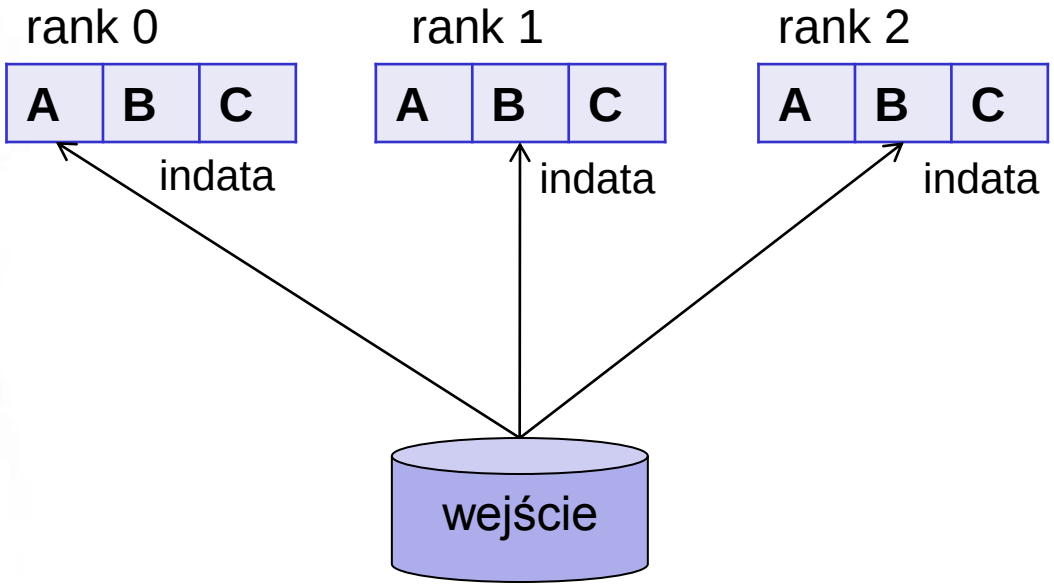
Programowanie współbieżne... (12)

Andrzej Baran 2010/11

LINK: http://kft.umcs.lublin.pl/baran/prir/index.html

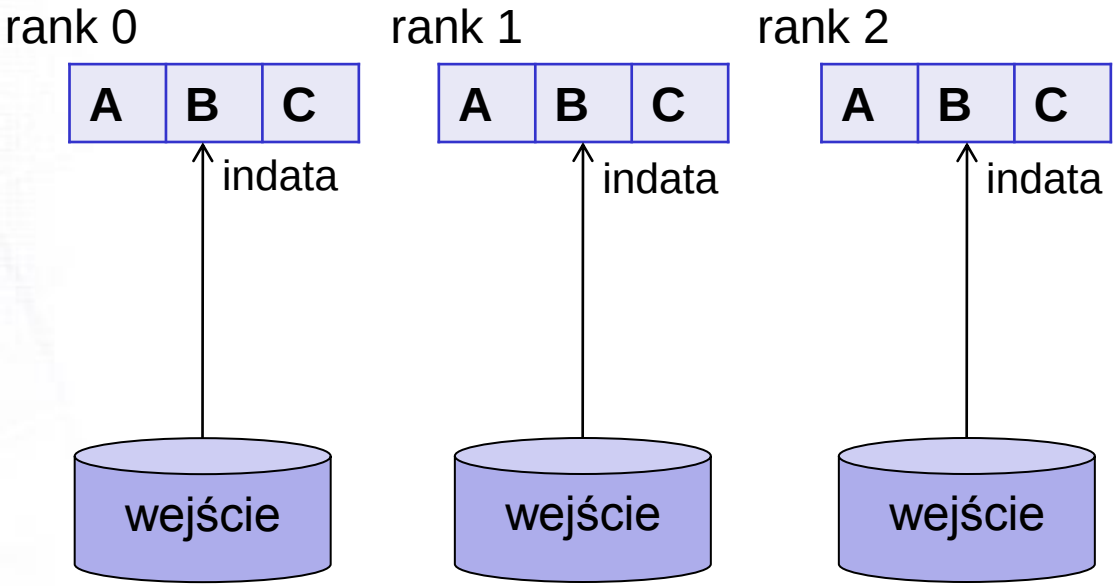
Dekompozycja danych. Wejście.



