



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***)

<https://wcss.pl/images/image-01.jpg>



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

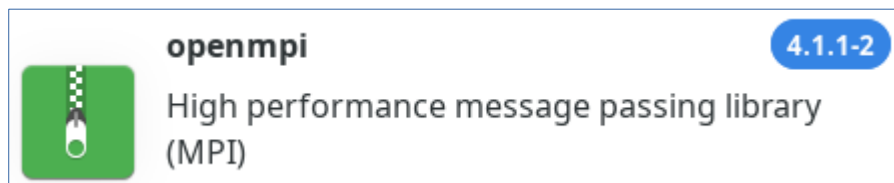
```
#include <iostream>
#include <mpi.h>

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();
}
```

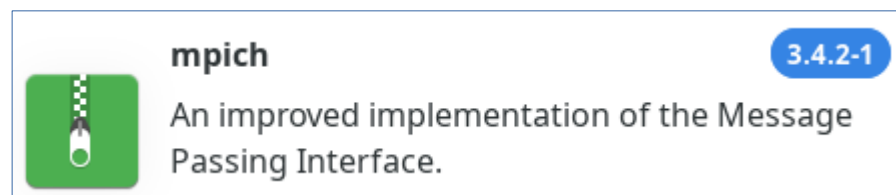
Instalacja

Zainstaluj jedną z bibliotek, np:

- Open MPI



- MPICH



Kompilacja

- > **mpicxx** prog.cpp -O2 ... C++
- > **mpic++** prog.cpp -O2 ... C++
- > **mpicc** prog.c -O2 ... C
- > **mpif90** prog.f90 FORTRAN 90


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 (ewentualnym) 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**

Jak to działa?

Uruchom n kopii programu; na każdej z nich:

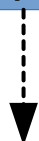
Wypełnij bufor danych



Pobierz swój id w grupie n procesów



Jeżeli (`my_id == 0`)
 Wyślij dane do procesu o `id == 1`
Jeżeli (`my_id == 1`)
 Odbierz dane z procesu o `id == 0`



Zakończ pracę w trybie MPI

Wypełnij bufor danych



Pobierz swój id w grupie n procesów



Jeżeli (`my_id == 0`)
 Wyślij dane do procesu o `id == 1`
Jeżeli (`my_id == 1`)
 Odbierz dane z procesu o `id == 0`



Zakończ pracę w trybie MPI

Uruchom n kopii programu; na każdej z nich:

Wypełnij bufor danych

Wypełnij bufor danych

Pobierz swój id w grupie n procesów

Pobierz swój id w grupie n procesów

Jeżeli (`my_id == 0`)
 Wyślij dane do procesu o `id == 1`
Jeżeli (`my_id == 1`)
 Otwórz bufor na dane z `id == 0`
 Rób_coś()
Jeżeli (`my_id == 0`)
 Poczekaj na koniec transmisji do 1
Jeżeli (`my_id == 1`)
 Poczekaj na koniec transmisji z 0

Jeżeli (`my_id == 0`)
 Wyślij dane do procesu o `id == 1`
Jeżeli (`my_id == 1`)
 Otwórz bufor na dane z `id == 0`
 Rób_coś()
Jeżeli (`my_id == 0`)
 Poczekaj na koniec transmisji do 1
Jeżeli (`my_id == 1`)
 Poczekaj na koniec transmisji z 0

Zakończ pracę w trybie MPI

Zakończ pracę w trybie MPI

Uruchom n kopii programu; na każdej z nich:

Pobierz swój id w grupie n procesów

Pobierz swój id w grupie n procesów

Oblicz $f(id)$ i wynik zapisz w k

Oblicz $f(id)$ i wynik zapisz w k

Jeżeli ($my_id == 0$)
Odbierz k od każdego procesu
Zapisz ich sumę w swoim k
W przeciwnym wypadku
Wyślij k do procesu o $id == 0$

Jeżeli ($my_id == 0$)
Odbierz k od każdego procesu
Zapisz ich sumę w swoim k
W przeciwnym wypadku
Wyślij k do procesu o $id == 0$

Zakończ pracę w trybie MPI

Zakończ pracę w trybie MPI

Uruchom n kopii programu; na każdej z nich:

Pobierz swój id w grupie n procesów

Pobierz swój id w grupie n procesów

Wypełnij swój bufor (id)

Wypełnij swój bufor (id)

Jeżeli ($\text{my_id} == 0$)
Odbierz bufor od każdego procesu
Dane zapisz w swoim n-razy
większym buforze
W przeciwnym wypadku
Wyślij bufor do procesu o $\text{id} == 0$

