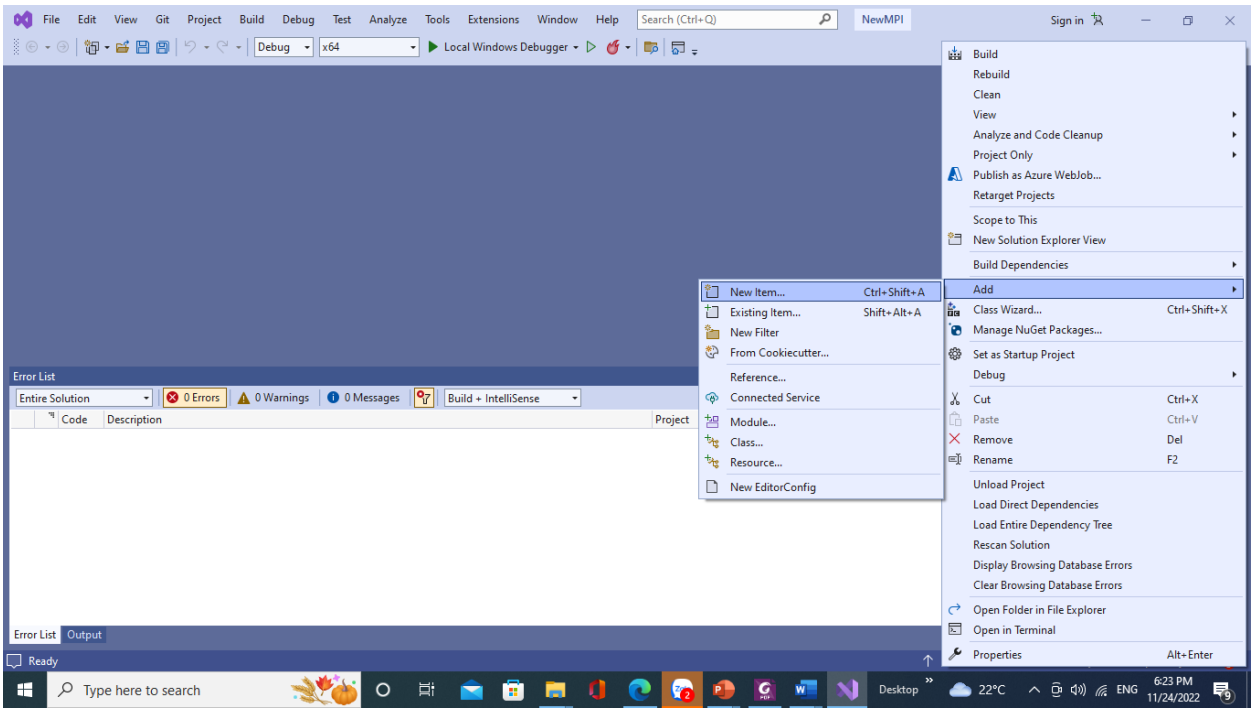
Hình H2

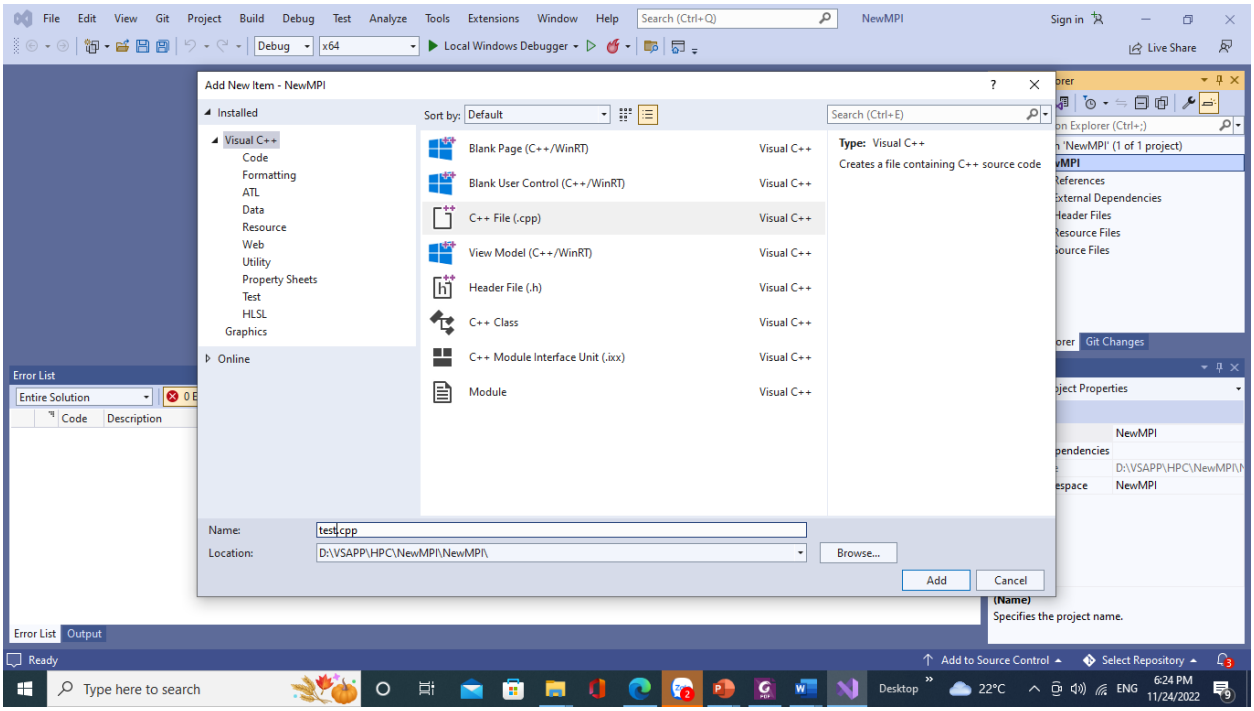


Hình H3

Bước 3: Tạo chương trình test

Bước 31: Từ cửa số dự án tạo tếp chương trình nguồn:





Bước 32: Nhập mã chuwong trình sau vào tệp test.cpp

```
//test.cpp
#include<stdio.h>
#include<mpi.h>
#include<stdlib.h>
int main(int argc, char* argv[])
{
	int myid, numprocs, namelen;
	char processor_name[MPI_MAX_PROCESSOR_NAME];
	MPI_Init(&argc, &argv);
	MPI_Comm_rank(MPI_COMM_WORLD, &myid);
