Przemysław Szymoniak

Przetwarzanie współbieżne. Programowanie równoległe i rozproszone.

Sprawozdanie z laboratorium 12.
Celem laboratorium opanowanie podstaw programowania z przesyłaniem komunikatów MPI.

W ramach zajęć zrealizowałem następujące kroki:

- Pobrałem i rozpakowałem wszystkie wymagane pliki, utworzyłem wymagane katalogi oraz uruchomiłem program.
- Uzupełniłem program o przesyłanie w tablicy znaków adresu internetowego węzła nadawcy:

```
int rank, ranksent, size, source, dest, tag, i;
MPI_Status status;
char host[256], hostrec[256];
MPI_Init( &argc, &argv );
MPI_Comm_rank( MPI_COMM_WORLD, &rank );
MPI_Comm_size( MPI_COMM_WORLD, &size );
size_t length;
gethostname(&host, length);

if(size>1){
    if( rank != 0 ){
        dest=0; tag=0;
        MPI_Send( &host, 10, MPI_CHAR, dest, tag, MPI_COMM_WORLD );
    } else {
        for( i=1; i<size; i++ ) {
        MPI_Recv( &hostrec, 10, MPI_CHAR, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &status );
        printf("Dane od procesu o randze (i=%d): %s (%d)\n", i,hostrec, status.MPI_SOURCE );
        }
    }
}
```

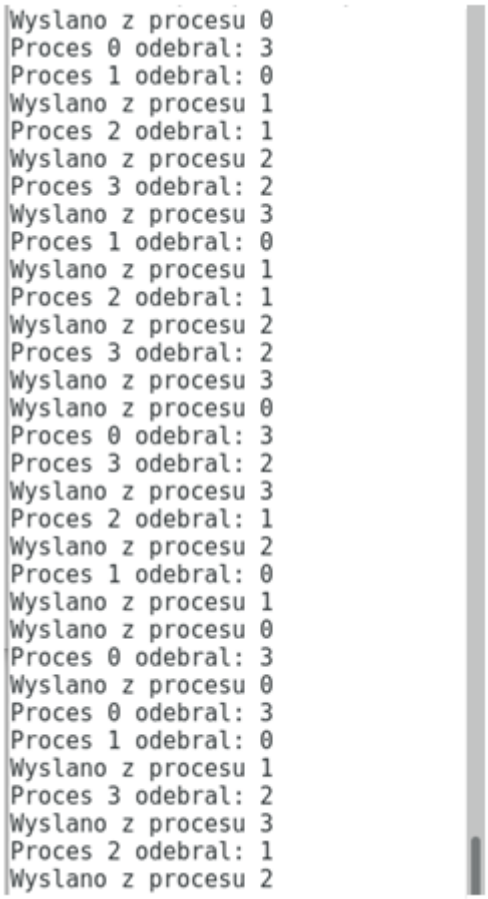
- Kod sztafety (zamknięty pierścień):

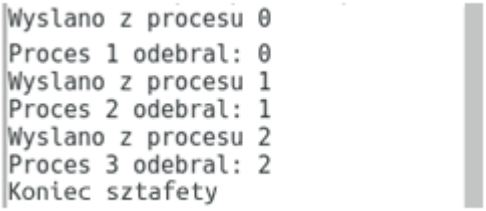
```
int rank, ranksent, size, source, dest, tag, i;
MPI_Status status;
MPI_Init( &argc, &argv );
MPI_Comm_rank( MPI_COMM_WORLD, &rank );
MPI_Comm_size( MPI_COMM_WORLD, &size );

int j;
for (j=0; j<4; j++){
    if( rank == 0 ){
        dest=rank+1;
        MPI_Send( &rank, 1, MPI_INT, dest, tag, MPI_COMM_WORLD );
        printf("Wyslano z procesu %d\n", rank);
        MPI_Recv( &ranksent, 1, MPI_INT, size-1, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
        printf("Proces %d odebral: %d\n", rank, ranksent);
    }else if (rank==size-1){
        dest=0;
        MPI_Recv( &ranksent, 1, MPI_INT, rank-1, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
        printf("Proces %d odebral: %d\n", rank, ranksent);
        MPI_Send( &rank, 1, MPI_INT, dest, tag, MPI_COMM_WORLD );
        printf("Wyslano z procesu %d\n", rank);
    }else{
        dest=rank+1;
        MPI_Recv( &ranksent, 1, MPI_INT, rank-1, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
        printf("Proces %d odebral: %d\n", rank, ranksent);
        MPI_Send( &rank, 1, MPI_INT, dest, tag, MPI_COMM_WORLD );
        printf("Wyslano z procesu %d\n", rank);
    }
}
```

