



Uniwersytet  
Wrocławski

# MPI

(Message Passing Interface)

Zbigniew Koza  
Wydział Fizyki i Astronomii

# MPI

- MPI = Message Passing Interface
- Biblioteka i środowisko do programowania aplikacji współbieżnych w środowisku z pamięcią rozproszoną (***distributed memory***)



# Najprostszy program

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

int main(int argc, char* argv[])
{
    MPI_Init(&argc, &argv);
    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    std::cout << "Hello! from process nr " << rank << "\n";
    MPI_Finalize();
}
```

# Najprostszy program



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

int main(int argc, char* argv[])
{
    MPI_Init(&argc, &argv);
    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    std::cout << "Hello! from process nr " << rank << "\n";
    MPI_Finalize();
}
```

# Instalacja

Zainstaluj jedną z bibliotek:

- openmpi
- mpich2

Stan	Nazwa	Wersja	Repozyt	Rozmiar
	 <b>openmpi</b> High performance message passing library (MPI)	3.1.3-1	extra	11,5 MB

# Kompilacja

- > **mpicxx** prog.cpp -O2 ...
- > **mpic++** prog.cpp -O2 ...
- > **mpicc** prog.c -O2 ...
- > **mpif90** prog.f90


To są tzw. wrappery dla C++/C/F90

```
> mpicxx --show  
g++ -pthread -Wl,-rpath -Wl,/usr/lib/openmpi -Wl,--enable-new-dtags -L/usr/lib/openmpi -lmpi_cxx -lmpi
```

# Uruchomienie

> mpirun -np 4 ./a.exe

> mpirun -np 10 --hosts master,slave1,slave2 ./a.exe



**10  
procesów**



**3 maszyny  
(dobrze skonfigurowane)**

> mpirun -np 8 --hostfile host\_file ./a.exe



**plik  
konfiguracyjny**

# Konfiguracja serwerów

- Wspólny dysk sieciowy

```
zkoza@zwei ~ $ cat /etc/fstab
# /etc/fstab: static file system information.
```

■ ■ ■

/dev/sda1	/boot	ext2	noauto,noatime	1	2
/dev/sda2	/	ext3	noatime	0	1
/dev/sda3	none	swap	sw	0	0
/dev/sda5	/data	ext4	noatime	0	3
<u>zero:/home</u>	<u>/home</u>	<u>nfs4</u>	rw,quota	0	0



# Konfiguracja serwerów

- Konfiguracja **ssh**
  - `ssh-keygen`
  - `ssh-copy-id -i inny_serwer`
  - `ssh inny_serwer`
- Testy:
  - `scp inny_serwer:zdalne_miejsce plik_lokalny`
  - `scp plik_lokalny inny_serwer:zdalne_miejsce`
  - `ssh inny_serwer`
  - `ssh inny_serwer komenda`
  - Pamiętaj o uruchomieniu serwisu `sshd`

# Jak to działa?

- Ten sam program uruchamia się jako **niezależne procesy** jednocześnie na tej samej lub różnych maszynach
- Procesy **synchronizują** się poprzez tzw. komunikaty MPI
- Nie ma wspólnej pamięci
  - MPI funkcjonuje w modelu **pamięci rozproszonej**

Prosty przykład

# MPI\_Send/MPI\_Recv

```
MPI_Init(NULL, NULL);
int world_rank, world_size; // id procesu; liczba procesów
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
MPI_Comm_size(MPI_COMM_WORLD, &world_size);

if (world_size < 2) // Muszą istnieć co najmniej dwa procesy
{
    fprintf(stderr, "World size must be greater than 1 for %s\n", argv[0]);
    MPI_Abort(MPI_COMM_WORLD, 1);
}

int number;
if (world_rank == 0)
{
    // Jeżeli proces ma rank 0, to wysyła liczbę -1 do procesu 1
    number = -1;
    printf("Process 0 is sending an int of value %d\n", number);
    MPI_Send(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
}
else if (world_rank == 1)
{
    MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    printf("Process 1 received number %d from process 0\n", number);
}
MPI_Finalize();
```


