Programowanie Równoległe
Wykład 5

MPI - Message Passing Interface
(część 3)

Maciej Matyka
Instytut Fizyki Teoretycznej

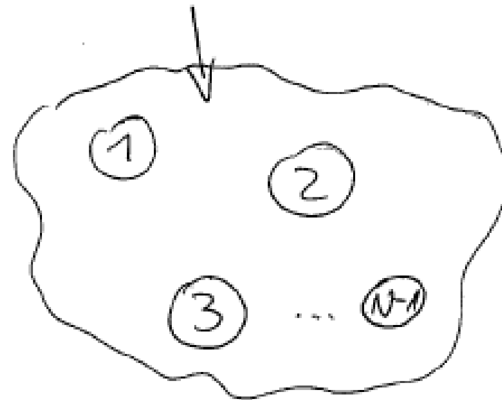
Uniwersytet Wrocławski

MPI, wykład 3.

Plan:

- wirtualne topologie
- badanie skalowanie czasu rozwiązania
- własne typy danych

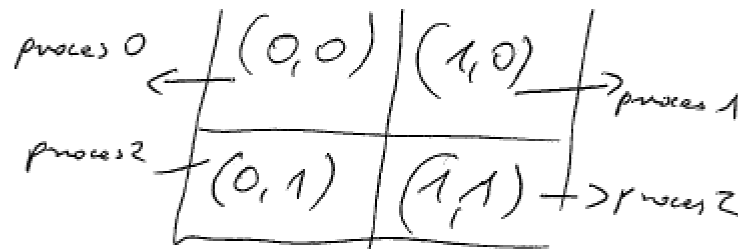
Wirtualne topologie



- predefiniowany komunikator MPI_COMM_WORLD
- zawiera listę wszystkich procesów z unikalnym numerowaniem

Wirtualne topologie

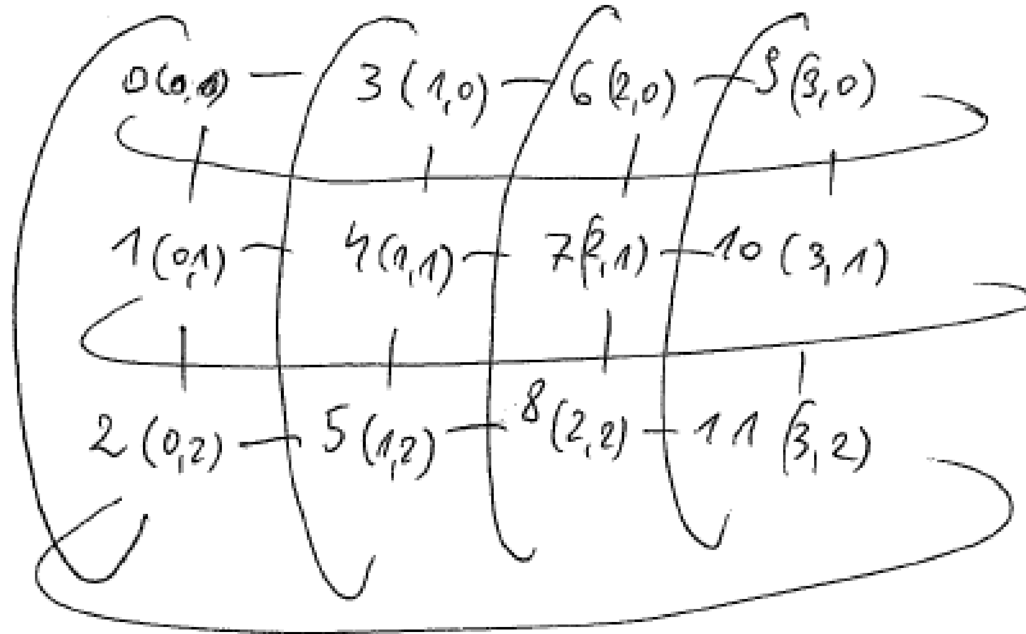
- chcemy mapować kawałki problemu (np. tablicy) na osobne procesy MPI
- przykład tablica 2x2:



- MPI dostarcza funkcje mapujące dla typowych problemów

Uniwersytet Wrocławski

Topologia sieci periodycznej 2D



- ndims = 2   (wymiar przestrzeni)
- dim[0] = 3   (ilość elementów w kierunku 'x')
- dim[1] = 4   (ilość elementów w kierunku 'y')
- per[0] = per[1] = true   (periodyczność sieci)

Uniwersytet Wrocławski

MPI_Cart_create

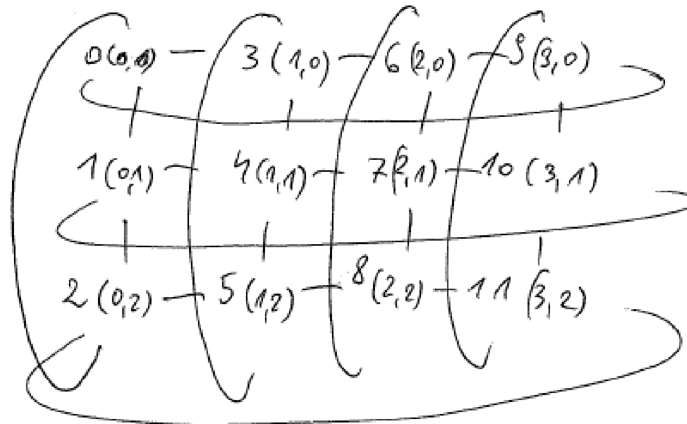
```
int MPI_Cart_create(MPI_Comm comm_old,
                             int ndims,
                             int *dims,
                          int *periods,
                           int reorder,
                    MPI_Comm *comm_cart)
```

- nowy komunikator dla topologii kartezjańskiej
- reorder – dopuszcza (lub nie) zmianę numeracji procesów (true/false)

Uniwersytet Wrocławski

Mapowanie współrzędne sieci → proces

```
int MPI_Cart_rank(  MPI_Comm comm,
                    int *in_coords,
                    int *out_rank)
```



in_coords – tablica współrzędnych

out_rank – id procesu, uwzględnia periodyczność (możliwe błędy, gdy kierunek nie jest periodyczny)

Uniwersytet Wrocławski

Mapowanie proces → współrzędne na sieci

```
int MPI_Cart_coords(   MPI_Comm comm,
                       int in_rank,
                       int in_maxdims,
                       int *out_coords)
```

- in_rank – id procesu (wejście)
- in_maxdims – ilość współrzędnych w out_coords
- out_coords – tablica współrzędnych (wynik)

Wyznaczenie sąsiada

```
int MPI_Cart_shift( MPI_Comm comm,
                    int dir,
                    int disp,
                    int *rank_source,
                    int *rank_dest)
```

- dir – kierunek [0-dim)
- disp – zwrot (0 - w dół, w górę)

zwraca id dwóch sąsiadów: source („w kierunku siebie”) i destination („od siebie”)