	MPI_Comm_size(MPI_COMM_WORLD, &numprocs
	MPI_Get_processor_name(processor_name, &namelen);
	if (myid == 0) printf("number of processes: %d\n...",
numprocs);
	printf("%s: Hello world from process %d \n", processor_name,
myid);
	MPI_Finalize();
	return 0;
}
```

Bước 33: Chọn Build -> Build Solution để dịch chương trình tạo ra tệp test.exe

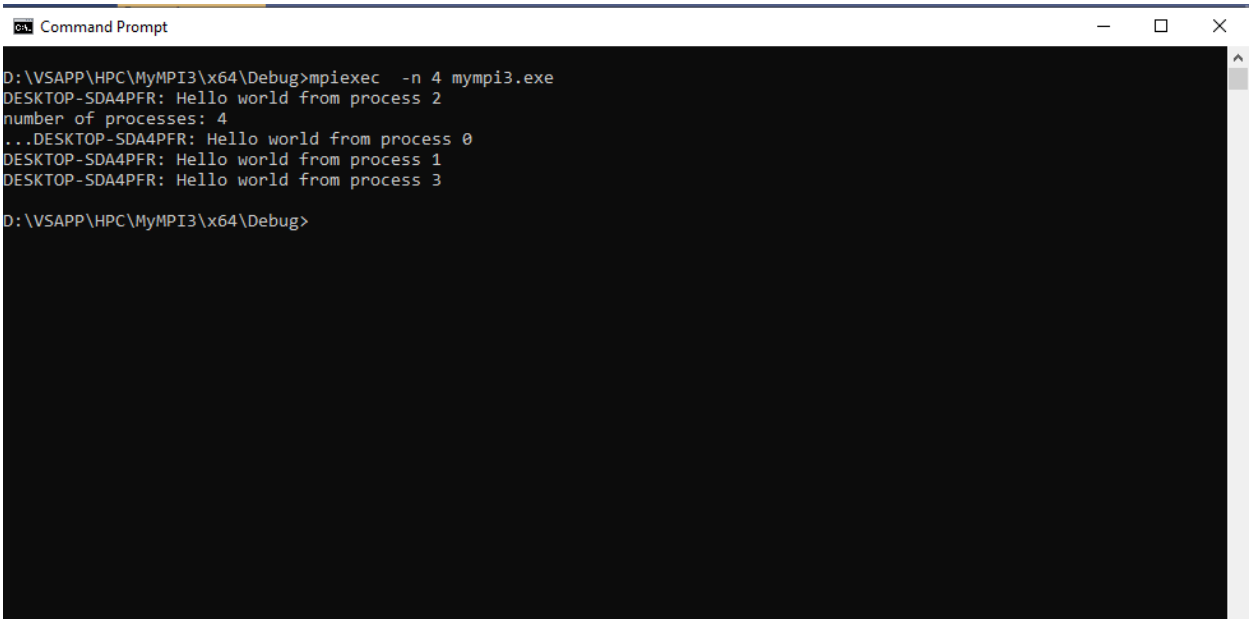
Bước 34: Chạy chương trình

- Mở cửa sổ console bằng lệnh cmd

- Đi đến thư mục dự án ->x84->debug chạy lệnh:

  mpiexec -n 4 test.exe [enter]