# Inicjalizacja / finalizacja

```
MPI_Init(NULL, NULL);
int world_rank, world_size; // id procesu; liczba procesów
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
MPI_Comm_size(MPI_COMM_WORLD, &world_size);

if (world_size < 2) // Muszą istnieć co najmniej dwa procesy
{
    fprintf(stderr, "World size must be greater than 1 for %s\n", argv[0]);
    MPI_Abort(MPI_COMM_WORLD, 1);
}

int number;
if (world_rank == 0)
{
    // Jeżeli proces ma rank 0, to wysyła liczbę -1 do procesu 1
    number = -1;
    printf("Process 0 is sending an int of value %d\n", number);
    MPI_Send(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
}
else if (world_rank == 1)
{
    MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    printf("Process 1 received number %d from process 0\n", number);
}
MPI_Finalize();
```

# Kim jestem? / Ilu nas jest?



```
MPI_Init(NULL, NULL);
int world_rank, world_size; // id procesu; liczba procesów
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
MPI_Comm_size(MPI_COMM_WORLD, &world_size);

if (world_size < 2) // Muszą istnieć co najmniej dwa procesy
{
    fprintf(stderr, "World size must be greater than 1 for %s\n", argv[0]);
    MPI_Abort(MPI_COMM_WORLD, 1);
}

int number;
if (world_rank == 0)
{
    // Jeżeli proces ma rank 0, to wysyła liczbę -1 do procesu 1
    number = -1;
    printf("Process 0 is sending an int of value %d\n", number);
    MPI_Send(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
}
else if (world_rank == 1)
{
    MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    printf("Process 1 received number %d from process 0\n", number);
}
MPI_Finalize();
```

# Czy nie jestem sam?

```
MPI_Init(NULL, NULL);
int world_rank, world_size; // id procesu; liczba procesów
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
MPI_Comm_size(MPI_COMM_WORLD, &world_size);

if (world_size < 2) // Muszą istnieć co najmniej dwa procesy
{
    fprintf(stderr, "World size must be greater than 1 for %s\n", argv[0]);
    MPI_Abort(MPI_COMM_WORLD, 1);
}

int number;
if (world_rank == 0)
{
    // Jeżeli proces ma rank 0, to wysyła liczbę -1 do procesu 1
    number = -1;
    printf("Process 0 is sending an int of value %d\n", number);
    MPI_Send(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
}
else if (world_rank == 1)
{
    MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    printf("Process 1 received number %d from process 0\n", number);
}
MPI_Finalize();
```

# Proces nr 0 („master”) wysyła dane

```
MPI_Init(NULL, NULL);
int world_rank, world_size; // id procesu; liczba procesów
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
MPI_Comm_size(MPI_COMM_WORLD, &world_size);

if (world_size < 2) // Muszą istnieć co najmniej dwa procesy
{
    fprintf(stderr, "World size must be greater than 1 for %s\n", argv[0]);
    MPI_Abort(MPI_COMM_WORLD, 1);
}

int number;
if (world_rank == 0)
{
    // Jeżeli proces ma rank 0, to wysyła liczbę -1 do procesu 1
    number = -1;
    printf("Process 0 is sending an int of value %d\n", number);
    MPI_Send(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
}
else if (world_rank == 1)
{
    MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    printf("Process 1 received number %d from process 0\n", number);
}
MPI_Finalize();
```



# Proces nr 1 („slave”) odbiera dane

```
MPI_Init(NULL, NULL);
int world_rank, world_size; // id procesu; liczba procesów
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
MPI_Comm_size(MPI_COMM_WORLD, &world_size);

if (world_size < 2) // Muszą istnieć co najmniej dwa procesy
{
    fprintf(stderr, "World size must be greater than 1 for %s\n", argv[0]);
    MPI_Abort(MPI_COMM_WORLD, 1);
}

int number;
if (world_rank == 0)
{
    // Jeżeli proces ma rank 0, to wysyła liczbę -1 do procesu 1
    number = -1;
    printf("Process 0 is sending an int of value %d\n", number);
    MPI_Send(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
}
else if (world_rank == 1)
{
    MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    printf("Process 1 received number %d from process 0\n", number);
}
MPI_Finalize();
```

