Komunikacja kolektywna w środowisku MPI

Zakres ćwiczenia

W tym ćwiczeniu dowiesz się, co to jest komunikacja kolektywna i w jaki sposób napisać swój pierwszy program wykorzystujący komunikację kolektywną w środowisku MPI.

Komunikacja kolektywna w środowisku MPI

A. Idea komunikacji kolektywnej:

Komunikacja kolektywna jest kolejnym mechanizmem komunikacji w środowisku MPI. Polega ona na tym, że komunikaty są przesyłane między wszystkimi procesami należącymi do grupy określonej przez komunikator. Podstawowe funkcje MPI realizujące komunikację kolektywną to: `MPI_Bcast`, `MPI_Reduce`, `MPI_Barrier`, `MPI_Gather`, `MPI_Scatter`, `MPI_Allgather`, i `MPI_Allreduce`.

Funkcja `MPI_Bcast` rozgłasza komunikat do wszystkich procesów w komunikatorze. Komunikat rozgłaszania nie może być odbierany przez `MPI_Recv`. Funkcja `MPI_Reduce` wykonuje globalną operację na danych przesłanych od całej grupy, na przykład sumuje wyniki uzyskane we wszystkich procesach należących do grupy i umieszcza wynik operacji w jednym procesie. Funkcja `MPI_Barrier` zapewnia mechanizm synchronizacji wszystkich procesów w komunikatorze. Każdy proces blokuje się do czasu, aż wszystkie procesy wywołają `MPI_Barrier`. Funkcja `MPI_Gather` zbiera dane z buforów nadawczych od wszystkich procesów i konkatenuje je w kolejności rosnących numerów procesów, w buforze odbiorczym w procesie `root`. Z kolei funkcja `MPI_Scatter` jest w pewnym sensie odwrotnością funkcji `MPI_Gather`. Funkcja `MPI_Scatter` dzieli zawartość bufora nadawczego na części, i każdą część wysyła do kolejnego procesu, poczynając od procesu o numerze 0. Kolejna funkcja, `MPI_Allgather`, przypomina działaniem funkcję `MPI_Gather`, z tą różnicą, że wynik jest umieszczany w buforze odbiorczym każdego procesu, a nie tylko jednego procesu. Na koniec funkcja `MPI_Allreduce` przypomina działaniem funkcję `MPI_Reduce`, z tą różnicą, że wynik operacji jest umieszczany w każdym procesie, a nie tylko w jednym.

B. Wykorzystanie komunikacji kolektywnej w aplikacji równoległej:

Komunikacja kolektywna jest wprost predysponowana do zastosowania w przypadku zadań obliczeniowych, w których obliczenia mogą być podzielone na określoną liczbę mniejszych zadań, które charakteryzują się zbliżoną ilością obliczeń do wykonania, a które mogą być z powodzeniem wykonywane równolegle. Wtedy mniejsze zadania obliczeniowe są rozsyłane do poszczególnych procesów jedną funkcją kolektywną. Podobnie wyniki obliczeń uzyskane w poszczególnych procesach, są przetwarzane albo zbierane w jedno miejsce jedną funkcją kolektywną.

Przykładem takiego zadania jest obliczanie liczby π metodą całkowania numerycznego. W tym ćwiczeniu będziesz miał okazję samodzielnego przygotowania aplikacji równoległej, dokonującej obliczania liczby π z wykorzystaniem funkcji MPI realizujących komunikację kolektywną.

Przygotowanie aplikacji równoległej realizującej komunikację kolektywną

W celu przygotowania aplikacji równoległej, realizującej komunikację kolektywną w środowisku MPI, niezbędne jest podjęcie następujących kroków:

A. Przygotowanie kodów źródłowych programów w języku C++:

Przykładowy program w języku C++, przystosowany do uruchomienia w środowisku MPI, który korzysta z komunikacji kolektywnej, pokazano poniżej. Program ten znajdziesz w pliku `cxxpi.cxx` w katalogu `\mpich2-1.0.3\examples`, po rozpakowaniu pliku `mpich2-1.0.3.tar.gz`. Plik `mpich2-1.0.3.tar.gz` możesz pobrać ze strony implementacji MPICH2 standardu MPI-2: http://www-unix.mcs.anl.gov/mpi/mpich/.