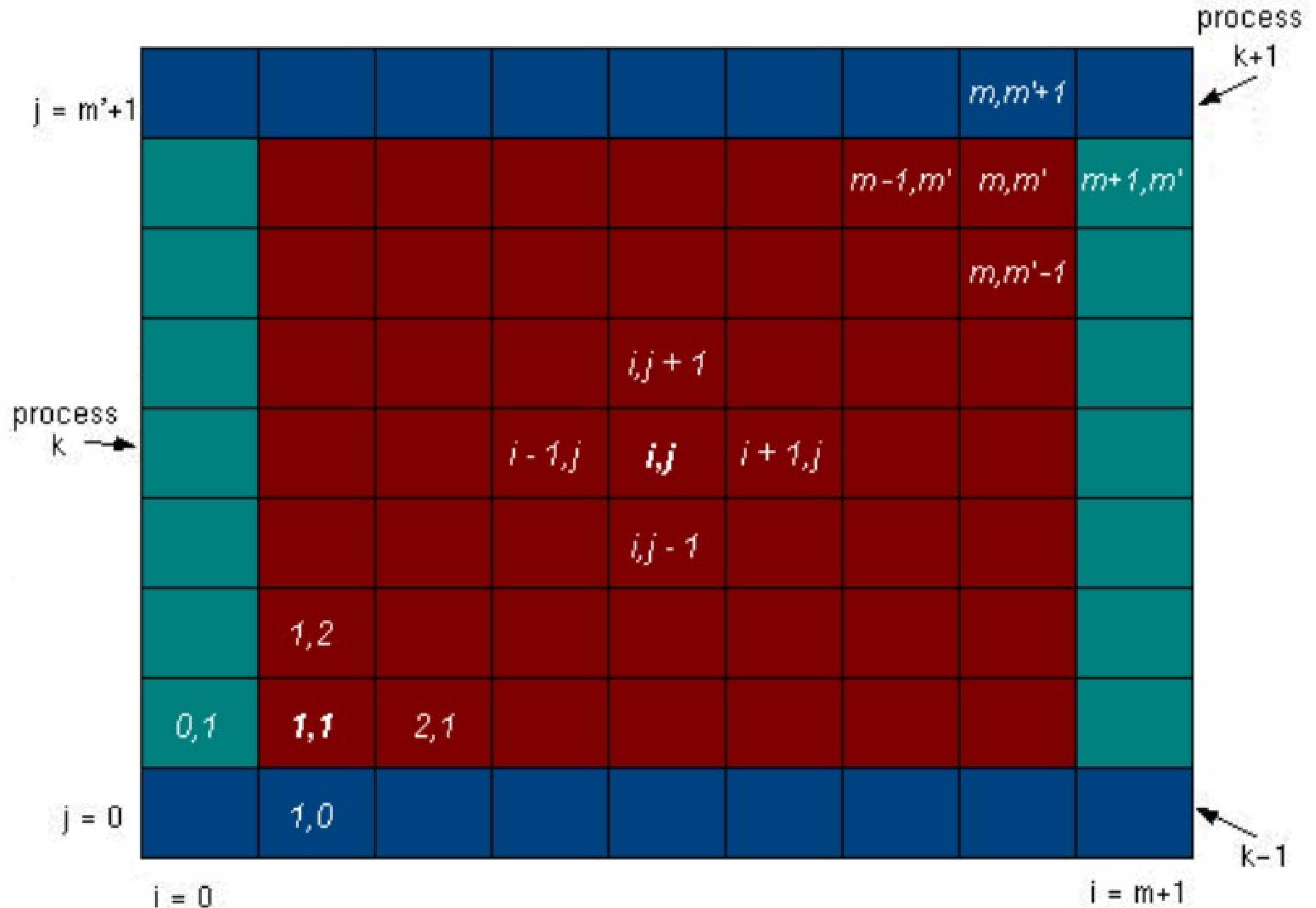
brak sąsiada: MPI_PROC_NULL

Podział kartezjański

Procedura:

- dzielimy problem (np. wydzielamy podsieci o mniejszym wymiarze)
- tworzymy osobny komunikator dla każdej podsieci
  - MPI_Cart_sub
- Każda podsieć może wykonywać własne operacje kolektywne (np. MPI_Reduce)
- Na przykład: operacje na macierzach (rzędy)

Przykład podziału problemu



(http://people.sc.fsu.edu/~jburkardt/pdf/mpi_course.pdf)

MPI_Cart_sub – tworzenie podgrupy w niższym wymiarze

```
int MPI_Cart_sub (  MPI_Comm comm,
                     int *remain_dims,
                    MPI_Comm *newcomm)
```

- comm – komunikator o topologii kartezjańskiej
- remain_dims – określa czy dany wymiar jest zachowany
- przykład: dim = 3, d[] = {3,2,4}
  - Remain_dims[] = {true, false, true}
    - Stworzy 2 komunikatory, każdy z 12 procesami w topologii kartezjańskiej 3 x 4
  - Remain_dims[] = {false, true, false}
    - Stworzy 12 komunikatorów 1d o wymiarze 2

Badanie wydajności

Wall time

- Przykład:

```
t0 = MPI_Wtime();

(…)

t1 = MPI_Wtime();

printf(" %f \n", t1-t0);
```

- czas może, ale nie musi być zsynchronizowany między procesami
- zmienna środowiskowa MPI_WTIME_IS_GLOBAL

Uniwersytet Wrocławski

Badanie wydajności

```
double MPI_Wtick(void);
```

- rozdzielczość czasu podawana przez MPI_Wtime()
- np. 10^{-3} jeśli MPI_Wtime() ma dokładność do milisekundy

MPI_Barrier

- każdy proces wykonuje się „inaczej”
- trzeba to uwzględnić w pomiarach (max time?)