Jeżeli ($\text{my_id} == 0$)
Odbierz bufor od każdego procesu
Dane zapisz w swoim n-razy
większym buforze
W przeciwnym wypadku
Wyślij bufor do procesu o $\text{id} == 0$

Zakończ pracę w trybie MPI

Zakończ pracę w trybie MPI

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();
```

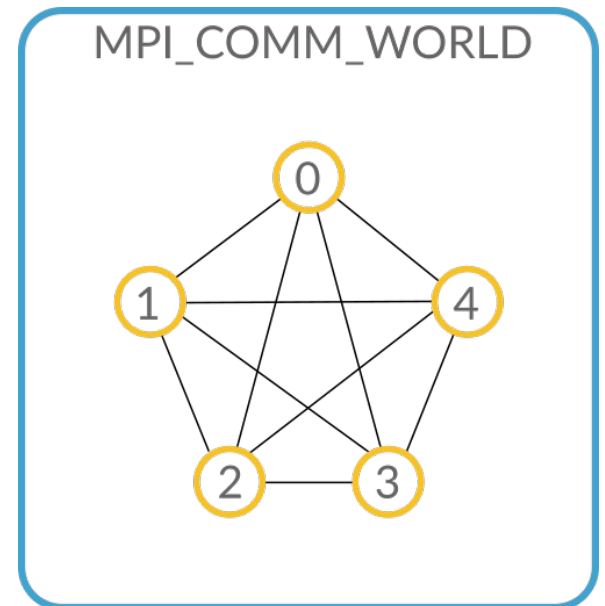
MPI_COMM_WORLD

- 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

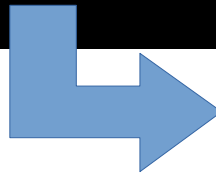
- MPI_Send i MPI_Recv są **operacjami blokującymi**

MPI_Isend/MPI_Irecv

- Istnieją analogiczne operacje **nieblokujące**, **MPI_Isend, MPI_Irecv**
- Używa się ich z funkcjami **MPI_Test** lub **MPI_Wait, MPI_Waitall**
- Przesyłanie danych odbywa się w tle, program może zajmować się czymś innym

Przykład

```
MPI_Request requests[2];
double k = 5* my_rank;
std::array<double,5> a = {k , 1+ k, 2 + k, 3 + k, 4 + k}, b = {0, 0, 0, 0, 0};
if (my_rank == 0)
{
    MPI_Irecv(&b[0], 5, MPI_DOUBLE, 1, 1, MPI_COMM_WORLD, &requests[0]);
    MPI_Isend(&a[0], 5, MPI_DOUBLE, 1, 2, MPI_COMM_WORLD, &requests[1]);
}
else if (my_rank == 1)
{
    MPI_Irecv(&b[0], 5, MPI_DOUBLE, 0, 2, MPI_COMM_WORLD, &requests[0]);
    MPI_Isend(&a[0], 5, MPI_DOUBLE, 0, 1, MPI_COMM_WORLD, &requests[1]);
}
print(my_rank, a, "a =");
print(my_rank, b, "b =");
std::cout << my_rank << " is waiting\n";
MPI_Waitall(2, requests, MPI_STATUSES_IGNORE);
print(my_rank, a, "a =");
print(my_rank, b, "b =");
```



```
my_rank: 0; a = 0 1 2 3 4
my_rank: 0; b = 0 0 0 0 0
0 is waiting
my_rank: 1; a = 5 6 7 8 9
my_rank: 1; b = 0 0 0 0 0
1 is waiting
my_rank: 0; a = 0 1 2 3 4
my_rank: 0; b = 5 6 7 8 9
my_rank: 1; a = 5 6 7 8 9
my_rank: 1; b = 0 1 2 3 4
```

Wyścig, zakleszczenie

- **Wyścig** (*race*) w MPI można uzyskać coś w rodzaju wyścigu, przedwcześnie używając buforów operacji nieblokujących
- **Zakleszczenie** (*deadlock*) można uzyskać niemal każdą funkcją MPI – to nieodłączna cecha programowania współbieżnego

Wyścig - przykład


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

int main()
{
    int rank;
    int sendbuf = 12345;
    int recvbuf = 0;
    MPI_Request request;

    MPI_Init(nullptr, nullptr);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    if (rank == 0)
    {
        MPI_Isend(&sendbuf, 1, MPI_INT, 1, 0, MPI_COMM_WORLD, &request);
    }
    if (rank == 1)
    {
        MPI_Irecv(&recvbuf, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &request);
    }
    std::this_thread::sleep_for(std::chrono::seconds(2));
    std::cout << "rank = " << rank << ", sendbuf = " << sendbuf
                << ", recvbuf = " << recvbuf << "\n";

    MPI_Finalize();
}
```

