

Sprawozdanie z przedmiotu Programowanie Równoległe i Rozproszone

MPI (Message Passing Interface)

Wykonał:

Imię i nazwisko

Magdalena Paszko

Nr indeksu

76024

Grupa

L2

Prowadzący : dr inż. Krzysztof Szerszeń

1. Wprowadzanie

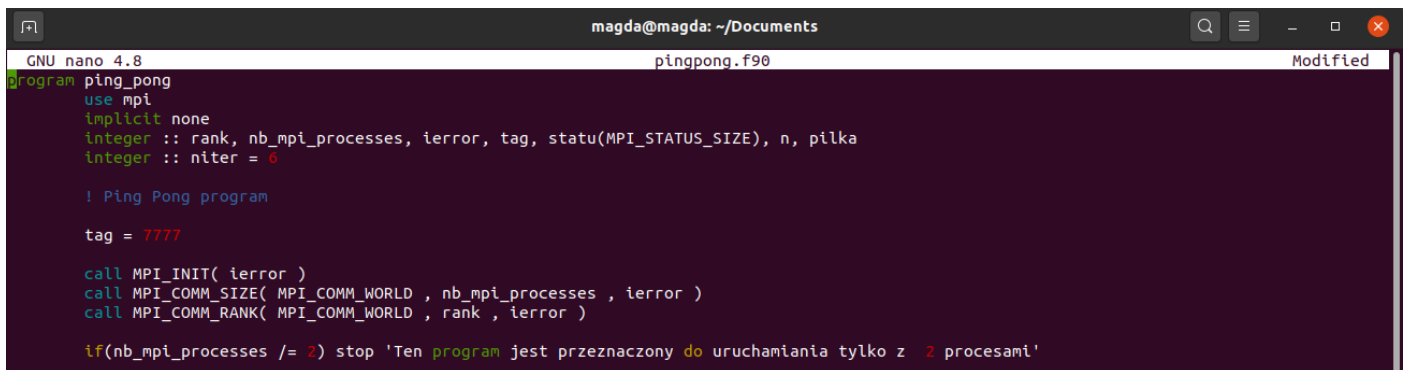
Fortran (od wersji 90 do aktualnej) a dawniej FORTRAN (do wersji 77 włącznie) (od ang. formula translation) – język programowania pierwotnie zaprojektowany do zapisu programów obliczeniowych, był niegdyś językiem proceduralnym, obecnie jest nadal rozwijanym językiem ogólnego przeznaczenia. Umożliwia programowanie strukturalne, obiektowe (Fortran 90/95), modułowe i równoległe (Fortran 2008). Jego zastosowaniami są, między innymi, obliczenia naukowo-inżynierskie, numeryczne, symulacja komputerowa itp. Początkowe wersje Fortranu miały mocno ograniczone możliwości, ale dzięki łatwości opanowania, Fortran stał się najpopularniejszym językiem do obliczeń numerycznych.

2. Definicja:

Aplikacja MPI składa się z dwóch procesów: P0 oraz P1. Obydwa procesy wymieniają się komunikatami stosując wzorzec wymiany jak w grze w ping-ponga. P0 wysyła komunikat do P1 następnie P1 odsyła komunikat do P0. Ta sekwencja wymiany jest powtarzana odpowiednią liczbę iteracji, tak aby program wykonywał się przez kilka sekund.

3. Implementacja algorytmów

a) Symulacja gry typu ping pong w Fortran:



```
GNU nano 4.8                                pingpong.f90                                Modified
program ping_pong
  use mpi
  implicit none
  integer :: rank, nb_mpi_processes, ierror, tag, stat(MPI_STATUS_SIZE), n, pilka
  integer :: niter = 6

  ! Ping Pong program

  tag = 7777

  call MPI_INIT( ierror )
  call MPI_COMM_SIZE( MPI_COMM_WORLD , nb_mpi_processes , ierror )
  call MPI_COMM_RANK( MPI_COMM_WORLD , rank , ierror )

  if(nb_mpi_processes /= 2) stop 'Ten program jest przeznaczony do uruchamiania tylko z 2 procesami'
```