Przykład:

```
MPI_Barrier(MPI_COMM_WORLD);        // 1

start = MPI_Wtime();
usleep(1000*rank);

MPI_Barrier(MPI_COMM_WORLD);        // 2

printf("proces %d: dt=%f \n", rank, (MPI_Wtime() - start));
```

Z synchronizacją MPI_Barrier

```
maq@amiga:~/IFT-2012-20
proces 0: dt=0.001058
proces 1: dt=0.001059
```

Bez synchronizacji MPI_Barrier

```
maq@amiga:~/IFT-2012-201
/a.out
proces 0: dt=0.000055
proces 1: dt=0.001065
```

Uniwersytet Wrocławski

Efektywność redukcji

```
(…)

   t0 = MPI_Wtime();

   for(i = 0; i < L; i++)
    send[i]  =  rand() / (float)RAND_MAX;

   MPI_Reduce(send, recv, L, MPI_DOUBLE, MPI_SUM,
   0, MPI_COMM_WORLD);

   t1 = MPI_Wtime();

   if(myrank==0)
   {
       for(i=1; i < L; i++)
           recv[0] += recv[i];
   }

(…)
```

```
maq@zero ~/pr $ gcc srednia.c -lmpi -Wall
maq@zero ~/pr $ mpirun -np 4 ./a.out 4
 0.735683 4 0.000132
maq@zero ~/pr $ mpirun -np 4 ./a.out 16
 0.561034 16 0.000198
maq@zero ~/pr $ mpirun -np 4 ./a.out 262144
 0.499849 262144 0.006271
maq@zero ~/pr $ mpirun -np 4 ./a.out 16777216
 0.499991 16777216 0.308301
```

Uniwersytet Wrocławski

Skalowanie rozwiązania

Jeśli:

L – rozmiar problemu

n – liczba procesów

Rozważamy dwa typy skalowania czasu wykonania:

- Skalowanie słabe
  - przyspieszenie w funkcji n dla stałego L/n
- Skalowanie silne
  - przyspieszenie w funkcji n dla stałego L

Przykład - reduce

```
(…)

   t0 = MPI_Wtime();

   for(i = 0; i < L; i++)
     send[i] = rand() / (float)RAND_MAX;

   MPI_Reduce(send, recv, L, MPI_DOUBLE, MPI_SUM, 0,
              MPI_COMM_WORLD);


   t1 = MPI_Wtime();

  (…)

 //  następnie redukcja różnic t1-t0 ze wszystkich procesów
```

(ćwiczenia)

Uniwersytet Wrocławski

Typy danych

Uniwersytet Wrocławski

Typy danych w MPI

To ważne żeby używać wbudowanych typów do komunikacji (komunikacja między różnymi systemami).

MPI datatype	C datatype
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	
MPI_PACKED	

Uniwersytet Wrocławski

Typy pochodne danych MPI

- MPI pozwala definiować pochodne typy danych
- Dzięki temu np. strukturę można przesłać jednym MPI_Send / MPI_Recv

- komunikaty wysyłane tylko w całości
- wymagają zdefiniowania typu i rejestracji

Ogólny typ danych MPI

- Typ danych w MPI to struktura zawierająca:
  - sekwencję typów podstawowych
  - sekwencję przesunięć (offsetów)
- Np:
  - *typemap = { (type0; disp0); ... ; (typen-1; dispn-1) } ;*

{ (MPI_INT, 0), (MPI_FLOAT, 4), (MPI_FLOAT, 20)}

Uniwersytet Wrocławski

MPI_Type_contiguous

```
int MPI_Type_contiguous(int count,
                        MPI_Datatype oldtype,
                        MPI_Datatype *newtype);
```

- count – ilość elementów
- oldtype – typ pojedynczego elementu
- newtype – nowy typ danych