Wejście wspólne.

```
...
READ(*) indata
...
```

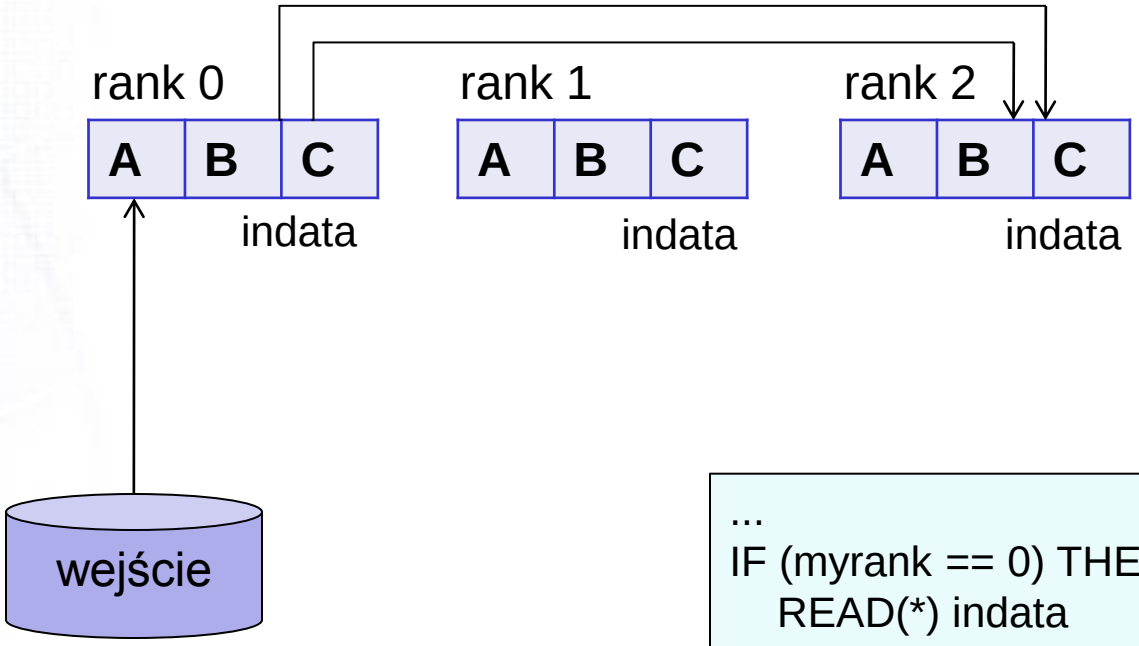
Dekompozycja danych. Wejście.



Wejście lokalne; rozdzielne

```
...
READ(*) indata
...
```

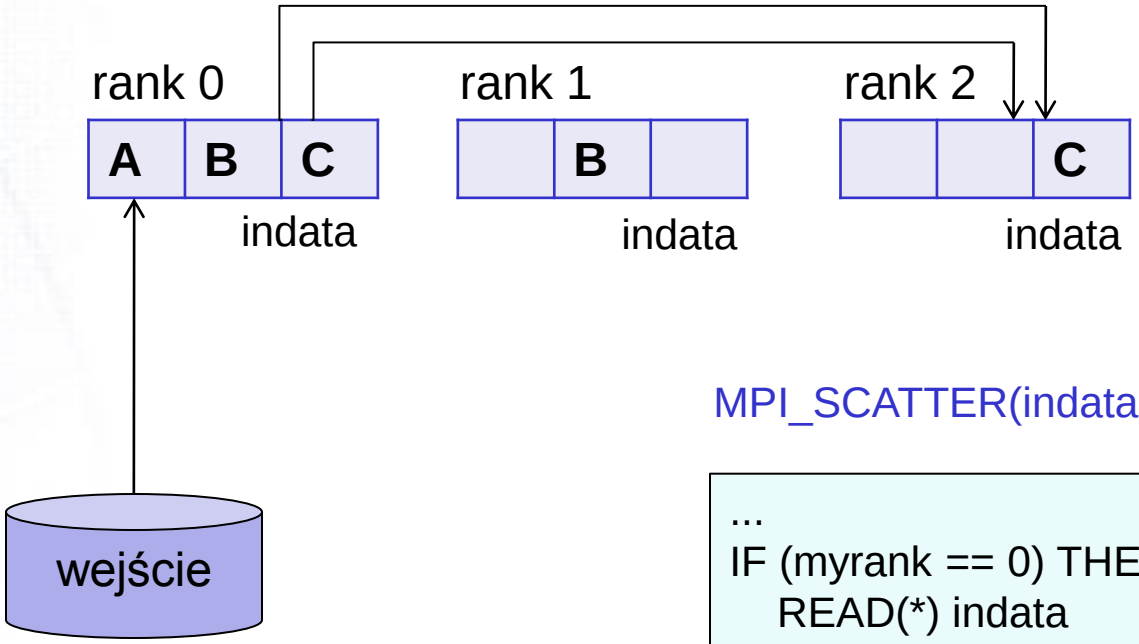
Dekompozycja danych. Wejście.