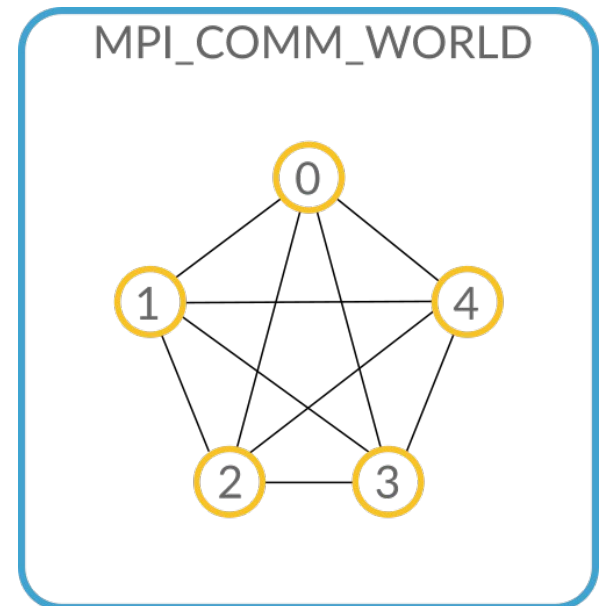
# MPI\_COMM\_WORLD

- Tzw. komunikator
- Argument niemal każdej funkcji MPI
- Oznacza **wszystkie uruchomione procesy**

```
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);  
MPI_Comm_size(MPI_COMM_WORLD, &world_size);
```

```
MPI_Abort(MPI_COMM_WORLD, 1);
```

```
MPI_Send(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
```



# MPI\_Send

```
MPI_Send(&number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
```

- 
- 1) Gdzie są dane? → `&number`
- 2) Ile ich jest? → `1`
- 3) Jakiego są typu? → `MPI_INT`
- 4) Komu je wysłać? → `1`
- 5) Jak je identyfikujemy? → `0`
- 6) W której grupie procesów? → `MPI_COMM_WORLD`

```
int MPI_Send(const void* buf,  
             int count,  
             MPI_Datatype datatype,  
             int dest,  
             int tag,  
             MPI_Comm comm)
```

# MPI\_Recv

```
MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
```

- 1) Gdzie umieścić dane?
- 2) Ile ich jest?
- 3) Jakiego są typu?
- 4) Kto je ma wysłać?
- 5) Jaki mają mieć identyfikator?
- 6) W której grupie procesów?
- 7) Jak operacja się zakończyła?

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

# MPI\_Send/MPI\_Recv

- MPI\_Send i MPI\_Recv są **operacjami blokującymi**

# MPI\_Isend/MPI\_Irecv

- Istnieją analogiczne funkcje nieblokujące, **MPI\_Isend, MPI\_Irecv**
- Używa się ich z funkcjami **MPI\_Test** lub **MPI\_Wait**

# Wyścig, zakleszczenie

- **Wyścig** w MPI łatwo wygenerować, stosując operacje nieblokujące
- **Zakleszczenie** można uzyskać niemal każdą funkcją MPI – to nieodłączna cecha programowania współbieżnego.

# Inne popularne funkcje MPI

- **MPI\_Probe** – czeka na komunikat, bez wczytania danych
- **MPI\_Barrier** – zakłada barierę
- **MPI\_Bcast** – przesyła dane do wszystkich procesów
- **MPI\_Scatter** – rozrzuca dane
- **MPI\_Gather** – zbiera rozrzucone dane
- **MPI\_Reduce** – „redukcja” jak w OpenMP
- **MPI\_Test, MPI\_Wait** – synchronizacja operacji nieblokujących



# Dalsza lektura

- <http://mpitutorial.com/tutorials/>
- <https://www.codingame.com/playgrounds/349/>
- ...