Implementację MPICH2 zrealizowano w Argonne National Laboratory, który jest wiodącym ośrodkiem prowadzącym badania nad przetwarzaniem rozproszonym. Ponieważ w ramach niniejszych ćwiczeń korzystamy z implementacji MPICH2, ze zrozumiałych względów często będziemy się odwoływać do tych badań.

```
/* -*- Mode: C++; c-basic-offset:4 ; -*- */
/*
 *  (C) 2004 by Argonne National Laboratory.
 *      See COPYRIGHT in top-level directory.
 */

#include "mpi.h"
#include <iostream>
#include <math.h>

using namespace std;

double f(double);

double f(double a)
{
    return (4.0 / (1.0 + a*a));
}

int main(int argc,char **argv)
{
    int n, myid, numprocs, i;
    double PI25DT = 3.14159265358979323846264643;
    double mypi, pi, h, sum, x;
    double startwtime = 0.0, endwtime;
    int  namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    MPI::Init(argc,argv);
    numprocs = MPI::COMM_WORLD.Get_size();
    myid     = MPI::COMM_WORLD.Get_rank();
    MPI::Get_processor_name(processor_name,namelen);

    cout << "Process " << myid << " of " << numprocs << " is on "
<<
      processor_name << endl;
```

```
    n = 10000;                /* default # of rectangles */
    if (myid == 0)
      startwtime = MPI::Wtime();

    MPI::COMM_WORLD.Bcast(&n, 1, MPI_INT, 0);

    h   = 1.0 / (double) n;
    sum = 0.0;
    /* A slightly better approach starts from large i and works
back */
    for (i = myid + 1; i <= n; i += numprocs)
    {
      x = h * ((double)i - 0.5);
      sum += f(x);
    }
    mypi = h * sum;

    MPI::COMM_WORLD.Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM,
0);

    if (myid == 0) {
      endwtime = MPI::Wtime();
      cout << "pi is approximately " << pi << " Error is " <<
          fabs(pi - PI25DT) << endl;
      cout << "wall clock time = " << endwtime-startwtime <<
endl;
    }

    MPI::Finalize();
    return 0;
}
```

Zwróć uwagę na postać tego programu. W pętli głównej programu znajdują się znane już nam funkcje MPI: `MPI_init`, `MPI_Comm_rank`, `MPI_Comm_size`, i `MPI_Finalize`. Oprócz nich są dwie funkcje MPI realizujące komunikację kolektywną: `MPI_Bcast` i `MPI_Reduce`. Są ponadto dwie funkcje, których jeszcze nie znasz: `MPI_Get_processor_name` i `MPI_Wtime`. Pierwsza z nich wyświetla nazwę procesora, druga podaje czas, tzw. „wall-clock time", który jest czasem wyrażanym w sekundach, jaki upłynął od jakiejś chwili w przeszłości. Wszystkie wymienione tutaj funkcje wyróżniono cieniowaniem.

B. Kompilacja kodów źródłowych i konsolidacja z odpowiednimi bibliotekami:

1. Utworzenie projektu w środowisku Microsoft Visual Studio. NET

Uruchom środowisko Microsoft Visual Studio .NET. Pojawia okno Microsoft Development Environment [design] Start Page. Wybierz File→New→Blank Solution. Pojawi się wpis Blank Solution w okienko Solution Explorera w prawej górnej ćwiartce okna Start Page. Nadaj temu rozwiązaniu nazwę, w tym wypadku będzie to nazwa **liczba_pi**. Otrzymasz w okienku Solution Explorera wpis: Solution 'liczba_pi' (0 projects). Kliknij prawym przyciskiem myszy na Solution 'liczba_pi' i dodaj projekt do tego rozwiązania, za pomocą Add→New Project. Pojawi się okienko Add New Project. W okienku tym, w jego lewym panelu wybierz z listy wpis Visual C++ Projects, a w prawym panelu wybierz Managed C++ Application. Poniżej podaj nazwę projektu, tym razem **liczba_pi_proj**, i zatwierdź lokalizację projektu w następującym katalogu:

C:\Documents and Settings\michal\Moje dokumenty\Visual Studio Projects\liczba_pi

gdzie w miejsce podkatalogu `michal` powinna się pojawić Twoja nazwa użytkownika w systemie Windows.