- Sztafeta (ostatni kończy):

```
int rank, ranksent, size, source, dest, tag, i;
MPI_Status status;
MPI_Init( &argc, &argv );
MPI_Comm_rank( MPI_COMM_WORLD, &rank );
MPI_Comm_size( MPI_COMM_WORLD, &size );

int j;
for (j=0; j<4; j++){
    if( rank == 0 ){
        dest=rank+1;
        MPI_Send( &rank, 1, MPI_INT, dest, tag, MPI_COMM_WORLD );
        printf("Wyslano z procesu %d\n", rank);
    }else if (rank==size-1){
        dest=0;
        MPI_Recv( &ranksent, 1, MPI_INT, rank-1, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
        printf("Proces %d odebral: %d\n", rank, ranksent);
        MPI_Send( &rank, 1, MPI_INT, dest, tag, MPI_COMM_WORLD );
        printf("Wyslano z procesu %d\n", rank);
    }else{
        MPI_Recv( &ranksent, 1, MPI_INT, rank-1, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
        printf("Proces %d odebral: %d\n", rank, ranksent);
        printf("Koniec sztafety");
        break;
    }
}
```

- Działanie:





Wnioski:

- Polecenie MPI_send(…) jest blokującą procedurą wysyłania w MPI. Blokująca oznacza, że sterowanie jest przekazywane z powrotem po wykonaniu pełnego wysłania danych, nie tak jak w przypadku procedury nie blokującej MPI_Isend(…), gdzie wysyłanie jest jedynie inicjowane a następnie sterowanie wraca. Wysłanie danych odbywa się później np. przy użyciu innego wątku.
- Polecenie MPI_recv(…) jest tak samo jak poprzednik procedurą blokującą, próba odebrania danych więc odbywana się od razu po wywołaniu funkcji a sterowanie wraca w momencie uzyskania danych. Procedurą nie blokującą jest MPI_Irecv(…), która również tylko inicjuje odbieranie danych odbywające się z opóźnieniem.
- Polecenia MPI_send(…) oraz MPI_recv(…) posiadając możliwość określenia do którego wątku dane mają zostać wysłane oraz od którego odebrane. Dodatkowo funckja MPI_recv(…) posiada argument MPI_Status zwracający informację o statusie wykonania operacji.
- Wykorzystując MPI jesteśmy w stanie wysyłać różne typy danych zaczynając od CHAR kończąc na BYTE. Większość nazw typów pokrywa się z tymi wykorzystywanymi w programowaniu wymagają one jednak wykorzystania przedrostka MPI_ np.(MPI_DOUBLE).
- W naszym algorytmie wykorzystujemy również funkcje MPI_Comm_rank(…) jest ona odpowiedzialna za pobranie numeru, identyfikatora wywołującego procesu zapisuje go w zmiennej rank typu int *.
- Natomiast funkcja MPI_Comm_size(…) odpowiada za pobranie liczby procesów występujących w obrębie komunikatora MPI_Comm zapisując tą wielkość do zmiennej typu int * o nazwie size;
- Funkcjami również wymaganymi do działania aplikacji jest MPI_Init(…) odpowiedzialna za inicjowanie środowiska wykonywania programu. Jest to między innymi tworzenie domyślnego komunikatora MPI_COMM_WORLD.
- Na zakończenie wykorzystywania MPI powinniśmy zwolnić wszystkie używane zasoby. Możemy to wykonać funkcją MPI_Finalize(). Dodatkowo ta funkcja przygotowuje system do zamknięcia.
- Odbierając informacje od wątków możemy podać konkretny identyfikator. Jednak w momencie gdy nie wiemy z którego wątku przyjdą oczekiwane informacje możemy wykorzystać flagę MPI_ANY_SOURCE. Jej ustawienie sprawia, że wątek odczyta pierwszy natrafiony komunikat od dowolnego procesu.
- Ustawienie MPI_ANY_TAG powoduje, że podczas odbierania nie będziemy sprawdzać znacznika typu wiadomości jedynie odbierzemy komunikat, który przyjdzie najwcześniej.