- Nie wiadomo, kiedy recvbuf zostanie zapisany



```
rank = 1, sendbuf = 12345, recvbuf = 0
rank = 0, sendbuf = 12345, recvbuf = 0
```

MPI_Test

```
std::this_thread::sleep_for(std::chrono::seconds(2));  
int result = 0;  
MPI_Test( &request, &result, MPI_STATUSES_IGNORE);  
std::cout << "rank = " << rank << ", sendbuf = " << sendbuf  
          << ", recvbuf = " << recvbuf << ", result = " << result << "\n";
```



```
rank = 1, sendbuf = 12345, recvbuf = 12345, result = 1  
rank = 0, sendbuf = 12345, recvbuf = 0, result = 1
```

- Funkcja nieblokująca
- Result != 0 oznacza, że MPI_Isend / MPI_Irecv zakończyła się powodzeniem

MPI_Wait

```
std::this_thread::sleep_for(std::chrono::seconds(2));  
MPI_Wait(&request, MPI_STATUSES_IGNORE);  
std::cout << "rank = " << rank << ", sendbuf = " << sendbuf  
          << ", recvbuf = " << recvbuf << "\n";
```



```
rank = 0, sendbuf = 12345, recvbuf = 0  
rank = 1, sendbuf = 12345, recvbuf = 12345
```

- Funkcja blokująca
- MPI_Isend / MPI_Irecv
musisz zakończyć funkcję
MPI_Wait (MPI_Waitall) lub MPI_Test

Zakleszczenie - przykład

```
int number = 189;
if (world_rank == 0)
{
    // Proces 0 wysyła liczbę -1 do procesu 1 (ale tag == 1 - deadlock!)
    number = -1;
    printf("Process 0 is sending an int of value %d\n", number);
    MPI_Send(&number, 1, MPI_INT, 1, 1, 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);
}
```

- 0 wysyła komunikat nr 1
- 1 oczekuje na komunikat nr 0
- Efekt: zakleszczenie

MPI_Bcast

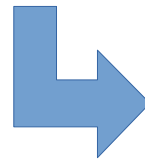
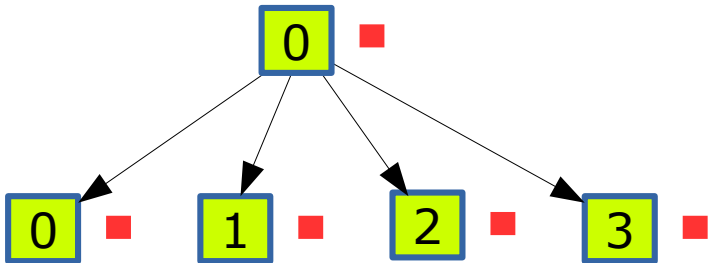
```
int main()
{
    MPI_Init(nullptr, nullptr);

    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    int sendbuf = 1000 + rank;
    std::cout << "rank = " << rank << ", sendbuf = " << sendbuf << "\n";
    MPI_Bcast(&sendbuf, 1, MPI_INT, 0, MPI_COMM_WORLD);
    std::cout << "rank = " << rank << ", sendbuf = " << sendbuf << "\n";

    MPI_Finalize();
}
```

Broadcast = „rozgłaszanie”



rank = 0,	sendbuf = 1000
rank = 0,	sendbuf = 1000
rank = 1,	sendbuf = 1001
rank = 1,	sendbuf = 1000
rank = 3,	sendbuf = 1003
rank = 3,	sendbuf = 1000
rank = 2,	sendbuf = 1002
rank = 2,	sendbuf = 1000

MPI_Barrier

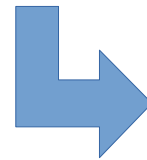
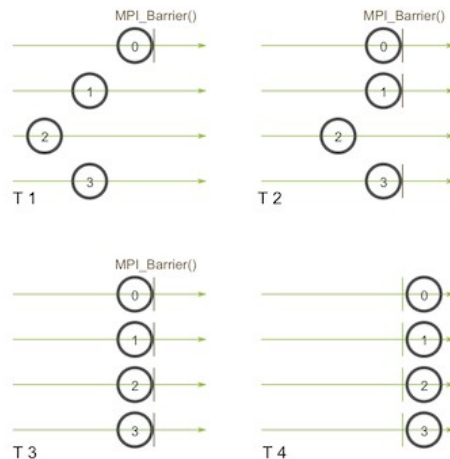
```
MPI_Init(nullptr, nullptr);

int rank;
MPI_Comm_rank(MPI_COMM_WORLD, &rank);

int sendbuf = 1000 + rank;
std::cout << "rank = " << rank << ", sendbuf = " << sendbuf << "\n";
MPI_Barrier(MPI_COMM_WORLD);
MPI_Bcast(&sendbuf, 1, MPI_INT, 0, MPI_COMM_WORLD);
std::cout << "rank = " << rank << ", sendbuf = " << sendbuf << "\n";

MPI_Finalize();
```