W tym momencie, w okienku Solution Explorera pojawia się projekt o nazwie **liczba_pi_proj** z „drzewkiem" dołączonych plików. W projekcie **liczba_pi_proj** otrzymujesz strukturę, w której występują katalogi: Source Files, Header Files, i Reasource Files.

Cała struktura Solution 'liczba_pi' wygląda następująco:

```
Solution 'liczba_pi' (1 project)
      liczba_pi_proj
            Source Files
                  liczba_pi_proj.cpp
                  AssemblyInfo.cpp
                  stdafx.cpp
            Header Files
                  stdafx.h
            Resource Files
            ReadMe.txt
```

Głównym plikiem źródłowym Twojej aplikacji jest plik `liczba_pi_proj.cpp`. Zobacz co znajduje się w tym pliku. Jest to plik źródłowy, wygenerowany przez Application Wizard, który powinieneś uzupełnić kodem Twojego programu.

Plik `liczba_pi_proj.cpp` początkowo wygląda następująco:

```
// This is the main project file for VC++ application project
// generated using an Application Wizard.

#include "stdafx.h"

#using <mscorlib.dll>
#include <tchar.h>

using namespace System;

// This is the entry point for this application
int _tmain(void)
{
    // TODO: Please replace the sample code below with your own.
    Console::WriteLine(S"Hello World");
    return 0;
}
```

Kod Twojego programu, o nazwie `cxxpi.cxx`, znajdziesz w katalogu:

C:\Program Files\MPICH2\examples\moje przyklady\

do którego wcześniej powinieneś go skopiować z katalogu `examples`.

Przenieś teraz kod programu `cxxpi.cxx` do Solution Explorera rozwiązania **'liczba_pi'**, a dokładnie do projektu **liczba_pi_proj** w środowisku Microsoft Visual Studio .NET. W tym celu wejdź w okienko Solution Explorera, wybierz projekt **liczba_pi_proj,** kliknij prawym klawiszem myszy, wybierz Add →Add existing item i wskaż lokalizację:

`C:\Program Files\MPICH2\examples\moje przyklady\cxxpi.cxx`

Wtedy plik `cxxpi.cxx` pojawi się na liście Source Files w Solution Explorerze. Możesz go wyświetlić, klikając na jego nazwę. Zawartość pliku `cxxpi.cxx` pojawi się w lewym górnym okienku w oknie Microsoft Visual Studio .NET:

```
/* -*- Mode: C++; c-basic-offset:4 ; -*- */
/*
 *  (C) 2004 by Argonne National Laboratory.
 *      See COPYRIGHT in top-level directory.
 */

#include "mpi.h"
#include <iostream>
#include <math.h>

using namespace std;

double f(double);

double f(double a)
{
    return (4.0 / (1.0 + a*a));
}

int main(int argc,char **argv)
{
    int n, myid, numprocs, i;
    double PI25DT = 3.14159265358979323846264643;
    double mypi, pi, h, sum, x;
    double startwtime = 0.0, endwtime;
    int  namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    MPI::Init(argc,argv);
    numprocs = MPI::COMM_WORLD.Get_size();
    myid     = MPI::COMM_WORLD.Get_rank();
    MPI::Get_processor_name(processor_name,namelen);

    cout << "Process " << myid << " of " << numprocs << " is on "
<<
      processor_name << endl;

    n = 10000;                /* default # of rectangles */
    if (myid == 0)
      startwtime = MPI::Wtime();

    MPI::COMM_WORLD.Bcast(&n, 1, MPI_INT, 0);
```

```
    h   = 1.0 / (double) n;
    sum = 0.0;
    /* A slightly better approach starts from large i and works
back */
    for (i = myid + 1; i <= n; i += numprocs)
    {
      x = h * ((double)i - 0.5);
      sum += f(x);
    }
    mypi = h * sum;

    MPI::COMM_WORLD.Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM,
0);

    if (myid == 0) {
      endwtime = MPI::Wtime();
      cout << "pi is approximately " << pi << " Error is " <<
          fabs(pi - PI25DT) << endl;
      cout << "wall clock time = " << endwtime-startwtime <<
endl;
    }

    MPI::Finalize();
    return 0;
}
```

Oczywiście jest to ten sam program, który poznałeś na początku tego ćwiczenia.