Wejście wspólne; rozdzielne

```
...
IF (myrank == 0) THEN
   READ(*) indata
END IF
CALL MPI_BCAST(indata,...)
...
```

Dekompozycja danych. Wejście.

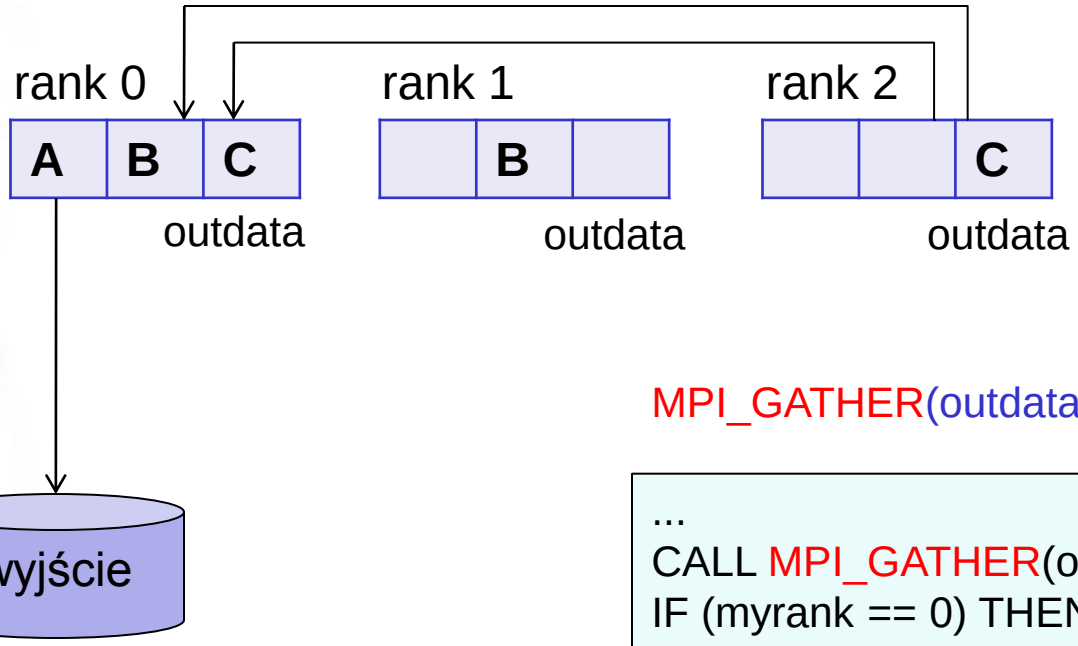


MPI_SCATTER(indata,...)

```
...
IF (myrank == 0) THEN
   READ(*) indata
END IF
CALL MPI_SCATTER(indata,...)
...
```

Wejście wspólne/rozdzielne

Dekompozycja danych. Wyjście.