MPI_Type_contiguous

- Przykład, wektor 2D i 3D:

```
(…)

MPI_Datatype MPI_2D;
MPI_Datatype MPI_3D;

MPI_Type_contiguous(2, MPI_DOUBLE, &MPI_2D );
MPI_Type_contiguous(3, MPI_DOUBLE, &MPI_3D );

MPI_Type_commit(&MPI_2D);
MPI_Type_commit(&MPI_3D);

(…)
```

- Teraz można wykonać np.:

```
MPI_Send( buf, size, MPI_2D, 1, 0, MPI_COMM_WORLD);
```

Inne typy pochodne

- MPI_Type_struct
  - najbardziej ogólny typ pochodny (podajemy lokację, rozmiar i typ wszystkich danych)
- MPI_Type_contiguous
  - dane ułożone kolejno w pamięci (tablica 1d)
- MPI_Type_vector
- MPI_Type_hvector
  - dane ułożone jak w MPI_Type_contiguous z przesunięciem
  - np. fragment tablicy (podajemy m.in. parametr stride oznaczający jaką część pomijamy)

Następny wykład - OpenMP

Uniwersytet Wrocławski

Programowanie Rownolegle

Zbigniew Koza, Maciej Matyka

Lista wykladow:

- Wyklad 1, Wstep (ZK), 7/10/2013
- Wyklad 2, Projektowanie algorytmow rownoleglych (ZK), 14/10/2013 -
- Wyklad 3, MPI (MM), 21/10/2013 [pdf] [lista1]
- Wyklad 4, MPI (MM), 28/10/2013 [ppt] [lista2]
- Wyklad 5, MPI (MM), 4/11/2013 [ppt] [lista3]
- Wyklad 6, OpenMP (ZK), 11/11/2012
    - Lista osob: [link]
    - http://www.mpi-forum.org/
    - http://www.mcs.anl.gov/research/projects/mpi/sendmode.html
    - http://locklessinc.com/man/MPI_Issend.shtml
    - http://people.sc.fsu.edu/~jburkardt/pdf/mpi_course.pdf
    - http://www.zib.de/zibdoc/mpikurs/

www.ift.uni.wroc.pl/~koma/pr/

Literatura

- http://people.sc.fsu.edu/~jburkardt/pdf/mpi_course.pdf (polecam!)
- Wolfgang Baumann, ZIB – Parallel Programming with MPI
- MPI: A Message-Passing Interface Standard, Version 1.3
- http://aragorn.pb.bialystok.pl/~wkwedlo/PC3.pdf

Slajdy odrzucone

MPI_Type_struct

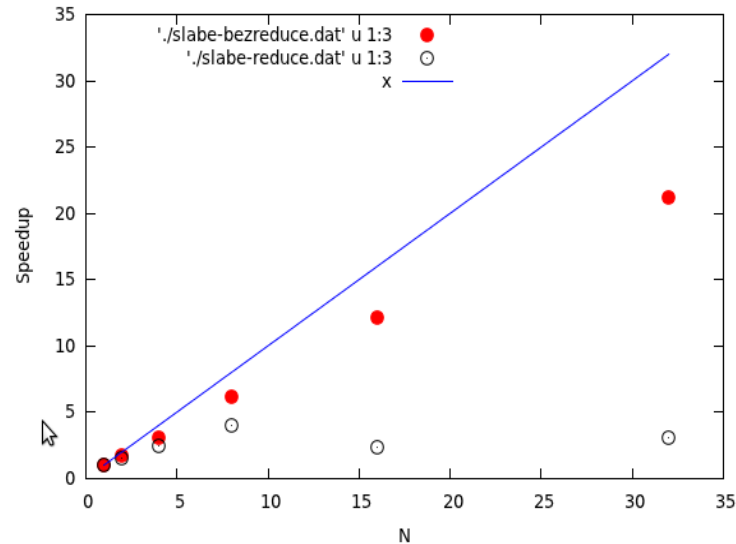
```
int MPI_Type_create_struct(
                    int count,
                    int array_of_blocklengths[],
                    MPI_Aint array_of_displacements[],
                    MPI_Datatype array_of_types[],
                    MPI_Datatype *newtype);

MPI_Datatype        MPI_AAA;
```