Korzystając z Solution Explorera, podglądu obu programów, i operacji kopiowania *Ctrl-C* i wklejania *Ctrl-V*, wklej kod programu `cxxpi.cxx` do pliku `liczba_pi_proj.cpp`, za kodem pliku `liczba_pi_proj.cpp`. Otrzymasz następującą postać pliku `liczba_pi_proj.cpp`:

```
// This is the main project file for VC++ application project
// generated using an Application Wizard.

#include "stdafx.h"

#using <mscorlib.dll>
#include <tchar.h>

using namespace System;

// This is the entry point for this application
int _tmain(void)
{
    // TODO: Please replace the sample code below with your own.
    Console::WriteLine(S"Hello World");
    return 0;
}
/* -*- Mode: C++; c-basic-offset:4 ; -*- */
/*
 *  (C) 2004 by Argonne National Laboratory.
 *      See COPYRIGHT in top-level directory.
 */
```

```
#include "mpi.h"
#include <iostream>
#include <math.h>

using namespace std;

double f(double);

double f(double a)
{
    return (4.0 / (1.0 + a*a));
}

int main(int argc,char **argv)
{
    int n, myid, numprocs, i;
    double PI25DT = 3.14159265358979323846264643;
    double mypi, pi, h, sum, x;
    double startwtime = 0.0, endwtime;
    int  namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    MPI::Init(argc,argv);
    numprocs = MPI::COMM_WORLD.Get_size();
    myid     = MPI::COMM_WORLD.Get_rank();
    MPI::Get_processor_name(processor_name,namelen);

    cout << "Process " << myid << " of " << numprocs << " is on
" <<
      processor_name << endl;

    n = 10000;                /* default # of rectangles */
    if (myid == 0)
      startwtime = MPI::Wtime();

    MPI::COMM_WORLD.Bcast(&n, 1, MPI_INT, 0);

    h   = 1.0 / (double) n;
    sum = 0.0;
    /* A slightly better approach starts from large i and works
back */
    for (i = myid + 1; i <= n; i += numprocs)
    {
      x = h * ((double)i - 0.5);
      sum += f(x);
    }
    mypi = h * sum;

    MPI::COMM_WORLD.Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM,
0);

    if (myid == 0) {
      endwtime = MPI::Wtime();
      cout << "pi is approximately " << pi << " Error is " <<
          fabs(pi - PI25DT) << endl;
      cout << "wall clock time = " << endwtime-startwtime <<
```

```
endl;
    }

    MPI::Finalize();
    return 0;
}
```

Usuń w tym pliku wiersze od:

```
//This is the entry point ...
```

do końca automatycznie generowanego szablonu, czyli pierwszego wystąpienia nawiasu `}` po `return 0;`. Wiersze do usunięcia zacieniowano.

Otrzymujesz nową postać pliku `liczba_pi_proj.cpp`:

```
// This is the main project file for VC++ application project
// generated using an Application Wizard.

#include "stdafx.h"

#using <mscorlib.dll>
#include <tchar.h>

using namespace System;

/* -*- Mode: C++; c-basic-offset:4 ; -*- */
/*
 *  (C) 2004 by Argonne National Laboratory.
 *      See COPYRIGHT in top-level directory.
 */

#include "mpi.h"
#include <iostream>
#include <math.h>

using namespace std;

double f(double);

double f(double a)
{
    return (4.0 / (1.0 + a*a));
}

int main(int argc,char **argv)
{
    int n, myid, numprocs, i;
    double PI25DT = 3.14159265358979323846264643;
    double mypi, pi, h, sum, x;
    double startwtime = 0.0, endwtime;
    int  namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];
```

```
    MPI::Init(argc,argv);
    numprocs = MPI::COMM_WORLD.Get_size();
    myid     = MPI::COMM_WORLD.Get_rank();
    MPI::Get_processor_name(processor_name,namelen);

    cout << "Process " << myid << " of " << numprocs << " is on "
<<
      processor_name << endl;

    n = 10000;                /* default # of rectangles */
    if (myid == 0)
      startwtime = MPI::Wtime();

    MPI::COMM_WORLD.Bcast(&n, 1, MPI_INT, 0);

    h   = 1.0 / (double) n;
    sum = 0.0;
    /* A slightly better approach starts from large i and works
back */
    for (i = myid + 1; i <= n; i += numprocs)
    {
      x = h * ((double)i - 0.5);
      sum += f(x);
    }
    mypi = h * sum;

    MPI::COMM_WORLD.Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM,
0);

    if (myid == 0) {
      endwtime = MPI::Wtime();
      cout << "pi is approximately " << pi << " Error is " <<
          fabs(pi - PI25DT) << endl;
      cout << "wall clock time = " << endwtime-startwtime <<
endl;
    }

    MPI::Finalize();
    return 0;
}
```

