



Fakulteta za
informacijske študije
Faculty of information studies

Univerza v Ljubljani
Fakulteta *za strojništvo*



Kreativno jedro:
Simulacije
Creative core: Simulations

Kampus šola HPC 2014

Osnove MPI v jeziku C

Matjaž Depolli
Odsek za komunikacijske sisteme
Inštitut „Jožef Stefan“



Naložba v vašo prihodnost
OPERACIJO DELNO FINANCIRA EVROPSKA UNIJA
Evropski sklad za regionalni razvoj



REPUBLIKA SLOVENIJA
**MINISTRSTVO ZA IZOBRAŽEVANJE,
ZNANOST IN ŠPORT**

Pregled predavanja

- MPI – kaj, zakaj, kako
- Splošno o medprocesni komunikaciji
 - Proces
 - Vrste komunikacije
 - Sinhronizacija
- Priprava MPI okolja
- Pogonjanje MPI programov
- MPI Hello world
- Sinhrona komunikacija
- Skupinska komunikacija
- Asinhrona komunikacija

Kaj je MPI

- *MPI (Message-Passing Interface) is a message-passing library interface specification*
- Definira **model** za prenos informacij med oddaljenimi računalniki
- Podan je kot **specifikacija knjižnice** – vse MPI operacije so v obliki funkcij in tipov za posamezni jezik, za katerega je implementacija knjižnice namenjena
- Za mnogo jezikov: **C**, **FORTRAN**, C++, Java, Python, Perl, R, ...
- Obstaja več implementacij MPI knjižnice (OpenMPI, MPICH)
- De facto standard, specifikacijo ureja MPI forum

Cilji MPI

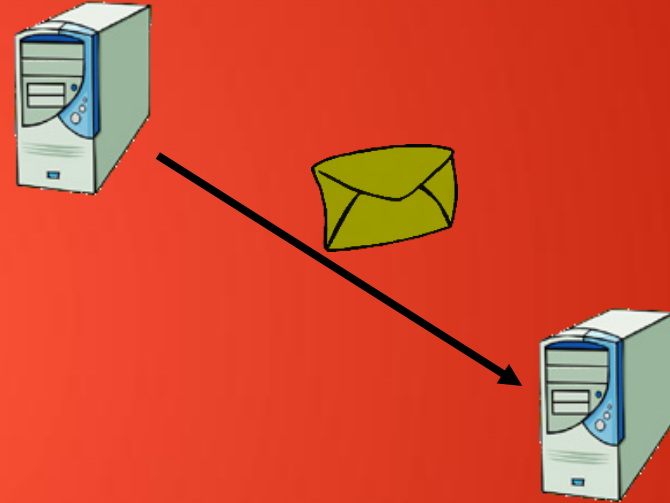
- Preprosta uporaba
- Prenosljivost
- Omogoča podporo strojne opreme
- Praktičnost, učinkovitost, skalabilnost
- Fleksibilnost (tudi za heterogena okolja)
- Praktična implementacija za C in FORTRAN
- Neodvisnost od jezika
- Zanesljiva komunikacija

Kaj sestavlja MPI implementacijo

- Izvajalno okolje (runtime environment)
- Knjižnjica funkcij, definicij tipov in struktur
- Integracija z okoljem in operacijskim sistemom (npr. procesiranje signalov)

Komunikacija

- Preko sporočil
- Enemu ali več naslovnikom
- Sinhrono / asinhrono
- Z / brez medpomnilnika
- Naslovnik je drug **proces**
 - isti / drug računalnik

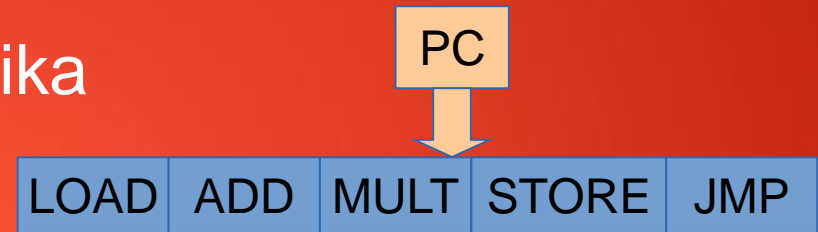


Proces

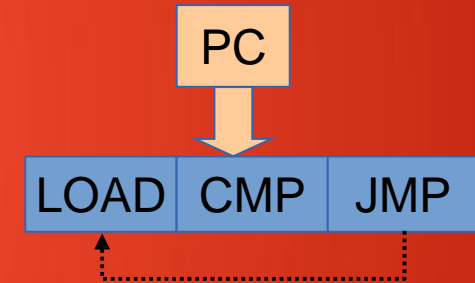
- Je program v izvrševanju
- Je okolje na računalniku, znotraj katerega se izvaja program
 - Niti programa
 - Bloki pomnilnika (dovoljenje za dostop)
 - Odprte datoteke in ostali sistemski viri
 - Ima na voljo komunikacijo s operacijskim sistemom
 - Lahko upravlja z večimi jedri / procesorji / pospeševalniki
 - Teče na enem računalniku