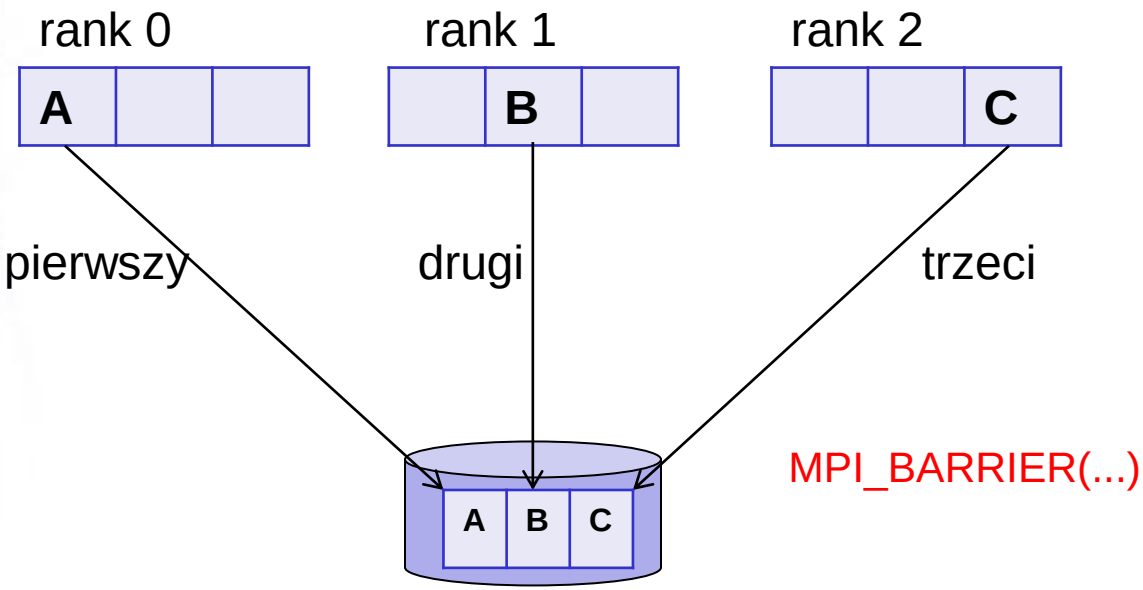


Wyjście wspólne/rozdzielone

MPI_GATHER(outdata,...)

```
...
CALL MPI_GATHER(outdata,...)
IF (myrank == 0) THEN
   READ(*) outdata
END IF
...
```

Zapis sekwencyjny. Wyjście.



Wyjście wspólne.

Zapis wyników w koleności

Zapis sekwencyjny danych

```
! Zapis danych do pliku dzielonego; sekwencyjny
...
DO irank = 0, nprocs - 1
        CALL MPI_BARRIER(MPI_COMM_WORLD, ierr)
        IF (irank == myrank) THEN
                IF (myrank == 0) THEN
                        OPEN(10, FILE='output')
                ELSE
                        OPEN(10, FILE='output', POSITION='APPEND')
                ENDIF
                WRITE(10) (outdata(i), i=ista,iend)
                CLOSE(10)
        ENDIF
ENDDO
...
```

Paralelizacja pętli *do*

- Blokowa dekompozycja danych
- Cykliczna dekompozycja danych
- Blokowo-cykliczna dekompozycja danych

Blokowa dd

- Dane są podzielone na p porcji (p=liczba procesów)
- Wygodnie jest używać sprawdzonej procedury, np.

```
SUBROUTINE range(n1, n2, nprocs, irank, ista, iend)
    iwork1 = (n2 - n1 + 1) / nprocs
    iwork2 = MOD(n2 - n1 + 1, nprocs)
    ista = irank * iwork1 + n1 + MIN(irank, iwork2)
    iend = ista + iwork1 - 1
    IF (iwork2 > irank) iend = iend + 1
   END
```

n1, n2 – zakres wskaźników (integer);

nprocs – liczba procesów (int); irank – numer procesu (integer

ista, iend – numery wskaźników dla procesu irank

Blokowa dd

Podobna procedura (Sci. Sub. Lib. For AIX):

```
SUBROUTINE range(n1, n2, nprocs, irank, ista, iend)
iwork = (n2 - n1) / nprocs + 1
ista = MIN(irank * iwork + n1, n2 + 1)
iend = MIN(ista + iwork - 1, n2)
END
```

Zadanie. Narysuj schemat podziału na 4 procesory tablicy 14-o elementowej wyznaczony przez obie procedury range().

Zadanie. Wykonaj sumowanie liczb i=1,…,n (przez zwykłe dodawanie!) i) z jednym procesorem ii) z p procesorami używając dekompozycji blokowej danych.

Cykliczna dekompozycja danych

- W dekompozycji cyklicznej iteracje przypisuje się procesom kolejno, z powtórzeniami.
- Przykład. Następującą pętlę

```
do i=n1, n2
   obliczenia
end do
```

zapisuje się w postaci:

```
do i = n1 + myrank, n2, nprocs
   obliczenia
end do
```

Pętla ta jest wykonywana przez każdy proces ale zakres jest zawsze inny.

Cykliczna dd

Zadanie. Wykonaj sumowanie liczb i=1,…,n (przez zwykłe dodawanie!) i) z jednym procesorem ii) z p procesorami używając dekompozycji cyklicznej danych w modelu MPI. (Użyj MPI_REDUCE() do zsumowania wyników)

Dekompozycja blokowo-cykliczna

Przykład. Pętlę

```
do i=n1, n2
        obliczenia
end do
```

zamieniamy na:

```
do j = n1+myrank*iblock, n2, nprocs*iblock
  do i=j, MIN(j+iblock-1, n2)
     obliczenia
    end do
end do
```

Zadanie. Narysuj schemat dekompozycji elementów 14-o elementowej tablicy między cztery procesory 0,...,3 (blokowo, cyklicznie).