Na koniec, usuń program `cxxpi.cxx` korzystając z listy Source Files w oknie Solution Explorera. Zaznacz program i naciśnij delete. Program `cxxpi.cxx` nie będzie już więcej potrzebny.

2. Kompilacja kodów źródłowych i konsolidacja z odpowiednimi bibliotekami

Proces kompilacji i konsolidacji rozpocznij od ustawienia odpowiednich cech w Twoim projekcie `liczba_pi_proj`.

W tym celu kliknij na nazwę projektu w oknie Solution Explorera. W okienku Properties, w prawym dolnym rogu okna Microsoft Visual Studio .NET, pojawia się napis: **liczba_pi_proj Project properties** oraz cztery ikony. Ikona z prawej nazywa się Property pages. Kliknij na nią, wtedy rozwinie się okienko z cechami. Wprowadź następujące ustawienia.

W wierszu Configuration Properties →C/C++ wybierz General. W panelu po prawej stronie, w wierszu Additional Include Directories wpisz, korzystając z listy podpowiedzi (po naciśnięciu przycisku **...** i ikony katalogu), następujący katalog:

```
C:\Program Files\MPICH2\include
```

W wierszu Configuration Properties →Linker wybierz General. W panelu po prawej stronie, w wierszu Additional Library Directories wpisz, korzystając z listy podpowiedzi (po naciśnięciu przycisku **...** i ikony katalogu), następujący katalog:

```
C:\Program Files\MPICH2\lib
```

W wierszu Configuration Properties →Linker wybierz Input. W panelu po prawej stronie, w wierszu Additional Dependencies wpisz, korzystając z listy podpowiedzi (po naciśnięciu przycisku **...** i ikony katalogu), następujące elementy na wejściu do konsolidatora (bez przecinków!):

```
mpi.lib cxx.lib
```

W tym momencie możesz dokonać kompilacji i konsolidacji programu.

Wybierz z paska okna Twojego projektu w środowisku Microsoft Visual Studio .NET, które gdy wyświetlasz program `liczba_pi_proj.cpp`, przyjmuje nazwę:

liczba_pi_proj Microsoft Visual C++ [design] liczba_pi_proj.cpp

polecenie Build →Build Solution.

Otrzymujesz następujący zapis z procesu kompilacji i konsolidacji, w lewym dolnym okienku okna projektu:

```
------ Build started: Project: liczba_pi_proj, Configuration: Debug Win32 ------

Compiling...
liczba_pi_proj.cpp
Linking...
LINK : warning LNK4098: defaultlib 'LIBCMT' conflicts with use of other libs; use
/NODEFAULTLIB:library
cxx.lib(initcxx1.obj) : warning LNK4204: 'c:\Documents and Settings\michal\Moje
dokumenty\Visual Studio Projects\liczba_pi\liczba_pi_proj\Debug\vc70.pdb' is
missing debugging information for referencing module; linking object as if no
debug info

Build log was saved at "file://c:\Documents and Settings\michal\Moje
dokumenty\Visual Studio Projects\liczba_pi\liczba_pi_proj\Debug\BuildLog.htm"
liczba_pi_proj - 0 error(s), 2 warning(s)


---------------------- Done ----------------------

    Build: 1 succeeded, 0 failed, 0 skipped
```