Proces in komunikacija – poenostavljeno

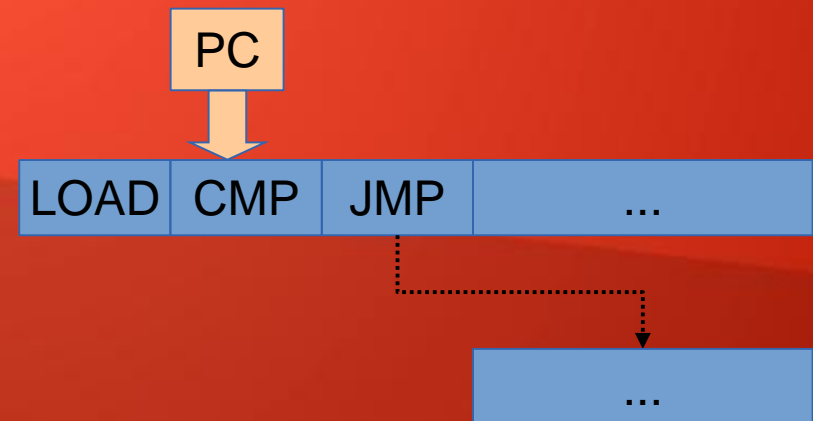
- CPU izvaja ukaze iz pomnilnika (proces v ožjem smislu)



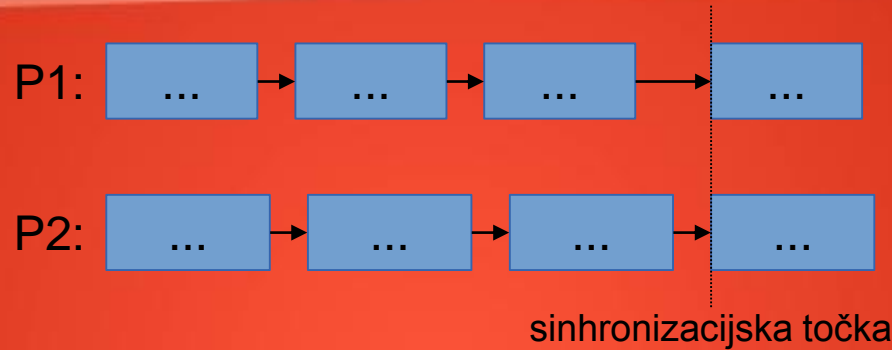
- Koko komunicirati z njim? Zamenjamo vsebino pomnilnika, ki ga bere