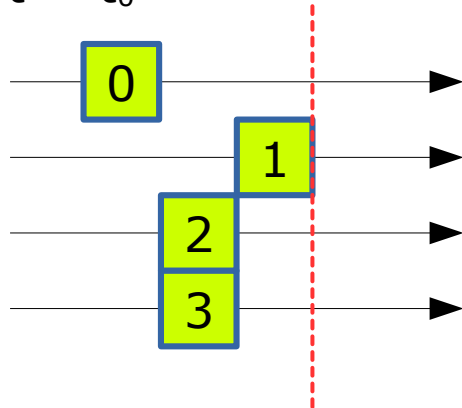
Bariera



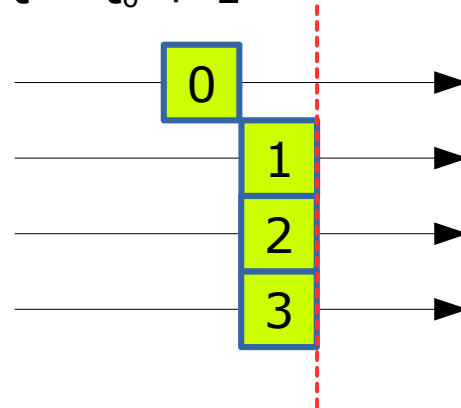
```
rank = 1, sendbuf = 1001
rank = 2, sendbuf = 1002
rank = 0, sendbuf = 1000
rank = 3, sendbuf = 1003
rank = 1, sendbuf = 1000
rank = 0, sendbuf = 1000
rank = 2, sendbuf = 1000
rank = 3, sendbuf = 1000
```

Bariera

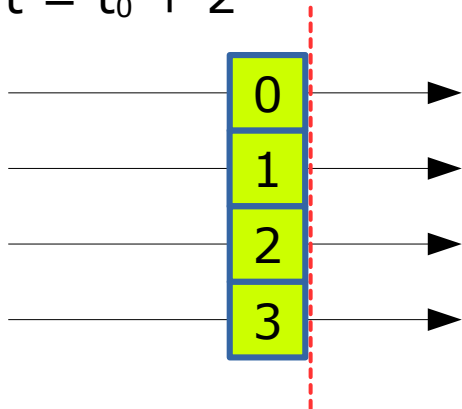
$t = t_0$



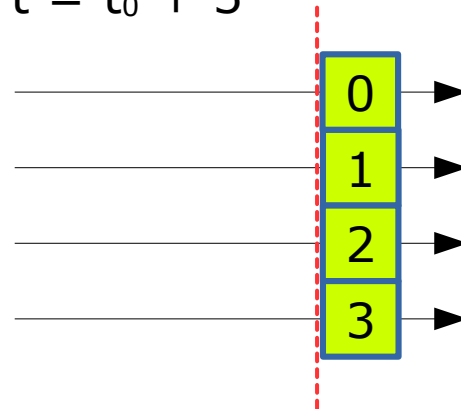
$t = t_0 + 1$



$t = t_0 + 2$



$t = t_0 + 3$



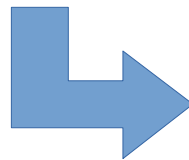
MPI_Scatter

```
MPI_Init(nullptr, nullptr);

int rank, world_size;
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &world_size);

std::vector<int> buffer;
if (rank == 0)
{
    buffer.resize(world_size*2);
    std::iota(begin(buffer), end(buffer), 0);
}
int recvd[2] = {-1, -1};
MPI_Scatter(buffer.data(), 2, MPI_INT, recvd, 2, MPI_INT, 0, MPI_COMM_WORLD);
std::cout << "rank = " << rank
          << ", received = " << recvd[0] << ", " << recvd[1] << "\n";

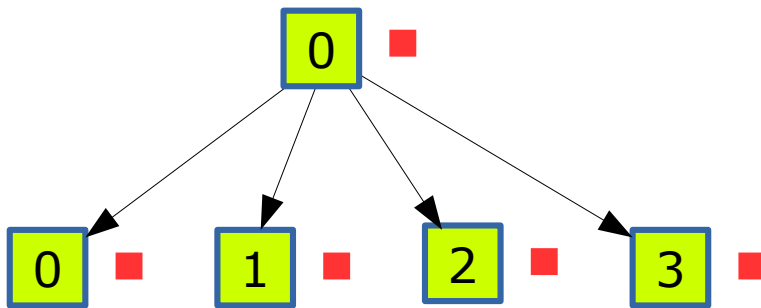
MPI_Finalize();
```