Paralelizacja pętli zanurzonych

- Tablice 2 wymiarowe (Fortran) zapisywane są w pamięci komputera kolumnami, tzn. Kolejne komórki pamięci zajmują elementy z kolejnych wierszy tablicy. Jeśli używamy macierzy a(:,:) w pętli podwójnej to możemy to zrobić na dwa sposoby:

```
! 1.
do i=
  do j=
    a(i,j)=...
```

```
2.
 do i=
    do j=
      a(j,i)=...
```

Sposób 2 jest bardziej ekonomiczny gdyż w tym przypadku szybszy jest dostęp do pamięci. Podobna reguła dotyczy paralelizacji pętli do.

PROGRAMY

Metoda różnic skończonych

W metodzie różnic skończonych interesują nas wyrażenia postaci

a(i) = b(i-1) + b(i+1) gdzie b(:) zawiera wartości funkcji, natomiast a(:) zawiera pochodne (opuszczone sa tu czynniki liczbowe itd.). Program obliczeń pokazano niżej.

```
PROGRAM mrs1d
 IMPLICIT REAL*8(a-h,o-z)
 PARAMETER (n = 11)
 DIMENSION a(n), b(n)
 DO i = 1, n
   b(i) = i
 END DO
 DO i = 2, n-1
    a(i) = b(i-1) + b(i+1)
 END DO
END
```

Metoda różnic skończonych (MRS)

W przypadku obliczeń równoległych program jest bardziej skomplikowany. *(Y. Aoyama, J. Nakano: RS/600 SP: Practical MPI Programming; www.redbooks.ibm.com; również: G. Karniadakis, R.M. Kirby. Parallel scientific computing in c++ and MPI, Cambridge UP, 2003.)*

```
 1  PROGRAM main
 2  INCLUDE 'mpif.h'
 3  IMPLICIT REAL*8(a-h,o-z)
 4  PARAMETER (n = 11)
 5  DIMENSION a(n), b(n)
 6  INTEGER istatus(MPI_STATUS_SIZE)
 7  CALL MPI_INIT(ierr)
 8  CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
 9  CALL MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
10   CALL range(1, n, nprocs, myrank, ista, iend)
11   ista2 = ista
12   iend1 = iend
13     IF (myrank == 0) ista2 = 2
14   IF(myrank==nprocs-1) iend1=n-1
15   inext = myrank + 1
16   iprev = myrank - 1
```

MRS 1-wymiar

```
17   IF (myrank == nprocs-1) inext = MPI_PROC_NULL
18  IF (myrank == 0)       iprev = MPI_PROC_NULL
19  DO i = ista, iend
20    b(i) = i
21  END DO
22  CALL MPI_ISEND(b(iend), 1, MPI_REAL8, inext, 1, MPI_COM_WORLD, isend1, ierr)
23  CALL MPI_ISEND(b(ista), 1, MPI_REAL8, iprev, 1, MPI_COM_WORLD, isend2, ierr)
24  CALL MPI_IRECV(b(ista-1), 1, MPI_REAL8, iprev, 1, MPI_COMM_WORLD, irecv1, ierr)
25  CALL MPI_IRECV(b(iend+1), 1, MPI_REAL8, inext, 1, MPI_COMM_WORLD, irecv2, ierr)
26  CALL MPI_WAIT(isend1, istatus, ierr)
27  CALL MPI_WAIT(isend2, istatus, ierr)
28  CALL MPI_WAIT(irecv1, istatus, ierr)
29  CALL MPI_WAIT(irecv2, istatus, ierr)
30  DO i = ista2, iend1
31        a(i) = b(i-1) + b(i+1)
32  END DO
33   CALL MPI_FINALIZE(ierr)
34  END
```

MRS - uwagi

Uwagi o programie

L 2,7,8,9 – inicjalizacja MPI; W linii 6 zadeklarowano istatus bo będzie używana procedura MPI_WAIT (w związku z MPI_ISEND/IRECV – nieblokujące). Dane przekazywane do innych procesów leżą na granicy obszaru

MRS – 2 wymiary

- Projekt.

  Opracuj program liczenia pochodnych w przypadku 2 wymiarowej sieci.