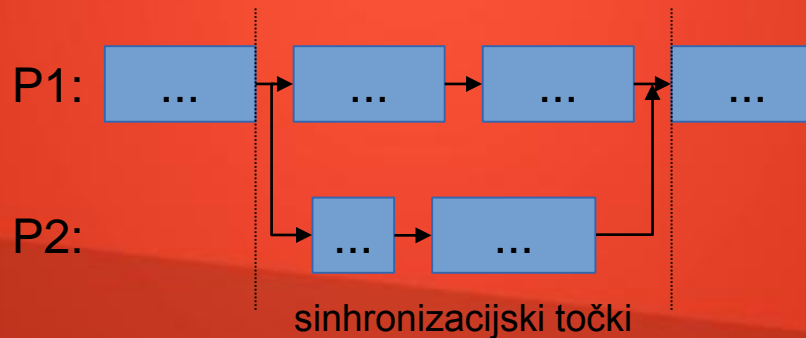
- Procesu ni treba čakati



Sinhronizacija

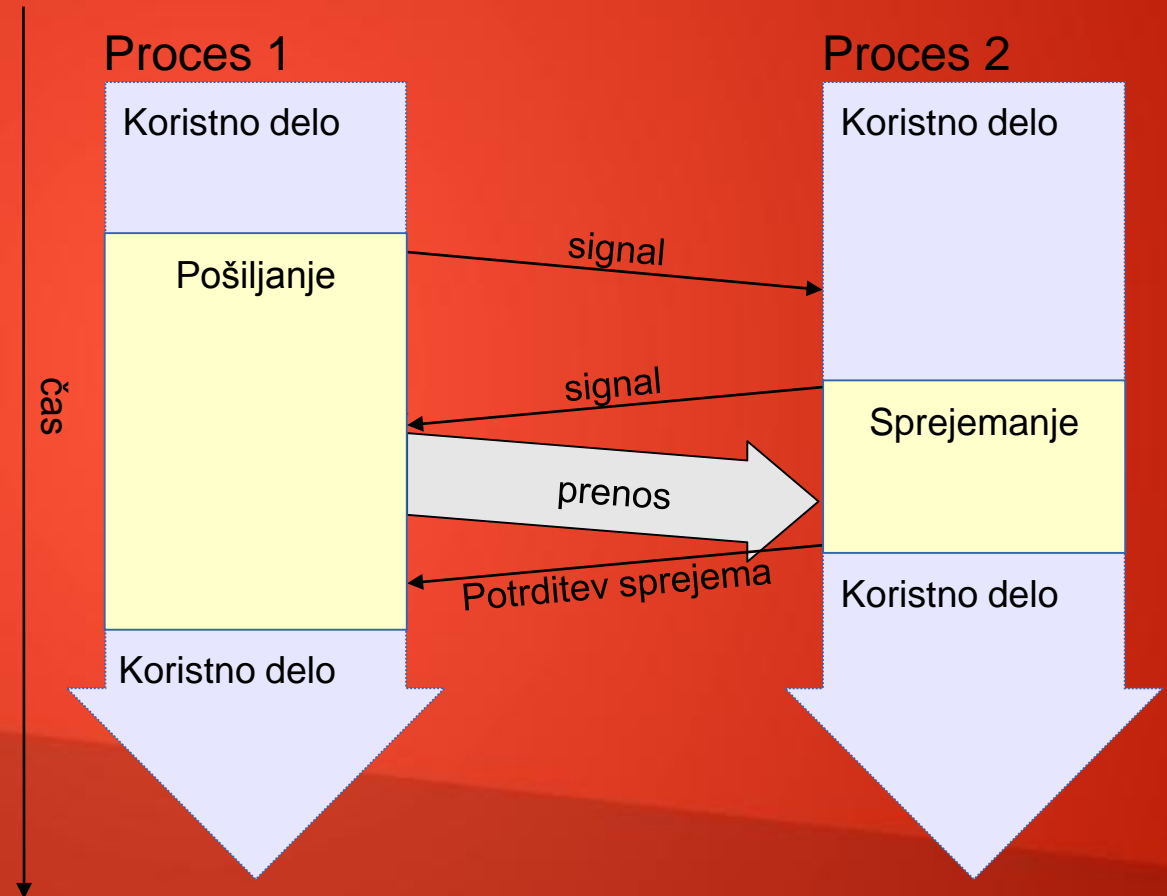


- Oba (vsi) procesi (niti) morajo končati neko nalogo preden grejo lahko na naslednjo



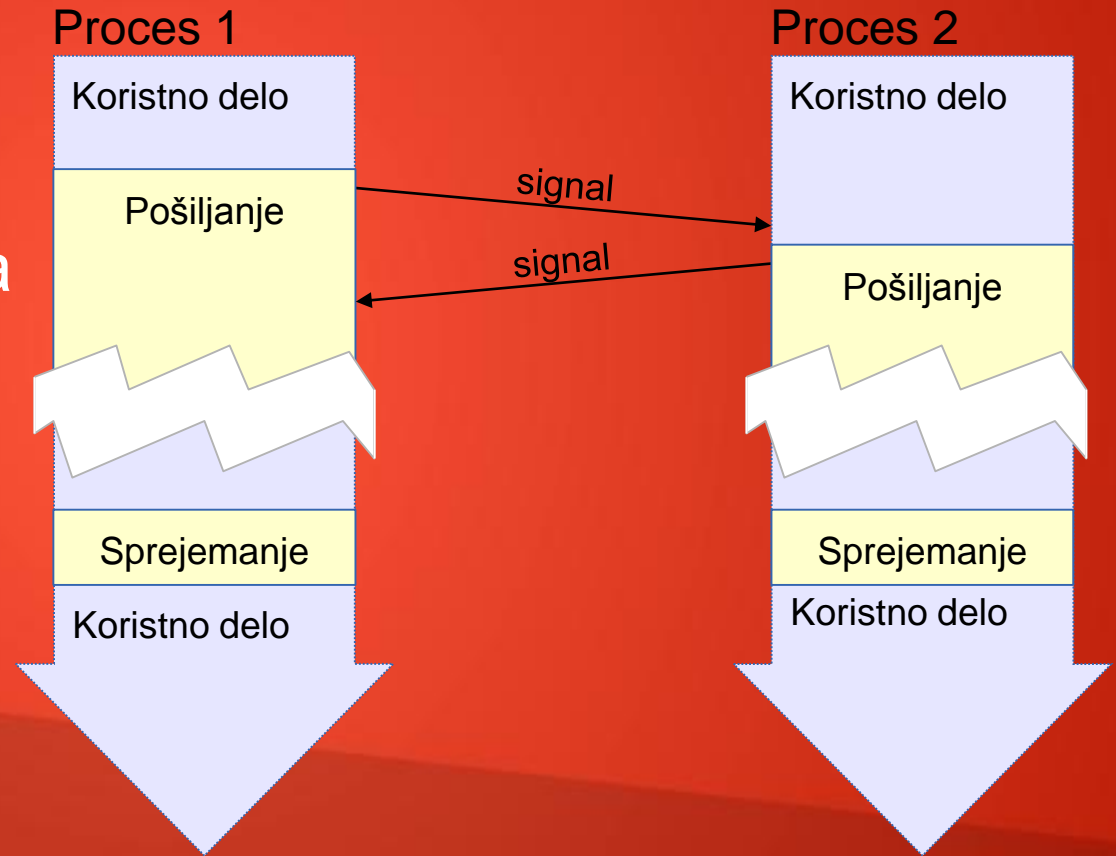
Sinhrono pošiljanje in sprejemanje

- Usklajeno
- Najbolj enostaven in varen model
- Vedno nastopata v paru pošiljanje in sprejemanje
- Vsaj en proces ponavadi čaka
- MPI_Send
- MPI_Recv