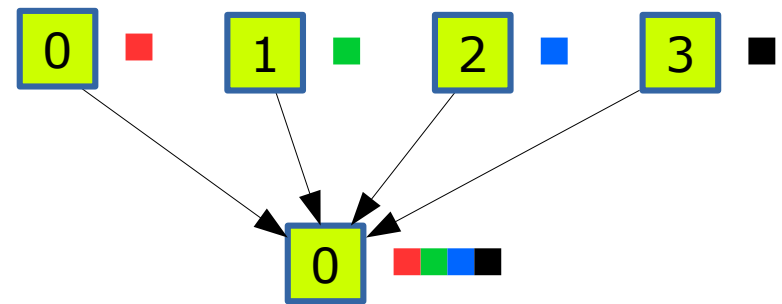
```
rank = 0, received = 0, 1
rank = 1, received = 2, 3
```

Broadcast, Scatter, Gather, Allgather

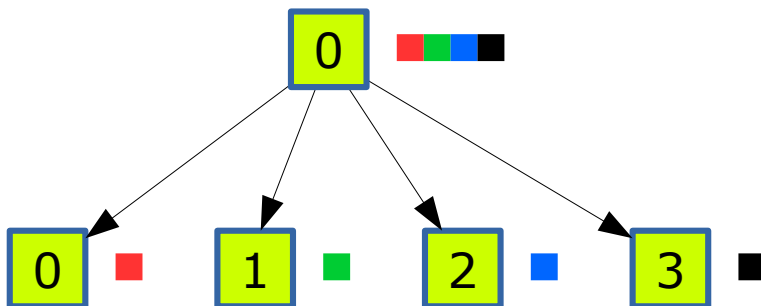
MPI_Bcast



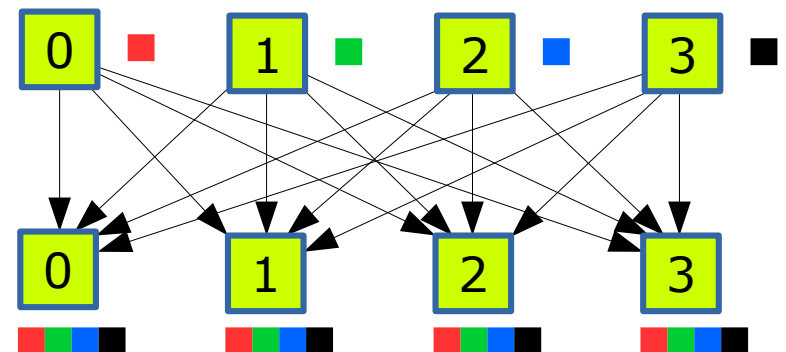
MPI_Gather



MPI_Scatter

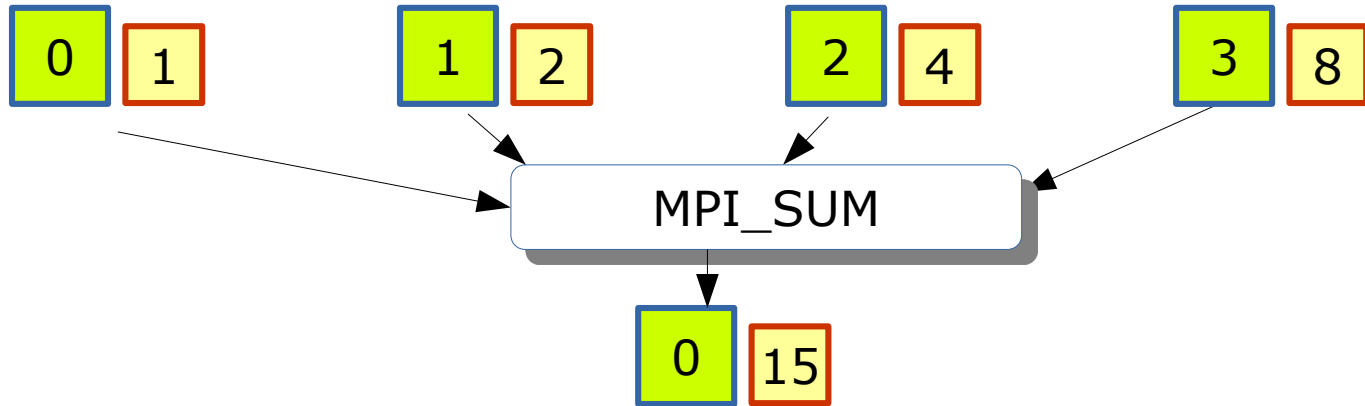


MPI_Allgather

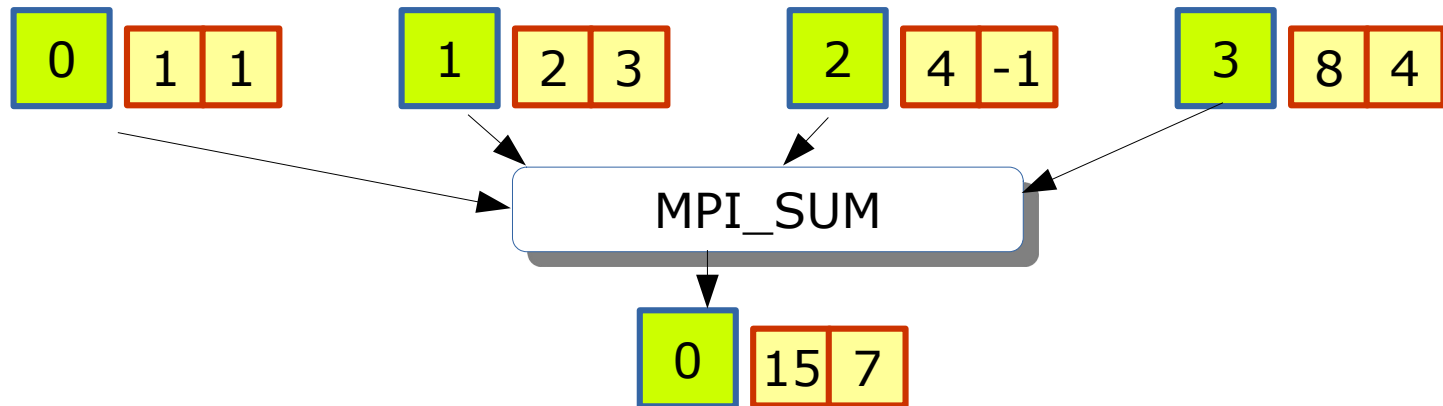


Reduce

MPI_Reduce

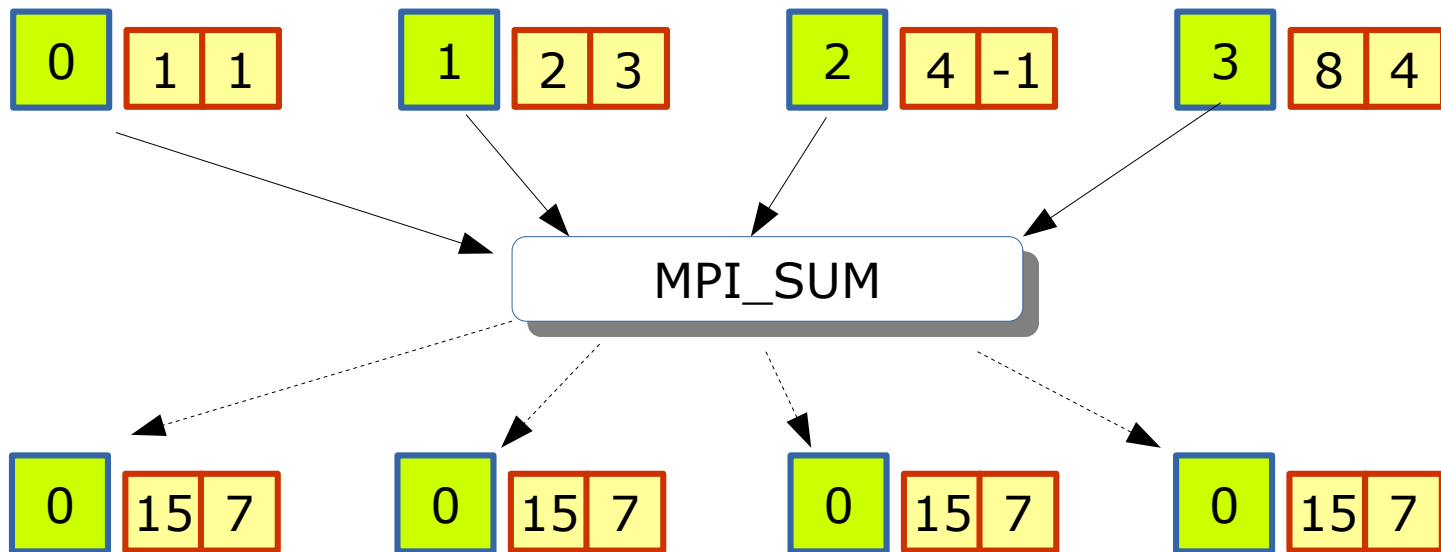


MPI_Reduce



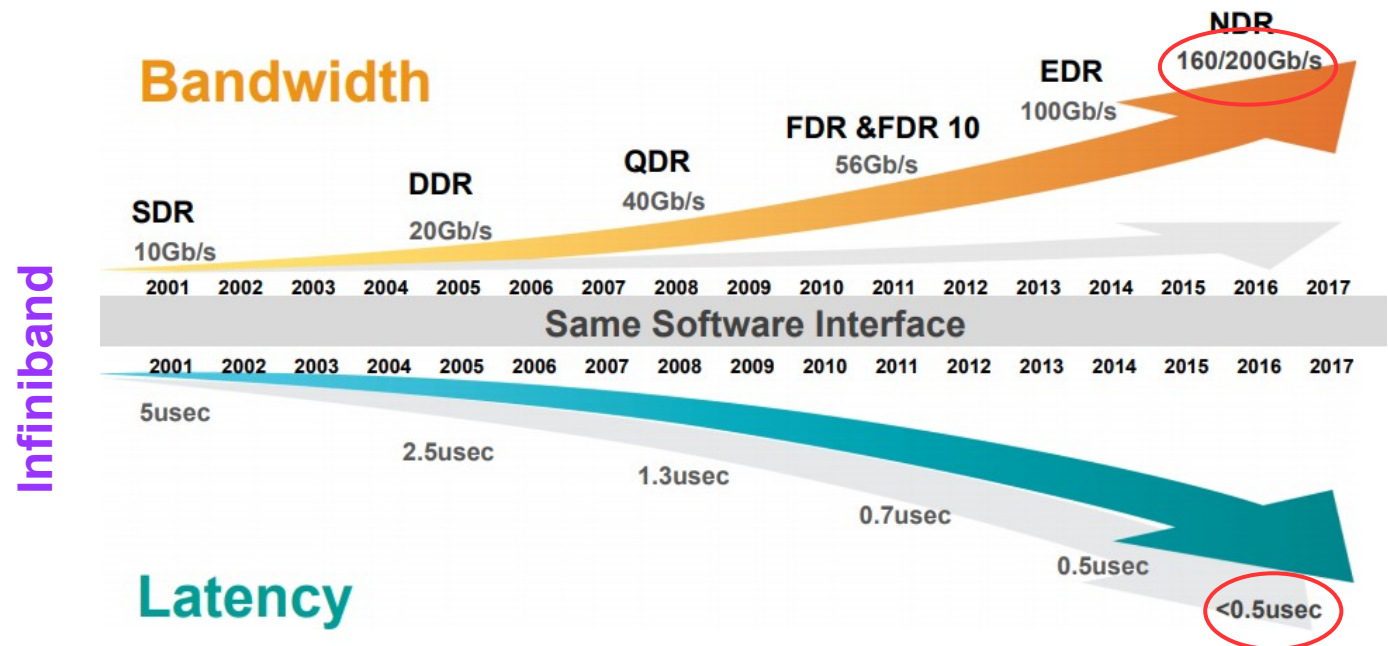