Redukcja - przykład

wersja sekwencyjna

```
...
sum1 = 0.0
sum2 = 0.0
amax = 0.0
DO i = 1, n
  a(i) = a(i) + b(i)
  c(i) = c(i) + d(i)
  e(i) = e(i) + f(i)
  g(i) = g(i) + h(i)
     x(i) = x(i) + y(i)
     sum1 = sum1 + a(i)
     sum2 = sum2 + c(i)
     IF (a(i) > amax) amax=a(i)
END DO
DO i = 1, n
g(i) = g(i) * sum1 + sum2
END DO
PRINT *, amax
...
```

Wersja równoległa

```
REAL works(2), workr(2)
sum1 = 0.0; sum2 = 0.0; amax = 0.0
DO i = ista, iend
  a(i) = a(i) + b(i); c(i) = c(i) + d(i)
  e(i) = e(i) + f(i); g(i) = g(i) + h(i)
  x(i) = x(i) + y(i)
sum1 = sum1 + a(i);   sum2 = sum2 + c(i)
IF (a(i) > amax) amax = a(i)
END DO
works(1) = sum1;  works(2) = sum2
CALL MPI_ALLREDUCE(works, workr, 2, MPI_REAL,&MPI_SUM,
     MPI_COMM_WORLD, ierr)
sum1 = workr(1);  sum2 = workr(2)
CALL MPI_REDUCE(amax, aamax, 1, MPI_REAL,&MPI_MAX, 0,
     MPI_COMM_WORLD, ierr)
amax = aamax
DO i = ista, iend
  g(i) = g(i) * sum1 + sum2
END DO
IF (myrank == 0) THEN
  PRINT *, amax
ENDIF
```

Pomiar czasu wykonania

```
PROGRAM Time
REAL*4 t1, t2
...
CALL MPI_BARRIER(MPI_COMM_WORLD, ierr)
t1 = secnds(0.0)
! Mierzy się czas wykonania tej sekcji…
CALL MPI_BARRIER(MPI_COMM_WORLD, ierr)
t2 = secnds(t1)
PRINT *, 'Elapsed time (sec) = ', t2 - t1
...
END
```

Komunikacja – deadlock, jeszcze...

Komunikacja odbywa się w dość złożony sposób:

WYSYŁANIE:

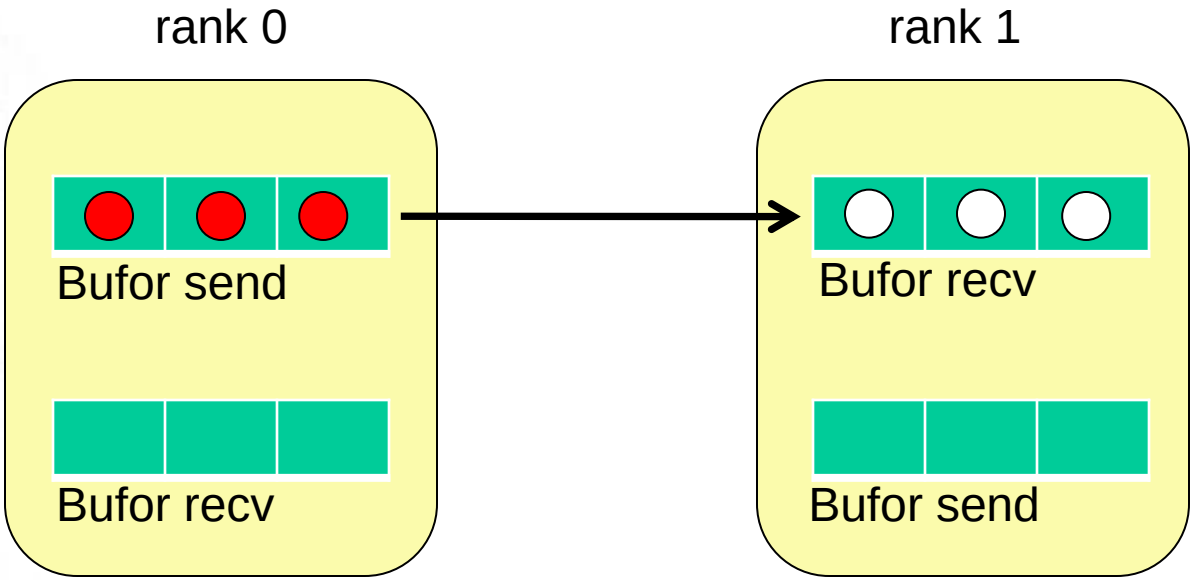
a) Użytkownik kopiuje dane do własnego bufora
b) Użytkownik woła procedurę MPI typu send
c) System kopiuje dane z bufora użytkownika do bufora systemowego
d) System wysyła dane z bufora systemowego do adresata

ODBIÓR:

a) Użytkownik wywołuje procedurę MPI typu recv
b) System otrzymuje dane od nadawcy i kopiuje je do bufora systemowego
c) System kopiuje dane do bufora użytkownika
d) Użytkownik korzysta z danych we własnym buforze

Zadanie. Wykonaj poglądowy szkic procesu wysyłania i odbioru danych.