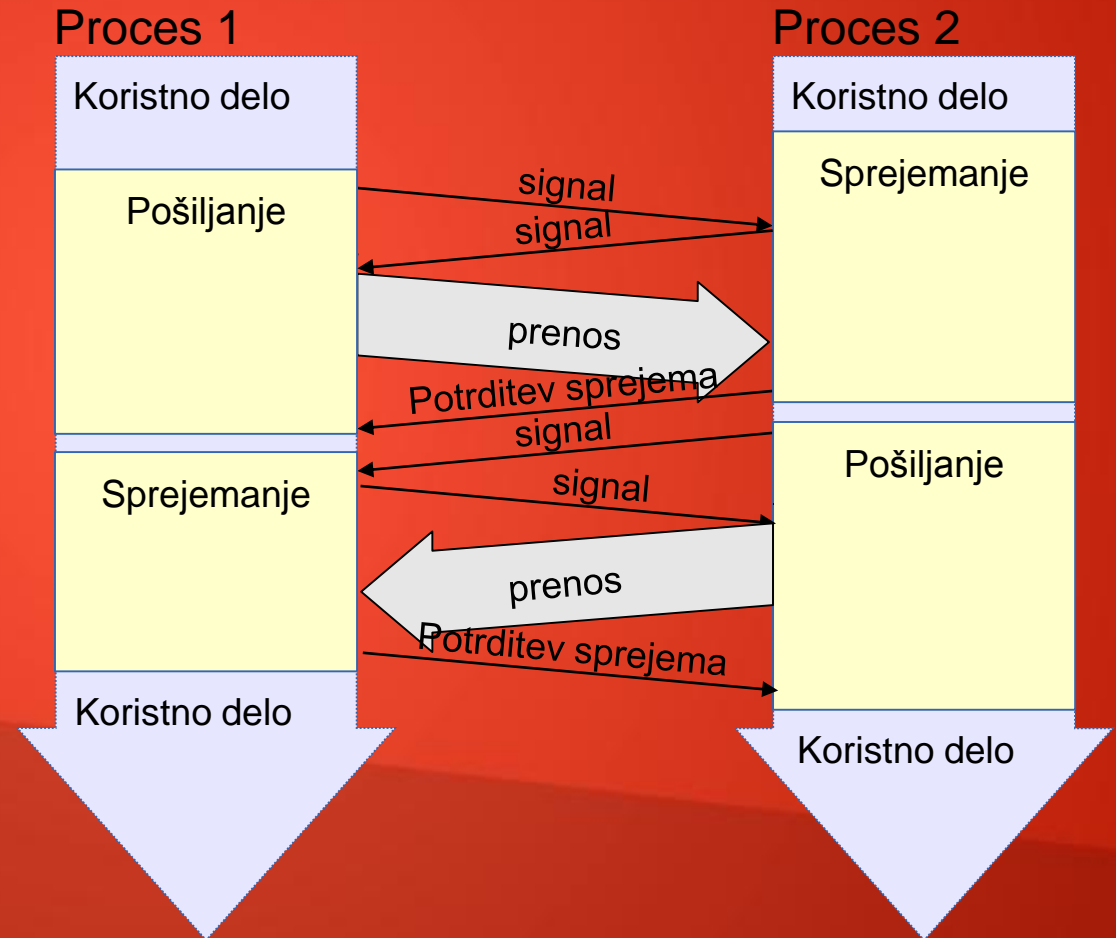
Deadlock

- Vsako pošiljanje ali sprejemanje blokira proces
- Končana komunikacija odblokira proces
- Neprevidna uporaba lahko povzroči 'obešanje' procesov
- Sporočila so prejeta v istem vrstnem redu kot so poslana