Kompilacja i konsolidacja zakończyły się poprawnie, niemniej podczas konsolidacji pojawiło się ostrzeżenie, które w tej chwili pominiemy. Zobacz teraz, co otrzymałeś jako wynik procesu kompilacji i konsolidacji. W tym celu przejdź do katalogu z Twoim projektem:

```
C:\Documents and Settings\michal\Moje dokumenty\Visual Studio
Projects\liczba_pi\liczba_pi_proj
```

Wejdź następnie do katalogu `Debug`. Sprawdź, czy jest tam plik `liczba_pi_proj.exe`. Jest to plik wynikowy z Twoim projektem.

3. Uruchomienie programu z wiersza poleceń systemu operacyjnego Windows

Aby sprawdzić czy program w ogóle działa, bez angażowania początkowo poleceń implementacji MPICH2 środowiska MPI, wystarczy uruchomić program `liczba_pi_proj.exe` z wiersza poleceń systemu operacyjnego Windows.

W tym celu wejdź w polecenie Uruchom i wpisz `cmd`. Pojawi się okno poleceń systemu Windows. Przejdź do katalogu:

```
C:\Documents and Settings\michal\Moje dokumenty\Visual Studio
Projects\liczba_pi\liczba_pi_proj\Debug
```

Pamiętaj, że zamiast podkatalogu `michal` jest Twoja nazwa użytkownika.

Sprawdź istnienie pliku `liczba_pi_proj.exe` poleceniem `dir liczba_pi_proj.exe`. Pojawi się okno z informacją o programie `liczba_pi_proj.exe`:

```
C:\Documents and Settings\michal\Moje dokumenty\Visual Studio
Projects\liczba_pi
\liczba_pi_proj\Debug>dir liczba_pi_proj.exe
 Wolumin w stacji C to IBM_PRELOAD
 Numer seryjny woluminu: 303C-7AE1

 Katalog: C:\Documents and Settings\michal\Moje
dokumenty\Visual Studio Projects
\liczba_pi\liczba_pi_proj\Debug

2006-08-02  18:24           475 136 liczba_pi_proj.exe
               1 plik(ów)         475 136 bajtów
               0 katalog(ów)  40 760 553 472 bajtów wolnych

C:\Documents and Settings\michal\Moje dokumenty\Visual Studio
Projects\liczba_pi
\liczba_pi_proj\Debug>
```

W oknie tym uruchom program, wpisując `liczba_pi_proj.exe`. Otrzymasz następujący obraz okna poleceń:

```
C:\Documents and Settings\michal\Moje dokumenty\Visual Studio
Projects\liczba_pi
\liczba_pi_proj\Debug>liczba_pi_proj.exe
Process 0 of 1 is on ms
pi is approximately 3.14159 Error is 8.33341e-010
wall clock time = 0.00154908

C:\Documents and Settings\michal\Moje dokumenty\Visual Studio
Projects\liczba_pi
\liczba_pi_proj\Debug>
```

Widać, że program się wykonał, generując napis:

```
Process 0 of 1 is on ms
pi is approximately 3.14159 Error is 8.33341e-010
wall clock time = 0.00154908
```

Otrzymałeś wynik wykonania Twojego programu. Do jego wytworzenia wykorzystano jednak tylko jeden proces, gdyż zgłosił się jedynie proces o numerze 0. W kolejnym punkcie przekonasz się jak uruchomić Twój program z wykorzystaniem wielu procesów.

4. Uruchomienie programu w środowisku MPI

Twoim zadaniem jest uruchomienie programu `liczba_pi_proj.exe` z wykorzystaniem wielu procesów. Rozpocznij od uruchomienia środowiska MPI.

Uruchom aplikację MPICH2, w zwykły sposób: Start→Wszystkie programy → MPICH2. Pokazuje się lista opcji, wybierz program `wmpiexec` realizujący interfejs graficzny do uruchamianie Twojego programu w środowisku MPI. Pojawia się okno programu `wmpiexec`, w którym możesz wskazać plik wynikowy Twojego programu, oraz zaznaczyć, że chcesz go uruchomić na określonej liczbie procesorów. Okno programu `wmpiexec` nazywa się MPIEXEC wrapper. W oknie tym dokonaj następujących wpisów.