- count - liczba bloków danych
- array_of_blocklength – tablica liczb elementów w bloku
- array_of_displacements - tablica przesunięć w bajtach dla bloków
- array_of_types – typ elementów w bloku
- newtype – nowy typ danych

Redukcja – skalowanie mocne

Mocne (stałe L)



- mocne skalowanie dobre do N=8 (liczba proc. na węzeł)
  - redukcja nieefektywna dla N>8
- skalowanie silne sopdziewane dużo gorsze (do sprawdzenia)

MPI_Pack – scalanie danych

```
int position, i, j, a[2];
char buff[1000];

MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

if (myrank == 0)
{
 /* SENDER CODE */
 position = 0;
 MPI_Pack(&i, 1, MPI_INT, buff, 1000, &position, MPI_COMM_WORLD);
 MPI_Pack(&j, 1, MPI_INT, buff, 1000, &position, MPI_COMM_WORLD);
 MPI_Send( buff, position, MPI_PACKED, 1, 0, MPI_COMM_WORLD);
}
else /* RECEIVER CODE */
    MPI_Recv( a, 2, MPI_INT, 0, 0, MPI_COMM_WORLD);
```

- position – automatyczna kontrola MPI
- Można użyć kombinacji:
  - MPI_Recv (... MPI_PACKED) + MPI_Unpack

  do rozpakowania złożonych struktur.

Przesyłanie struktur danych

- Jak przesłać w komunikacie więcej niż 1 zmienną?
- Najprościej: kilka MPI_Send/MPI_Recv (wady)
- Przykład (przesłanie tablicy L x H):

```
for (j=0; j<H; j++)
{
    MPI_Send(&t[j][1], L, MPI_DOUBLE, dst, tag, MPI_COMM_WORLD);
}
```

- (rzędy w całości w pamięci)

- Zalety: prostota
- Wady: dużo krótkich wiadomości (narzut komunikacji!)

Uniwersytet Wrocławski

Przesyłanie struktur danych

- Dane można przekopiować do jakiegoś bufora
- Np. tablicę 2d do tablicy 1d

```
double *buf;

(.. malloc etc.)

for(j=0; j<H; ++j)
for (i=0; i<L; ++i)
{
    *(buf++) = t[j][i];
}

MPI_Send(buf, L*H, MPI_DOUBLE, dst, tag, MPI_COMM_WORLD);
```

- Zalety: prostota, 1 długa wiadomość
- Wady: jak przesłać różne typy? Konwersja – nieefektywna.

MPI_Pack – scalanie danych

```
int MPI_Pack(
            void* inbuf,
            int incount,
            MPI_Datatype datatype,
            void *outbuf,
            int outsize,
            int *pos,
            MPI_Comm comm)
```

- inbuf - bufor wejściowy
- outbuf - bufor wyjściowy (adres początku)
- outsize – rozmiar bufora wyjściowego (w bajtach)
- pos – pozycja (w bajtach)

- Dopuszcza np. warunkowe przesłanie danych (odbieram dalej tylko jeśli część danych jest taka, albo taka)

Uniwersytet Wrocławski

Przesyłanie struktur danych

- Dane można przekopiować do jakiegoś bufora
- Np. tablicę 2d do tablicy 1d

```
double *buf;

(.. malloc etc.)

for(j=0; j<H; ++j)
for (i=0; i<L; ++i)
{
    *(buf++) = t[j][i];
}

MPI_Send(buf, L*H, MPI_DOUBLE, dst, tag, MPI_COMM_WORLD);
```

- Zalety: prostota, 1 długa wiadomość
- Wady: jak przesłać różne typy? Konwersja – nieefektywna.

Uniwersytet Wrocławski

MPI_Pack – scalanie danych

```
int MPI_Pack(
             void* inbuf,
             int incount,
             MPI_Datatype datatype,
             void *outbuf,
             int outsize,
             int *pos,
             MPI_Comm comm)
```

- inbuf - bufor wejściowy
- outbuf - bufor wyjściowy (adres początku)
- outsize – rozmiar bufora wyjściowego (w bajtach)
- pos – pozycja (w bajtach)

- Dopuszcza np. warunkowe przesłanie danych (odbieram dalej tylko jeśli część danych jest taka, albo taka)