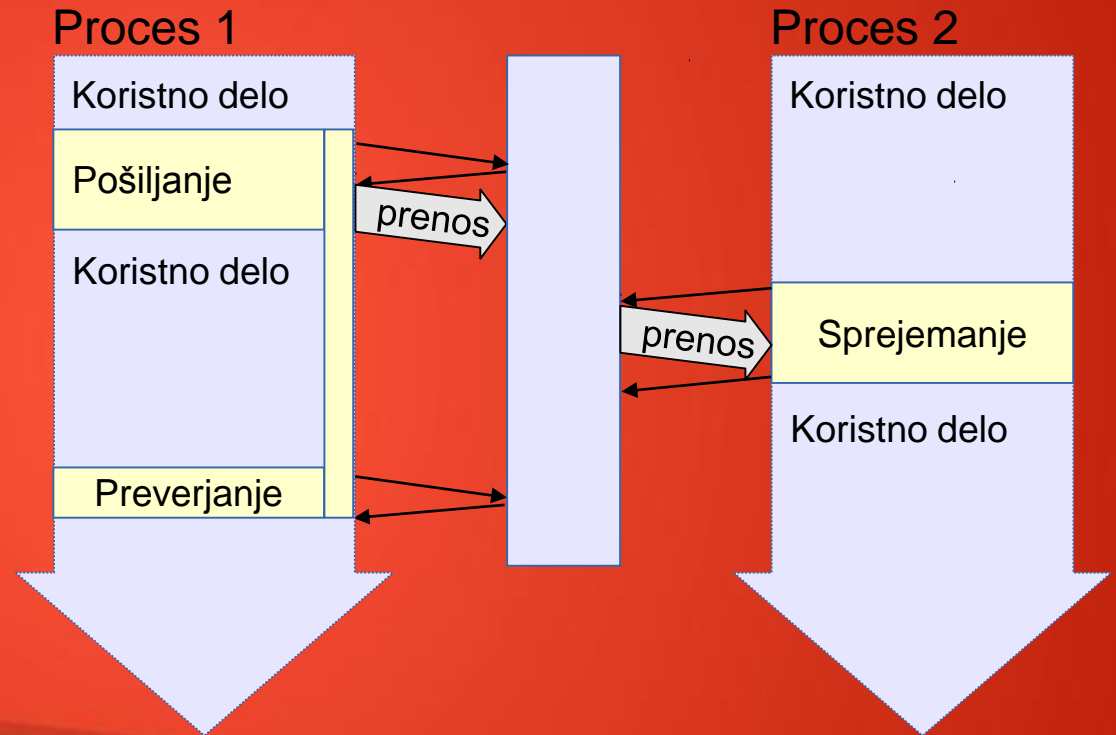
Sinhrono pošiljanje in sprejemanje

- Pošiljanja in sprejemanja si lahko sledijo – izmenjava podatkov
- Predstavljajo pogost vzorec uporabe
- MPI pozna zato optimirano funkcijo
- `MPI_Sendrecv`



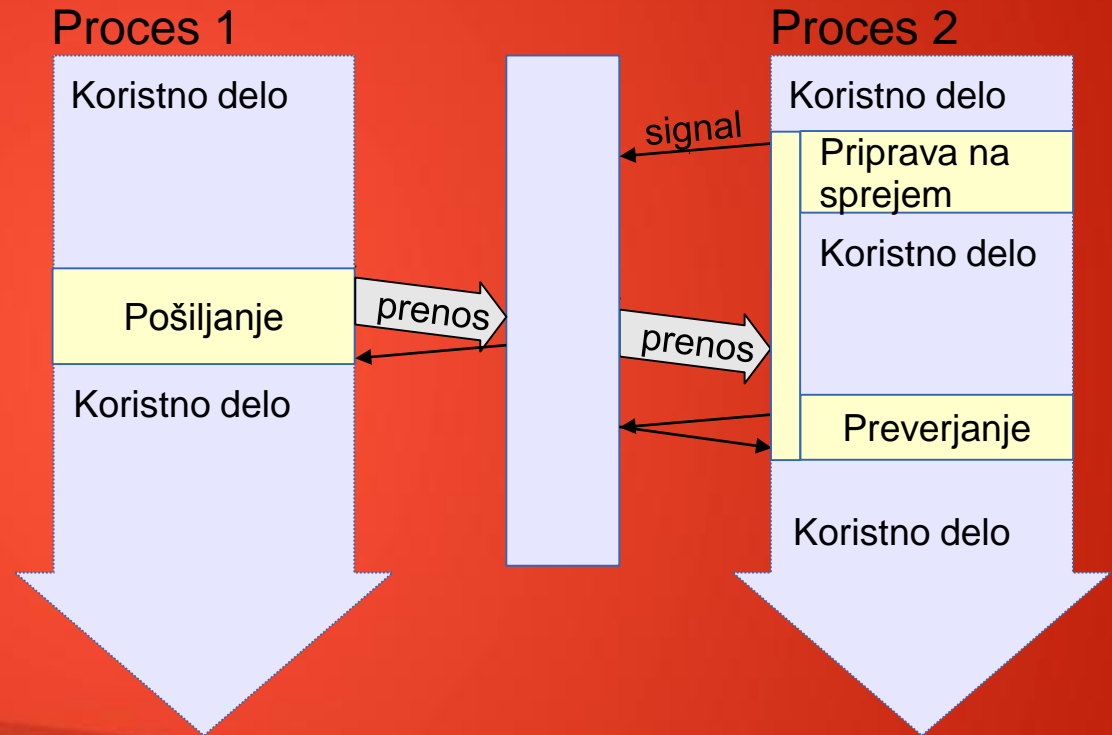
Asinhrona komunikacija

- Vmesnik (MPI), ki skrbi za hrambo sporočil v tranziciji
- Ne blokira procesa
- Sporočilo se ne sme zbrisati, dokler ni preverjeno poslano
- MPI_Isend
- MPI_Wait ali MPI_Test



Asinhrona komunikacija

- Podobno velja za asinhron sprejem sporočil
- Prostor za sporočilo mora obstajati ob pripravi
- uspešno preverjanje pomeni veljavno sporočilo v tem prostoru
- MPI_Irecv
- MPI_Wait ali MPI_Test



MPI send in receive

- Standard / Immediate + Buffered + Ready + Persistent
- MPI_Send (standardni, blokirajoči – a ne vedno!)
- MPI_Isend (I -> Immediate)
- MPI_Ibsend (B -> Buffered)
- MPI_Rsend (R -> Ready)
- MPI_Ssend (S -> Synchronous)
- MPI_Send_init (isto spremenljivko se bo pošiljalo pogosto)

* Glavna sta MPI_Send in MPI_Isend, ostalo so 'optimirane' verzije za posebne primere

Blokirajoči in neblokirajoči ukazi

- Direktno asociirani z sinhrono in asinhrono komunikacijo
- Med seboj jih lahko mešamo
- Blokirajoči – enostavni, zanesljivi, brez presenečenj
- Neblokirajoči – omogočajo prekrivanje komunikacije in računanja
- V ozadju se dogajata še dodeljevanje in sproščanje sistemskih virov – zato neblokirajoči ukazi nastopajo v paru z ukazoma Wait in Test.