- Kết quả thu được:



PHẦN 2: LÀM CHỦ THƯ VIỆN MPI

2.1. Các cấu trúc liệt kê (eumeration)

```
typedef enum _MPI_Comm {}
typedef enum _MPI_Op{}
typedef enum _MPI_Datatype{}
```

Link:

https://learn.microsoft.com/en-us/message-passing-interface/mpi-enumerations

2.2. Các hàm MPI

-Yêu câu nghiên cứu và nắm được công dụng của các hàm thư viện MPI theo Link:

https://learn.microsoft.com/en-us/message-passing-interface/mpi-functions

Một số hàm thông dụng	
MPIAPI MPI_Comm_rank	MPI_File_write_at
MPIAPI MPI_Comm_size	MPI_File_close
MPIAPI MPI_Comm_split	MPI_Bcast
MPI_Send	MPIAPI MPI_Allgather
MPI_Recv	MPI_Allreduce

MPI_Recv_init	MPI_Alltoall
MPI_Rsend	MPI_Gather
MPI_Rsend_init	MPI_Iallreduce
MPI_Sendrecv	MPI_Reduce
MPI_Sendrecv_replace	MPI_Scatter
MPI_Ssend	MPI_Iallgather
MPI_Isend	MPI_Iallreduce
MPI_Issend	MPI_Barrier
MPI_Irecv	MPI_Ibarrier
MPI_Group_rank	MPI_Ibcast
MPI_Group_size	MPI_Igather
MPI_File_open	MPI_Igatherv
MPI_File_read	MPI_Ireduce
MPI_File_read_at	MPI_Iscatter
MPI_File_write	MPI_Iscatterv

2.3. Các cấu trúc

Link:

https://learn.microsoft.com/en-us/message-passing-interface/mpi-structs

PHẦN 3: LẬP TRÌNH MỘT SỐ CHƯƠNG TRÌNH MOI ĐOEN GIẢN

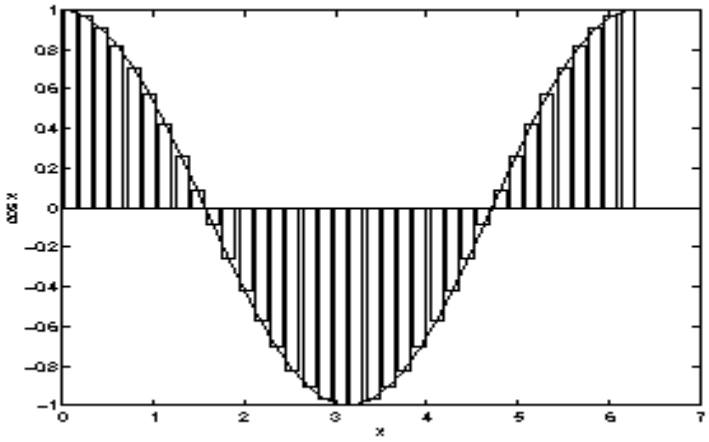
Bài 1: Thực hiện và giải thích kết quả các chương trình MPI ví dụ trong link sau:

https://curc.readthedocs.io/en/latest/programming/MPI-C.html

Bài 2: Tính toán tích phân số với luật Mid-point (Numerical Integration with Mid-point rule)

a) Tham khảo các luật:

https://math.libretexts.org/Courses/Mount_Royal_University/MATH_2200%3A_Calculus_for_Scientists_II/2%3A_Techniques_of_Integration/2.5%3A_Numerical_Integration_-_Midpoint%2C_Trapezoid%2C_Simpson's_rule



$$\int_a^b \cos(x)dx = \sum_{i=0}^{p-1}\sum_{j=0}^{n-1}\int_{a_{ij}}^{a_{ij}+h}\cos(x)dx$$

$$\simeq \sum_{i=0}^{p-1}\left[\sum_{j=0}^{n-1}\cos(a_{ij}+\frac{h}{2})h\right]$$

```
where p = # of processes
      n = number of intervals per process
      a = lower limit of integration
      b = upper limit of integration
      h = (b-a)/(n*p)
      aij = a +[ i*n +j]h
```