W okienku Application wybierz pełną ścieżkę do pliku wynikowego, w tym wypadku jest to plik `liczba_pi_proj.exe`. Pełna ścieżka do tego pliku to:

```
"C:\Documents and Settings\michal\Moje dokumenty\Visual Studio
Projects\liczba_pi\liczba_pi_proj\Debug\liczba_pi_proj.exe"
```

Najprościej wybrać ją korzystając z przycisku przeglądania, oznaczonego trzema kropkami.

Następnie ustaw liczbę procesów na 4, zaznacz polecenie uruchomienia w oddzielnym oknie: run in separate window, oraz wybierz myszką przycisk show command, którego naciśnięcie wyzwala pokazanie wykorzystanego do uruchomienia programu polecenia środowiska MPI. Na pasku okna MPIEXEC wrapper pojawia się pełna ścieżka do polecenia mpiexec:

```
"C:\Program Files\MPICH2\bin\mpiexec.exe" -n 4  -noprompt
"C:\Documents and Settings\michal\Moje dokumenty\Visual Studio
Projects\liczba_pi\liczba_pi_proj\Debug\liczba_pi_proj.exe"
```

Następnie wskaż myszką przycisk execute. Naciśnięcie tego przycisku wyzwala wydanie polecenia mpiexec w środowisku MPI-2, w postaci pokazanej powyżej. W wyniku jego wykonania otrzymujemy wynik programu w oknie poleceń systemu Windows:

```
Process 2 of 4 is on ms
Process 1 of 4 is on ms
Process 3 of 4 is on ms
Process 0 of 4 is on ms
pi is approximately 3.14159 Error is 8.33331e-010
wall clock time = 0.0176562
Aby kontynuować, naciśnij dowolny klawisz . . .
```

Zauważ, że każdy z 4 procesów, o numerach od 0 do 3, się zgłosił. Przy kolejnym uruchomieniu, procesy zgłaszają się w innej kolejności:

```
Process 1 of 4 is on ms
Process 3 of 4 is on ms
Process 2 of 4 is on ms
Process 0 of 4 is on ms
pi is approximately 3.14159 Error is 8.33331e-010
wall clock time = 0.0170569
Aby kontynuować, naciśnij dowolny klawisz . . .
```

Zauważ, że przy stałej liczbie przedziałów, w których program dokonuje całkowania, wynoszącej 10 000, przy każdym uruchomieniu otrzymujesz tę samą wartość liczby π.

Na koniec będziesz miał okazję zobaczyć, jak działa program obliczający liczbę π, w wersji interakcyjnej. Wersja ta pozwoli Tobie na wybór liczby przedziałów całkowania.

Obliczanie liczby π metodą całkowania numerycznego

A. W celu obliczenia liczby π metodą interakcyjną wykonaj następujące kroki:

Uruchom program wynikowy o nazwie `cpi.exe` z katalogu:

```
C:\Program Files\MPICH2\examples
```

za pomocą aplikacji MPICH2. W tym celu przejdź: Start→Wszystkie programy→MPICH2→wmpiexec. W oknie programu wmpiexec wskaż ścieżkę do pliku:

```
C:\Program Files\MPICH2\examples\cpi.exe
```

W pliku `cpi.exe` znajduje się plik wynikowy z interakcyjną wersją obliczania liczby π. Wpisz liczbę 4 procesów, zaznacz wykonanie w oddzielnym oknie i wyświetlenie postaci polecenia `mpiexec`. Wybierz przycisk execute. Otrzymasz okno poleceń z następującą informacją:

```
Enter the number of intervals: (0 quits)
```