Sprejemanje sporočila

- Dve skrajnosti glede pričakovanega sporočila
 - Točno določen naslovnik, tip sporočila, oznaka
 - Popolnoma nedoločen naslovnik, tip, oznaka
- Možnosti sprejema
 - Blokiranje sprejem
 - Testiranje za sprejem (ali obstaja kako sporočilo / ali obstaja točno določeno sporočilo?)
 - Čakanje na sporočilo
 - Čakanje na signal o sporočilu (probe)

Kolektivne operacije

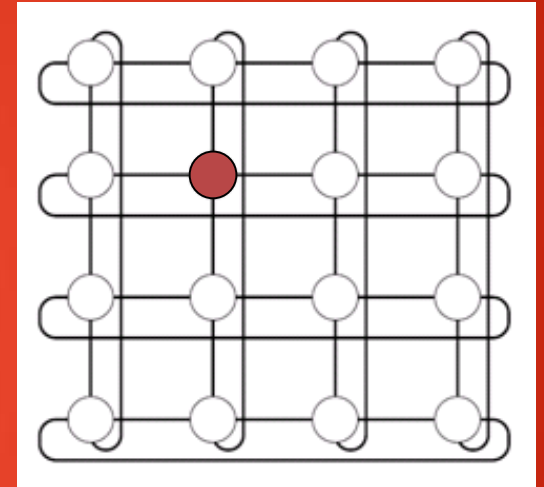
- 1 -> n
 - Broadcast
 - Scatter
- n -> 1
 - Reduce (add, mult, min, max)
 - Gather
- n -> n
 - Allgather
 - Allreduce

Operacije so lahko optimirane na topologijo omrežja

Optimiranje kolektivnih operacij

- Vzemimo primer `MPI_Scatter` in regularno mrežo kot na sliki
- Kako sploh poslati sporočilo nepovezanemu vozlišču?
- Kaj je hitrejše od

```
for (int i=1; i<n; ++i)  
    MPI_Send(..., i, ...)
```
- Bližnjim sosedom se pošlje sporočila za njih in za njihove sosede, nato oni pošljejo drugi del sporočila dalje



Asinhrona komunikacija

- Proti pričakovanjem je precej bolj zapletena
- Nove nevarnosti nepremišljenega programiranja
 - Puščanje (*leaking*) resursov
 - Neuravnovešeno pošiljanje/sprejemanje
 - Procesi ostanejo pozabjeni v pošiljanju ali čakanju
- Bolj vsestranska
 - $\text{MPI_Irecv} + \text{MPI_Wait} = \text{MPI_Recv}$
 - $\text{MPI_Isend} + \text{MPI_Wait} = \text{MPI_Send}$

MPI_Wait

- Kdaj konča?
 - Pri MPI_Isend – ko MPI ne dostopa več do pomnilnika s sporočilom
 - Pri MPI_Irecv – ko MPI sporočilo zapiše v podan pomnilnik
- Več variant – waitall, waitsome, waitany
- MPI_Test je neblokirajoča različica MPI_Wait

MPI okolje

- Komunikator
(*communicator*)
 - Definira množico procesov, ki lahko komunicirajo med seboj
 - Identifikacija procesov v komunikatorju preko ranka (*rank*)
 - MPI_COMM_WORLD je privzet komunikator
- Poročanje o napakah
 - Je vezano na komunikator
 - Funkcije vračajo *error code*
 - Privzeto je »poenostavljeno« delovanje – napake so *fatal*

MPI okolje

- Datoteka »hosts« vsebuje imena vozlišč za izvedbo programa
- Ukaz mpirun
mpirun -n 4 -hostfile hosts
./ime_programa
parametri_programa
- Funkciji MPI_Init in MPI_Finalize

Na HPCFS

- module load openmpi
- Ukaz bsub
bsub -o output.txt -e error.txt -n 4 mpirun
./ime_programa
parametri_programa

»Hello world« v MPI

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

int main(int argc, char **argv) {
    int rank, size;
    char name[MPI_MAX_PROCESSOR_NAME];
    int sLen;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Get_processor_name(name, &sLen);
    printf("%s: rank %d of %d\n", name, rank, size);
    MPI_Finalize();
}
```

- #include <mpi.h>
- Funkcije:
 - MPI_Init in MPI_Finalize
 - MPI_Comm_rank
 - MPI_Comm_size
 - MPI_Get_processor_name
- Prevedemo z *mpicc*
- Izhod:
cn52: rank 0 of 3
cn52: rank 1 of 3
cn52: rank 2 of 3

MPI send in recv

```
MPI_Send(&count,  
         1,  
         MPI_INT,  
         dest,  
         tag,  
         MPI_COMM_WORLD);
```

- pomnilnik, kjer se nahaja sporočilo
- število elementov sporočila
- tip elementa
- destinacija (rank ciljnega procesa)
- oznaka
- komunikator

```
MPI_Recv(&count,  
         1,  
         MPI_INT,  
         source,  
         tag,  
         MPI_COMM_WORLD,  
         &status);
```

- pomnilnik, kamor se bo shranilo sporočilo
- število elementov sporočila
- tip elementa
- izvor (rank izvornega procesa)
- oznaka
- komunikator
- statusna struktura (vsebuje: source, tag, error)

Nevarno sinhrono

...