Point-to-point (p2p) jednostronna



Kiedy chcemy wysłać komunikat do dyspozycji mamy cztery kombinacje procedur MPI zależnie od tego czy komunikacja ma być blokująca czy nie.

p2p 2

1) Blokujące send I blokujące recive

```
IF (myrank==0) THEN
   CALL MPI_SEND(sendbuf, icount, MPI_REAL8, 1, itag, &
                 MPI_COMM_WORLD, ierr)
ELSEIF (myrank==1) THEN
   CALL MPI_RECV(recvbuf, icount, MPI_REAL8, 0, itag, &
                 MPI_COMM_WORLD, istatus, ierr)
ENDIF
```

2) Nieblokujące send I blokujące recive

```
IF (myrank==0) THEN
   CALL MPI_ISEND(sendbuf, icount, MPI_REAL8, 1, itag,    &
                  MPI_COMM_WORLD, ireq, ierr)
   CALL MPI_WAIT(ireq, istatus, ierr)
ELSEIF (myrank==1) THEN
    CALL MPI_IRECV(recvbuf, icount, MPI_REAL8, 0, itag,    &
                   MPI_COMM_WORLD, ireq, ierr)
ENDIF
```

p2p 3

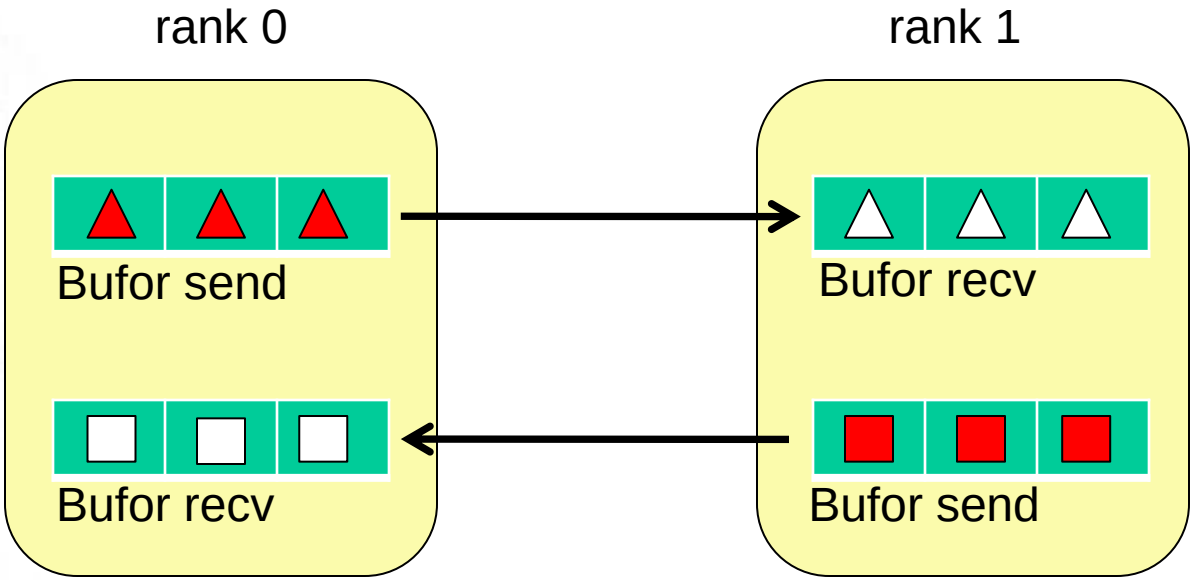
3) Blokujące send I nieblokujące recive

```
IF (myrank==0) THEN
  CALL MPI_SEND(sendbuf, icount, MPI_REAL8, 1, itag,  &
               MPI_COMM_WORLD, ierr)
ELSEIF (myrank==1) THEN
  CALL MPI_IRECV(recvbuf, icount, MPI_REAL8, 0, itag, &
               MPI_COMM_WORLD, ireq, ierr)
CALL MPI_WAIT(ireq, istatus, ierr)
ENDIF
```