Implementujemy zmienne oraz środowisko wykonywania programu. Dopiero od momentu wywołania MPI_Init można używać pozostałych funkcji MPI. Za pomocą funkcji MPI_COMM_SIZE jest określana liczba procesów, a za pomocą MPI_COMM_RANK identyfikator procesu. Następnie sprawdzamy czy na pewno są tylko dwa procesy, ponieważ inaczej program nie będzie działał poprawnie.

```

if(nb_mpi_processes /= 2) stop 'This program is design to be run with 2 processes only'

pilka = 0
do n=1,niter

    if(rank==0) then
        call MPI_SEND ( pilka , 1 , MPI_INTEGER , 1 , tag , MPI_COMM_WORLD , ierror)
        call MPI_RECV ( pilka , 1 , MPI_INTEGER , 1 , tag , MPI_COMM_WORLD , statu , ierror )
        pilka = pilka + 2
    end if

    if(rank==1) then
        call MPI_RECV ( pilka , 1 , MPI_INTEGER , 0 , tag , MPI_COMM_WORLD , statu , ierror )
        pilka = pilka + 1
        call MPI_SEND ( pilka , 1 , MPI_INTEGER , 0 , tag , MPI_COMM_WORLD , ierror )
    end if

    print*, 'Proces',rank,'iter',n,'wartosc pileczki wynisi:',pilka

    call MPI_BARRIER(MPI_COMM_WORLD,ierror)

end do

call MPI_FINALIZE ( ierror ) ! Close MPI
end program ping_pong

```

Po kolei sprawdzamy czy jest to proces 0 czy 1. Jeśli 0 wysyłany jest komunikat do procesu 1 za pomocą funkcji MPI_Send, za pomocą MPI_RECV komunikat jest odbierany z kolejki później następuje dodawanie.

W procesie 1 komunikat jest odbierany, później odpowiednio dodawanie i „odbicie” wysyłanie danych.

Na koniec w zależności na jaki proces padło wyświetlana informacja o numerze procesu, iteracji i wartości w danym momencie.

Wynik programu:

```

magda@magda:~/Documents$ mpif90 pingpong.f90 -o ping
magda@magda:~/Documents$ mpirun -n 2 ./ping
Proces      0 iter      1 wartosc pileczki wynisi:      3
Proces      1 iter      1 wartosc pileczki wynisi:      1
Proces      0 iter      2 wartosc pileczki wynisi:      6
Proces      1 iter      2 wartosc pileczki wynisi:      4
Proces      0 iter      3 wartosc pileczki wynisi:      9
Proces      1 iter      3 wartosc pileczki wynisi:      7
Proces      0 iter      4 wartosc pileczki wynisi:     12
Proces      1 iter      4 wartosc pileczki wynisi:     10
Proces      0 iter      5 wartosc pileczki wynisi:     15
Proces      1 iter      5 wartosc pileczki wynisi:     13
Proces      0 iter      6 wartosc pileczki wynisi:     18
Proces      1 iter      6 wartosc pileczki wynisi:     16
magda@magda:~/Documents$

```

b) Symulacja gry typu ping pong w C:

Analogiczny program jak wyżej tylko napisany w C

```
GNU nano 4.8                                pingpong.c
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

// Ping Pong program

int main(int argc, char** argv)
{
    MPI_Init(NULL, NULL);
    int nb_mpi_processes;
    MPI_Comm_size(MPI_COMM_WORLD, &nb_mpi_processes);
    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    if(nb_mpi_processes != 2) { printf("Ten program jest przeznaczony do uruchamiania tylko z 2 procesami"); return 0;}

    int tag = 7777;
    int pilka = 0;
    int n;
    int niter = 6;

    int tag = 7777;
    int pilka = 0;
    int n;
    int niter = 6;
    for (n=1;n<=niter;n++)
    {
        if(rank==0) {
            MPI_Send ( &pilka , 1 , MPI_INTEGER , 1 , tag , MPI_COMM_WORLD );
            MPI_Recv ( &pilka , 1 , MPI_INTEGER , 1 , tag , MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            pilka = pilka + 2;
        }

        if(rank==1) {
            MPI_Recv ( &pilka , 1 , MPI_INTEGER , 0 , tag , MPI_COMM_WORLD, MPI_STATUS_IGNORE );
            pilka = pilka + 1;
            MPI_Send ( &pilka , 1 , MPI_INTEGER , 0 , tag , MPI_COMM_WORLD );
        }

        printf("Process %d iter %d wartosc pilki : %d\n",rank,n, pilka);

        MPI_Barrier(MPI_COMM_WORLD);
    }

    MPI_Finalize();

    return 0;
}
```

Wynik programu:

```
magda@magda:~/Documents$ mpicc pingpong.c -o pingpong.exe -lm
magda@magda:~/Documents$ mpirun --oversubscribe pingpong.exe
Process 0 iter 1 wartosc pilki : 3
Process 1 iter 1 wartosc pilki : 1
Process 0 iter 2 wartosc pilki : 6
Process 1 iter 2 wartosc pilki : 4
Process 0 iter 3 wartosc pilki : 9
Process 1 iter 3 wartosc pilki : 7
Process 0 iter 4 wartosc pilki : 12
Process 1 iter 4 wartosc pilki : 10
Process 0 iter 5 wartosc pilki : 15
Process 1 iter 5 wartosc pilki : 13
Process 0 iter 6 wartosc pilki : 18
Process 1 iter 6 wartosc pilki : 16
```