Allreduce

MPI_Allreduce

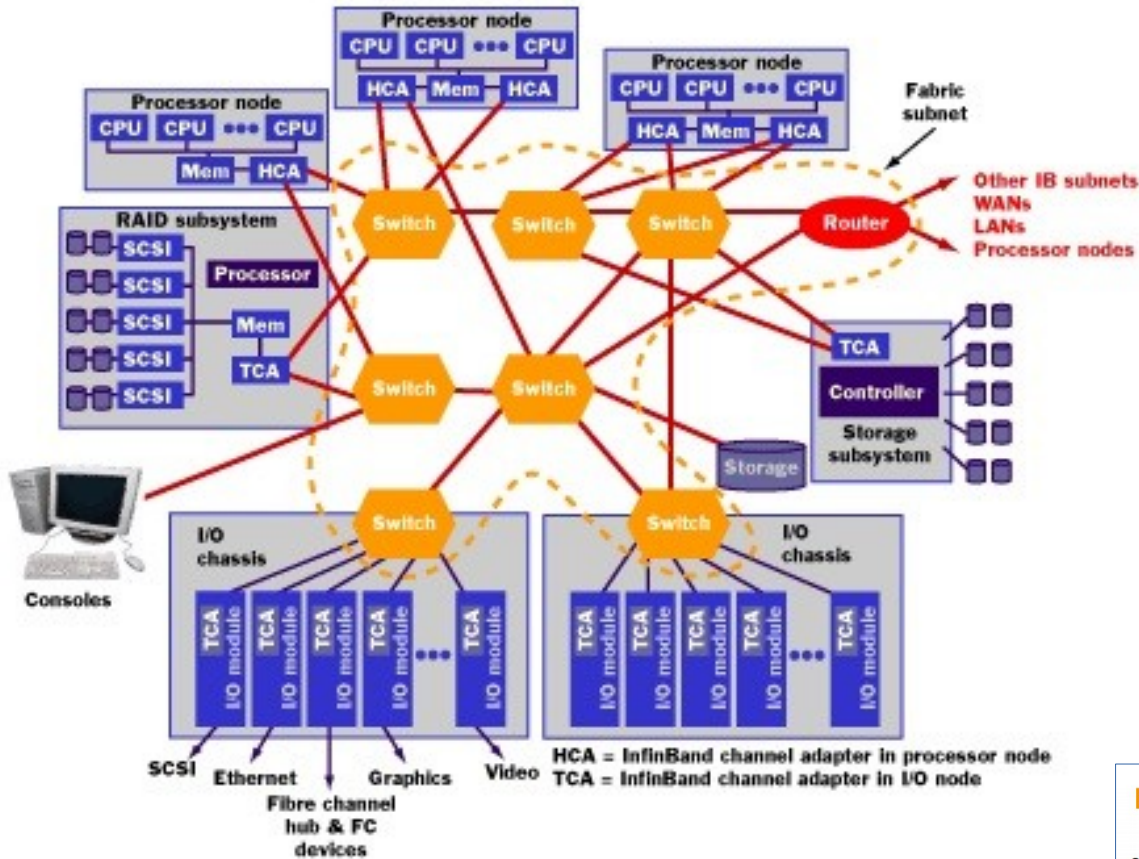


Latency i bandwidth

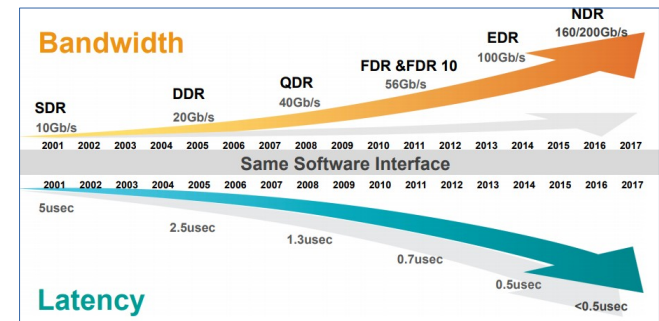
- Latencja: czas potrzebny do nawiązania połączenia i otwarcia kanału komunikacyjnego
- Przepustowość: prędkość transferu
 - Komunikaty MPI mogą być kosztowne



Latency i bandwidth

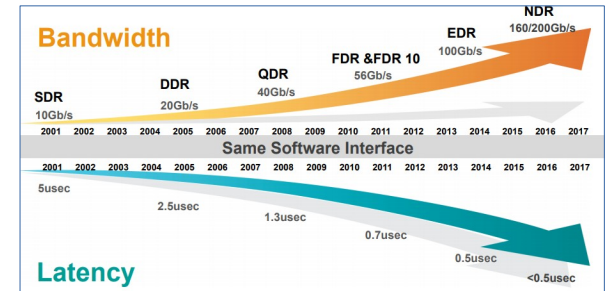


Infiniband



Latency i bandwidth

- Jeśli latencja = $1\mu\text{s}$
a przepustowość = 20GB/s , to
 - Przesłanie 1 bajta zajmuje $1\mu\text{s}$
