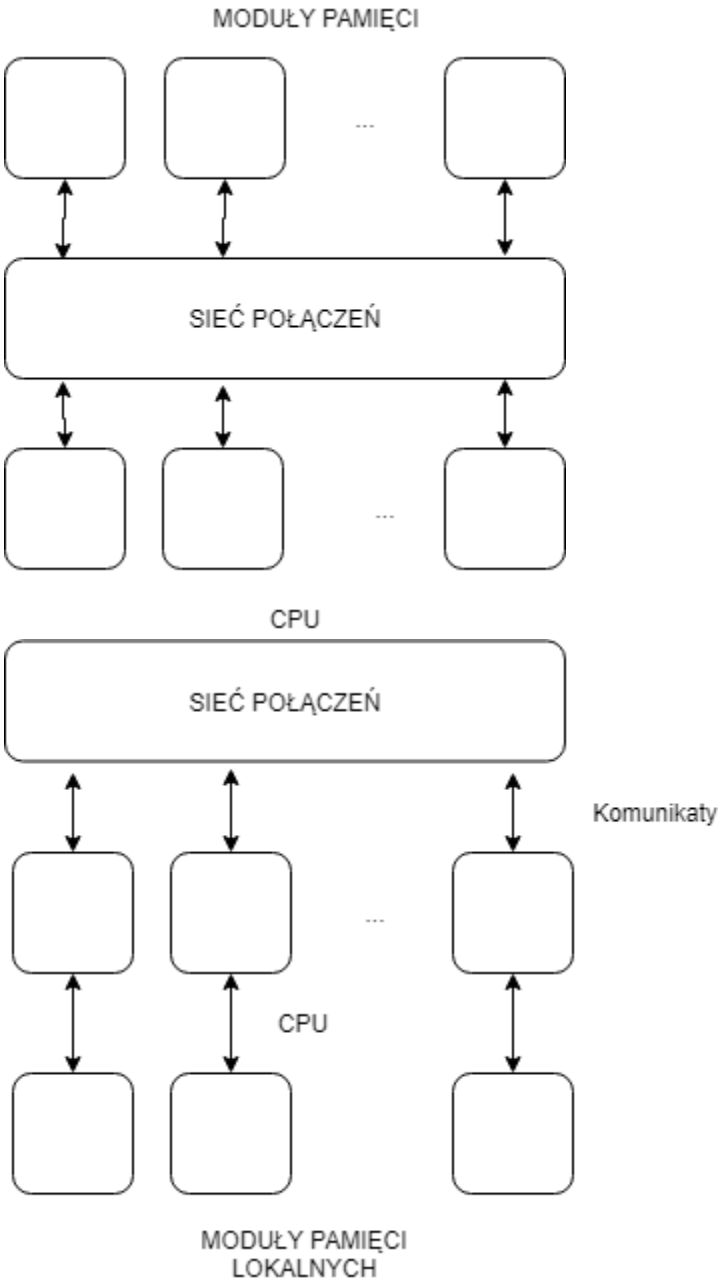
MPI CZ. 1	PRZETWARZANIE WSPÓŁBIEŻNE

1. CO TO JEST?

Message Passing Interface – protokół komunikacyjny, standard przesyłania komunikatów między procesami programów równoległych, działających na wielu lub jednym komputerze Model wykorzystywany obecnie w klastrach i superkomputerach. Implementowany w postaci bibliotek do różnych języków programowania.



MODEL ZE WSPÓLNĄ PAMIĘCIĄ (OpenMP)

Wszystkie procesory korzystają z jednej wspólnej pamięci. Każdy procesor powinien w dowolnym momencie mieć dostęp do danej komórki pamięci, w której dane mogą być ciągle aktualizowane. Pojawia się tu problem synchronizacji. Można zastosować mechanizmy takie jak np. bariery, ale oczywiście spowolnią one czas obliczeń.

MODEL Z PRZESYŁANIEM KOMUNIKATÓW (MPI)

Każda jednostka CPU posiada swoją własną pamięć. Dane nie są widoczne dla pozostałych jednostek centralnych. Dane przekazywane są do pozostałych jednostek za pomocą przesyłanych komunikatów. Ilość komunikacji powinna być ograniczona do minimum, aby zredukować narzuty.

2. WADY I ZALETY MPI

Na plus:

+ wysoka wydajność oraz możliwość efektywnej obsługi dużej liczby procesów

+ może być używany na wielu platformach, tak równoległych jak i skalarnych, bez większych zmian w sposobie działania.

+ zapewnia dobra komunikację między procesami

+ dostępny nieodpłatnie i do tego bardzo rozbudowany

+ nie obciążają procesów operacjami kopiowania pamięci

Na minus:

- złożony sposób tworzenia programów równoległych - należy rozdzielić zadania na poszczególne procesy

- brak wielowątkowości

- trzeba zadbać o poprawne rozdzielenie danych

- nie nadaje się do programów, których nie można podzielić na jednakowe podproblemy.