```
int rank, size;
```

```
int source, dest, tag=0xf00d;
```

```
int sendBuf, recvBuf;
```

```
MPI_Status status;
```

```
MPI_Init...
```

```
source = (rank - 1) % size;
```

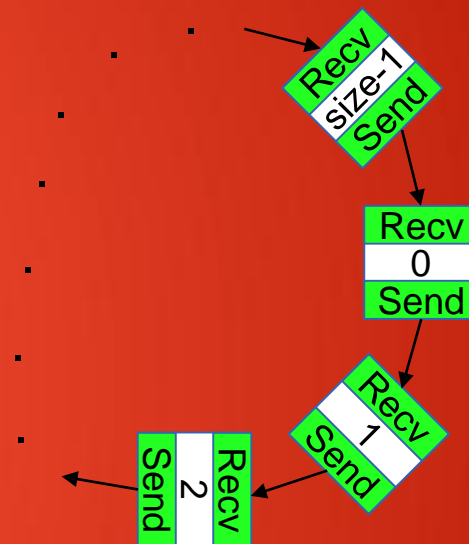
```
dest = (rank + 1) % size;
```

```
MPI_Send (&sendBuf, 1, MPI_INT, dest, tag, MPI_COMM_WORLD);
```

```
MPI_Recv (&recvBuf, 1, MPI_INT, source, tag, MPI_COMM_WORLD, &status);
```

```
printf("Process %3i sent %5i and received %5i\n", rank, sendBuf, recvBuf);
```

```
MPI_Finalize...
```



Nevarno sinhrono

- Zakaj nevarno?
 - Vsi procesi imajo isto zaporedje send in recv ukazov.
 - Lahko povzroči 'deadlock'
- Kdaj in zakaj deluje?
 - Na majhnih sporočilih
 - Ker MPI skopira majhna izhodna sporočila v svoj medpomnilnik (buffer)
- Kako elegantno obiti težavo?
 - Asinhrona sporočila
 - Vrstni red:
 - Pripravimo prostor za sprejem
 - Napovemo sprejem
 - Pošljemo sporočilo
 - Počakamo sprejem

Varno sinhrono

...

```
int rank, size, sLen;
```

```
int source, dest, tag=0xf00d;
```

```
int sendBuf, recvBuf;
```

```
MPI_Status status[2];
```

```
MPI_Request request[2];
```

```
MPI_Init...
```

```
source = (rank - 1) % size;
```

```
dest = (rank + 1) % size;
```

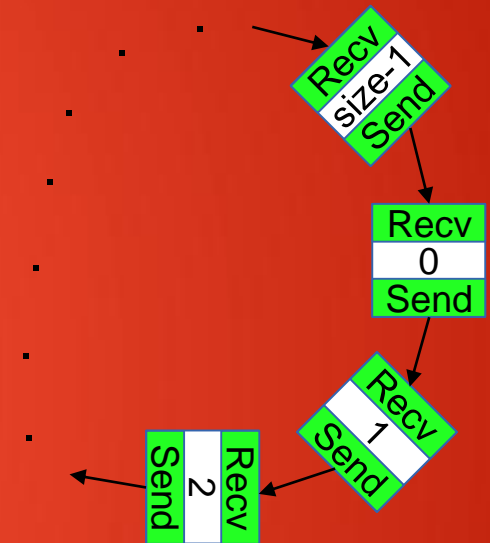
```
MPI_Irecv(&recvBuf, 1, MPI_INT, source, tag, MPI_COMM_WORLD, &request[0]);
```

```
MPI_Isend(&sendBuf, 1, MPI_INT, dest, tag, MPI_COMM_WORLD, &request[1]);
```

```
MPI_Waitall(2, request, status);
```

```
printf("Process %3i sent %5i and received %5i\n", rank, sendBuf, recvBuf);
```

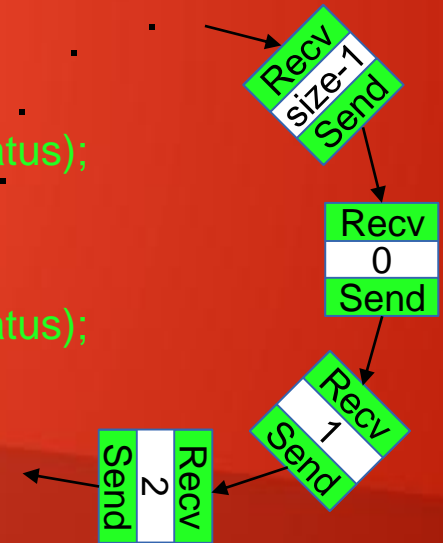
```
MPI_Finalize...
```



Varno sinhrono 2

```
...
int rank, size;
int source, dest, count, tag=0xf00d;
MPI_Status status;
MPI_Init ...
source = (rank - 1) % size;
dest = (rank + 1) % size;
if (rank == 0) {
    count = 1;
    MPI_Send(&count, 1, MPI_INT, dest, tag, MPI_COMM_WORLD);
    MPI_Recv(&count, 1, MPI_INT, source, tag, MPI_COMM_WORLD, &status);
    printf("The result of the count = %i\n", count);
} else {
    MPI_Recv(&count, 1, MPI_INT, source, tag, MPI_COMM_WORLD, &status);
    count += 1;
    MPI_Send(&count, 1, MPI_INT, dest, tag, MPI_COMM_WORLD);
}
MPI_Finalize...
```

Izhod (na 30 procesih):
The result of the count = 30



Asinhrona komunikacija

- Vektor (tabela) števil, iščemo določeno vrednost