4) Nieblokujące send I nieblokujące recive

```
IF (myrank==0) THEN
  CALL MPI_ISEND(sendbuf, icount, MPI_REAL8, 1, itag, &
               MPI_COMM_WORLD, ireq, ierr)
ELSEIF (myrank==1) THEN
  CALL MPI_IRECV(recvbuf, icount, MPI_REAL8, 0, itag,  &
               MPI_COMM_WORLD, ireq, ierr)
ENDIF
CALL MPI_WAIT(ireq, istatus, ierr)
```

p2p dwustronna



Możliwy impas.

Mamy w zasadzie 3 możliwości:

a) p0 i p1: wyślij, a potem odbierz (w-o)
b) p0 i p1 odbierz, a potem wyślij (o-w)
c) p0: w-o, a p1: o-w

p2p dwustronna (a)

a) Oba p0 i p1 w-o

```
IF (myrank==0) THEN
  CALL MPI_SEND(sendbuf, ...)
  CALL MPI_RECV(recvbuf, ...)
ELSEIF (myrank==1) THEN
  CALL MPI_SEND(sendbuf, ...)
  CALL MPI_RECV(recvbuf, ...)
ENDIF
-----------------------------------------------------
IF (myrank==0) THEN
CALL MPI_ISEND(sendbuf, ...,
ireq, ...)
CALL MPI_WAIT(ireq, ...)
CALL MPI_RECV(recvbuf, ...)
ELSEIF (myrank==1) THEN
CALL MPI_ISEND(sendbuf, ..., ireq, ...)
CALL MPI_WAIT(ireq, ...)
CALL MPI_RECV(recvbuf, ...)
ENDIF
```

Może się zdarzyć impas (deadlock)

...też deadlock

p2p dwustronna (a, okay)

```
IF (myrank==0) THEN
  CALL MPI_ISEND
     (sendbuf, ...,ireq, ...)
  CALL MPI_RECV(recvbuf, ...)
  CALL MPI_WAIT(ireq, ...)
ELSEIF (myrank==1) THEN
  CALL MPI_ISEND
    (sendbuf, ...,ireq, ...)
  CALL MPI_RECV(recvbuf, ...)
  CALL MPI_WAIT(ireq, ...)
ENDIF
```

Wymiana zawsze bezpieczna

p2p dwustronna (b)

b) o-w

```
IF (myrank==0) THEN
      CALL MPI_RECV(recvbuf, ...)
   CALL MPI_SEND(sendbuf, ...)
ELSEIF (myrank==1) THEN
      CALL MPI_RECV(recvbuf, ...)
   CALL MPI_SEND(sendbuf, ...)
ENDIF
---------------------------------
IF (myrank==0) THEN
CALL MPI_IRECV(recvbuf, ..., ireq, ...)
CALL MPI_SEND(sendbuf, ...) CALL
      MPI_WAIT(ireq, ...)
ELSEIF (myrank==1) THEN
CALL MPI_IRECV(recvbuf, ..., ireq, ...)
CALL MPI_SEND(sendbuf, ...)
CALL MPI_WAIT(ireq, ...)
ENDIF
```

Zakleszczenie

... zawsze

...wymiana danych zawsze bezpieczna

p2p dwustronna (c)

c) w-o, o-w

```
IF (myrank==0) THEN
  CALL MPI_SEND(sendbuf, ...)
  CALL MPI_RECV(recvbuf, ...)
ELSEIF (myrank==1) THEN
  CALL MPI_RECV(recvbuf, ...)
  CALL MPI_SEND(sendbuf, ...)
ENDIF
```

```
IF (myrank==0) THEN
  CALL MPI_ISEND(sendbuf, ..., ireq1, ...)
  CALL MPI_IRECV(recvbuf, ..., ireq2, ...)
ELSEIF (myrank==1) THEN
  CALL MPI_ISEND(sendbuf,..., ireq1, ...)
  CALL MPI_IRECV(recvbuf, ..., ireq2, ...)
ENDIF
CALL MPI_WAIT(ireq1, ...)
CALL MPI_WAIT(ireq2, ...)
```

Dobrze

ZALECANY SPOSÓB
WYMIANY DANYCH!

Problemy...?

proces:	0	1	2	3	0	1	2	3	0	1	2	3	0	1
element a:	a1	a2	a3	a4	a5	a6	a7	a8	a9	a10	a11	a12	a13	a14