Wpisz w oknie poleceń dowolną liczbę przedziałów, np. 100. Następnie kolejno zwiększaj tę liczbę dziesięciokrotnie. Otrzymasz następujący obraz w oknie poleceń:

```
Enter the number of intervals: (0 quits) 100
pi is approximately 3.1416009869231249, Error is
0.0000083333333318
wall clock time = 0.011448
Enter the number of intervals: (0 quits) 1000
pi is approximately 3.1415927369231267, Error is
0.0000000833333336
wall clock time = 0.000426
Enter the number of intervals: (0 quits) 10000
pi is approximately 3.1415926544231239, Error is
0.0000000008333307
wall clock time = 0.000583
Enter the number of intervals: (0 quits) 100000
pi is approximately 3.1415926535981167, Error is
0.0000000000083236
wall clock time = 0.002834
Enter the number of intervals: (0 quits) 1000000
pi is approximately 3.1415926535899033, Error is
0.0000000000001101
wall clock time = 0.019539
Enter the number of intervals: (0 quits) 10000000
pi is approximately 3.1415926535896865, Error is
0.0000000000001066
wall clock time = 0.185118
Enter the number of intervals: (0 quits) 100000000
pi is approximately 3.1415926535902168, Error is
0.0000000000004237
wall clock time = 1.846578
Enter the number of intervals: (0 quits) 1000000000
pi is approximately 3.1415926535897682, Error is
0.0000000000000249
wall clock time = 18.513324
Enter the number of intervals: (0 quits)
```

Zwróć uwagę, jak zmienia się wartość liczby π w zależności od liczby przedziałów całkowania i dla jakiej liczby przedziałów błąd obliczeń jest najmniejszy.

Kod źródłowy programu obliczania liczby π w wersji interakcyjnej znajdziesz w pliku `icpi.c` w katalogu:

```
C:\Program Files\MPICH2\examples
```

Kod ten ma następującą postać:

```
/* -*- Mode: C; c-basic-offset:4 ; -*- */
/*
 *  (C) 2001 by Argonne National Laboratory.
 *      See COPYRIGHT in top-level directory.
 */

/* This is an interactive version of cpi */
```

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

double f(double);

double f(double a)
{
    return (4.0 / (1.0 + a*a));
}

int main(int argc,char *argv[])
{
    int done = 0, n, myid, numprocs, i;
    double PI25DT = 3.14159265358979323846264 3;
    double mypi, pi, h, sum, x;
    double startwtime = 0.0, endwtime;
    int  namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    MPI_Get_processor_name(processor_name,&namelen);

    /*
    fprintf(stdout,"Process %d of %d is on %s\n",
          myid, numprocs, processor_name);
    fflush(stdout);
    */

    while (!done) {
        if (myid == 0) {
            fprintf(stdout, "Enter the number of intervals: (0
quits) ");
          fflush(stdout);
            if (scanf("%d",&n) != 1) {
            fprintf( stdout, "No number entered; quitting\n" );
            n = 0;
          }
          startwtime = MPI_Wtime();
        }
        MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
        if (n == 0)
            done = 1;
        else {
            h   = 1.0 / (double) n;
            sum = 0.0;
            for (i = myid + 1; i <= n; i += numprocs) {
                x = h * ((double)i - 0.5);
                sum += f(x);
            }
            mypi = h * sum;
            MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0,
MPI_COMM_WORLD);
```

```
            if (myid == 0) {
                printf("pi is approximately %.16f, Error is
%.16f\n",
                       pi, fabs(pi - PI25DT));
            endwtime = MPI_Wtime();
            printf("wall clock time = %f\n", endwtime-
startwtime);
            fflush( stdout );
          }
        }
    }
    MPI_Finalize();
    return 0;
}
```

Zadanie do samodzielnego wykonania

Dokonaj kompilacji, konsolidacji oraz uruchomienia programu obliczania liczby π w wersji interakcyjnej. `icpi.c`, z poprzedniego punktu, zgodnie z poznanymi w tym ćwiczeniu zasadami. Dokonaj analizy otrzymanych wyników. Porównaj te wyniki z wynikami uzyskanymi przy uruchomieniu gotowego programu wynikowego `cpi.exe`.

Podsumowanie

Podczas tego ćwiczenia mogłeś poznać i praktycznie sprawdzić funkcje komunikacji kolektywnej `MPI_Bcast` i `MPI_Reduce` w środowisku implementacji MPICH standardu MPI-2.

Co powinieneś wiedzieć:

- Na czym polega obliczanie liczby π metodą całkowania numerycznego.
- Jakie zadania obliczeniowe warto zrównoleglić.
- Co to jest komunikacja kolektywna.