 - Przesłanie 1 KB zajmuje $1.05\mu\text{s}$
 - Przesłanie 1 MB zajmuje $51\mu\text{s}$
 - Przesłanie 1 GB zajmuje 50ms
- } latencja
- } przepustowość



OpenMP vs MPI

OpenMP

- Wbudowane w kompilator
- Dyrektywy, biblioteka, zmienne środowisk.
- Pamięć współdzielona
- Wątki
- Fork & join

MPI

- Biblioteka, (OpenMPI, MPICH,...)
- Biblioteka
- Pamięć rozproszona
- Procesy
- Procesy są tworzone raz

OpenMP vs MPI

OpenMP

- Prosta kompilacja
(-fopenmp)
- Programy uruchamiane „normalnie”

MPI

- Nakładka na kompilator
(mpic++, etc.)
- Specjalny program uruchomieniowy
(mpirun)

OpenMP vs MPI

OpenMP

- Pamięć współdzielona
- Dane: prywatne i współdzielone
- Możliwy wyścig => muteksy, sekcje krytyczne, operacje atomowe...

MPI

- Komunikaty (narzut)
- Dane: prywatne
- nie potrzeba muteksów etc.

Dalsza lektura

- <https://mpitutorial.com/>
- <https://www.open-mpi.org/doc/current/>
- <https://www.codingame.com/playgrounds/349/>