b) Viết chương trình tuần tự

c) Thực hiện chương trình song song

```c
/* C Example */
#include <mpi.h>
#include <math.h>
#include <stdio.h>
float fct(float x)
{
    return cos(x);
}
/* Prototype */
float integral(float a, int n, float h);
void main(int argc,char* argv[])
 {
    /********************************************************************
     *                                                                  *
     * This is one of the MPI versions on the integration example       *
     * It demonstrates the use of :                                     *
     *                                                                  *
```

```
     * 1) MPI_Init                                                        *
     * 2) MPI_Comm_rank                                                  *
     * 3) MPI_Comm_size                                                  *
     * 4) MPI_Recv                                                       *
     * 5) MPI_Send                                                       *
     * 6) MPI_Finalize                                                   *
     *                                                                   *
     *********************************************************************/
    int n, p, i, j, ierr, num;
    float h, result, a, b, pi;
    float my_a, my_range;

    int myid, source, dest, tag;
    MPI_Status status;
    float my_result;

    pi = acos(-1.0);  /* = 3.14159... */
    a = 0.;           /* lower limit of integration */
    b = pi * 1. / 2.;     /* upper limit of integration */
    n = 100000;          /* number of increment within each process */

    dest = 0;         /* define the process that computes the final result */
    tag = 123;        /* set the tag to identify this particular job */

    /* Starts MPI processes ... */

    MPI_Init(&argc, &argv);              /* starts MPI */
    MPI_Comm_rank(MPI_COMM_WORLD, &myid);  /* get current process id */
    MPI_Comm_size(MPI_COMM_WORLD, &p);     /* get number of processes */

    h = (b - a) / n;    /* length of increment */
    num = n / p;   /* number of intervals calculated by each process*/
    my_range = (b - a) / p;
    my_a = a + myid * my_range;
    my_result = integral(my_a, num, h);

    printf("Process %d has the partial result of %f\n", myid, my_result);

    if (myid == 0) {
        result = my_result;
        for (i = 1; i < p; i++) {
            source = i;           /* MPI process number range is [0,p-1] */
            MPI_Recv(&my_result, 1, MPI_REAL, source, tag,
                MPI_COMM_WORLD, &status);
            result += my_result;
        }
        printf("The result =%f\n", result);
    }
    else
        MPI_Send(&my_result, 1, MPI_REAL, dest, tag,
            MPI_COMM_WORLD);      /* send my_result to intended dest.
            */
    MPI_Finalize();                      /* let MPI finish up ... */
}
float integral(float a, int n, float h)
{
    int j;
    float h2, aij, integ;
```

```
    integ = 0.0;                /* initialize integral */
    h2 = h / 2.;
    for (j = 0; j < n; j++) {            /* sum over all "j" integrals */
        aij = a + j * h;        /* lower limit of "j" integral */
        integ += fct(aij + h2) * h;
    }
    return (integ);
}
```

Bài 3: Tình số PI theo thuật toán Motecarno

- Tìm hiểu thuật toán Montecarno và viết chương trình tuần tự
- Thực hiện chương trình tính toán song song
- So sánh thời gian giữa xử lý tuần tự với xử lý song song

Chương trình tính toán PI song song:

```
// This program is to caculate PI using MPI

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <mpi.h>

#define N 1E8
#define d 1E-8

int main(int argc, char* argv[])
{
    int rank, size, error, i, result = 0, sum = 0;
    double pi = 0.0, begin = 0.0, end = 0.0, x, y;

    error = MPI_Init(&argc, &argv);

    //Get process ID
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    //Get processes Number
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    //Synchronize all processes and get the begin time
    MPI_Barrier(MPI_COMM_WORLD);
    begin = MPI_Wtime();

    srand((int)time(0));

    //Each process will caculate a part of the sum
    for (i = rank; i < N; i += size)
    {
        x = rand() / (RAND_MAX + 1.0);
        y = rand() / (RAND_MAX + 1.0);
        if (x * x + y * y < 1.0)
            result++;
    }

    //Sum up all results
```

```
    MPI_Reduce(&result, &sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

    //Synchronize all processes and get the end time
    MPI_Barrier(MPI_COMM_WORLD);
    end = MPI_Wtime();

    //Caculate and print PI
    if (rank == 0)
    {
        pi = 4 * d * sum;
        printf("np=%2d;    Time=%fs;    PI=%0.4f\n", size, end - begin, pi);
    }

    error = MPI_Finalize();

    return 0;
}
```