- program musi zostać kompletnie przebudowany pod kątem uruchamiania go przez różne procesy w ramach MPI.

- dość trudny standard w programowaniu

3. INSTALACJA (UBUNTU)

```
sudo apt-get install libcr-dev mpich mpich-doc
```

4. KOMPILACJA I URUCHOMIENIE

Kompilacja MPI

Dla plików c++: `mpic++ program.cpp –o program`

Dla plików c: `mpicc program.c –o program`

Uruchomienie MPI dla 4 wątków

```
mpirun –np 4 ./program
```

5. WYBRANE FUNKCJE

MPI_Comm_rank - pobiera identyfikator procesu.

```
int MPI_Comm_rank( MPI_Comm comm, int * rank );
```

comm	[wejście] komunikator (uchwyt).
size	[wyjście] identyfikator procesu wywołującego niniejszą funkcję.

MPI_Comm_size - Pobiera ilość dostępnych procesów

```
int MPI_Comm_size( MPI_Comm comm, int * size );
```

MPI_Init inicjalizuje mechanizm MPI

```
int MPI_Init( int * argc, char *** argv );
```

MPI_Finalize – kończy pracę w trybie MPI

```
int MPI_Finalize();
```

MPI_Bcast - Rozsyła określone dane do wszystkich procesów w grupie.

```
int MPI_Bcast( void * buffer, int count, MPI_Dataty
pe datatype, int root, MPI_Comm comm );
```

buffer	adres do bufora z danymi do wysłania/odebrania.
count	ilość elementów w buforze.
datatype	typ danych.
root	numer procesu rozsyłającego dane.
comm	komunikator.

MPI_Recv - Odbiera dane z określonego procesu.

```
int MPI_Recv( void * msg, int count, MPI_Datatype
datatype, int source, int tag, MPI_Comm
comm, MPI_Status * status );
```

msg	adres bufora do którego mają zostać zapisane dane.
count	ilość elementów w buforze.
datatype	typ danych bufora.
source	numer procesu wysyłającego.
tag	identyfikator wiadomości.
comm	komunikator.
status	zwraca status wykonanej operacji, np. MPI_STATUS_IGNORE

MPI_Send - Wysyła dane do określonego procesu

```
int MPI_Send( void * msg, int count, MPI_Datatype
datatype, int dest, int tag, MPI_Comm comm );
```

msg	adres do bufora z danymi do wysłania.
count	ilość elementów w buforze.
datatype	typ danych.
dest	numer procesu odbierającego.
tag	identyfikator wiadomości.
comm	komunikator.

MPI_Barierr - Wstrzymuje pracę procesu aż do wywołania tej funkcji przez wszystkie procesy należące do grupy

```
int MPI_Barrier( MPI_Comm comm );
```

KOMUNIKACJA GRUPOWA

MPI_Reduce - Obliczanie równoległe. Redukcja wszyscy do jednego. Wynik operacji trafia do jednego, wybranego procesu.

```
int MPI_Reduce( void * sendbuf, void * recvbuf, int
 count, MPI_Datatype datatype, MPI_Op
op, int root, MPI_Comm comm );
```

sendbuff	adres do bufora z danymi.
recvbuff	adres na bufor w którym mają znaleźć się wyniki obliczeń.
count	ilość elementów w buforze.
datatype	typ danych.
op	działanie do wykonania.
root	numer procesu rozsyłającego dane.
comm	komunikator.

MPI_Gather - zbieranie wszyscy do jednego,
MPI_Scatter - rozpraszanie jeden do wszystkich,

KOMUNIKATOR/ UCHWYT:

MPI_COMM - struktura przechowująca atrybuty komunikatora
MPI_COMM_WORLD - domyślny i podstawowy komunikator w MPI
MPI_COMM_RANK - nr procesu
MPI_COMM_SIZE - ilość procesów w komunikatorze (dodatkowo przedstawia wielkość danego procesu)

STAŁE PREDEFINIOWANE

MPI_Datatype

MPI_CHAR
MPI_UNSIGNED_CHAR
MPI_BYTE
MPI_SHORT

MPI_UNSIGNED_SHORT
MPI_INT
MPI_UNSIGNED
MPI_LONG
MPI_UNSIGNED_LONG
MPI_FLOAT
MPI_DOUBLE
MPI_LONG_DOUBLE
MPI_PACKED

MPI_Op Metoda redukowania danych.

MPI_Comm Typ wyliczeniowy określający rodzaj komunikacji.

```
#include<mpi.h>
enum MPI_Comm
{
    MPI_COMM_WORLD,
    MPI_COMM_SELF
}; //enum MPI_Comm
```

Przetestuj powyższe funkcje w swoim pierwszym programie implementującym MPI.
https://bitbucket.org/adrianwii/pw/src/master/MPI1/start.c