```
vecSize = 100000000; // 100 M
int vec*;
vec = malloc(sizeof(int)*vecSize);
```

- Zaporedni program:

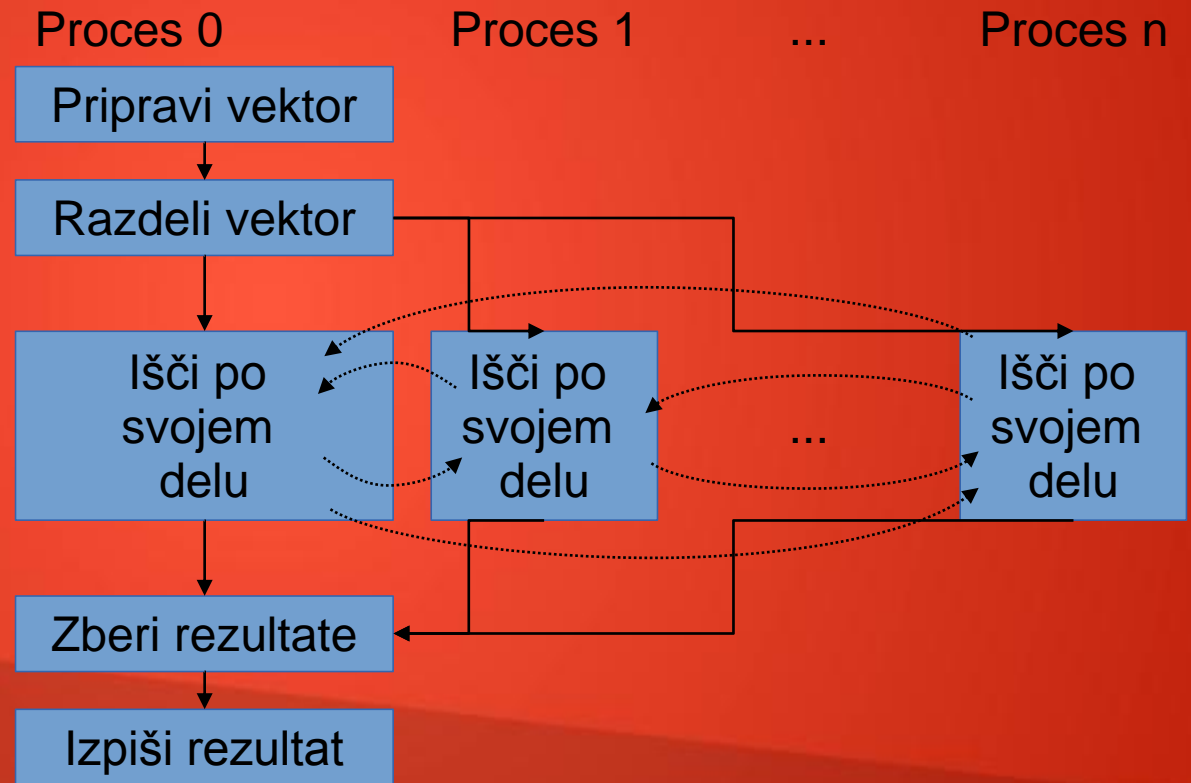
```
for (int i = 0; i < vecSize; ++i)
    if (vec[i] == vrednost) printf("%d", i);
```

- Vzporedni program:

- Razdelimo vektor med vse procese
- Vsak proces preišče svoj del vektorja
- Ko prvi konča obvesti vse ostale
- Rezultat vrne procesu 0

"Master-slave" princip

- Proces 0 je glavni
- Ostali izvajajo njegove ukaze
- Dodatno: med seboj se smejo obvestiti, da so končali



Primer za asinhrono komunikacijo

- Vhod: tabela (vektor) števil
- Problem: iskanje podane vrednosti – na katerem indeksu se nahaja določena vrednost
- Če je najdenih vrednosti več, ni pomembno katero se vrne kot rezultat
- Vektor se razdeli med procesorje
- Vsak procesor išče na svojem odseku in vrne rešitev glavnemu
 - Takoj ko nekdo najde rešitev se sme prekiniti iskanje
- Glavni zbere rešitve in vrne rezultat
 - Rezultat je lahko prva prispela rešitev

Primer za asinhrono komunikacijo

- Vsi procesorji iščejo rešitev na svojem odseku
- Vsi procesorji poslušajo (MPI_Irecv)
- Ko procesor najde rešitev, to sporoči vsem
- Procesorji ugotovijo, da je nekdo že našel rešitev (MPI_Test) in končajo svoje izvajanje
- Glavni proces počaka da vsi končajo izvajanje

Asinhrona komunikacija

```
int rank, size, tag_fail = 0xbad, tag_success = 0xcce;
MPI_Status status;
MPI_Request request, *retRequest;
int done, nValues, luckyProcess = -1;
int vecSize = 400000;
int* vec;
int i, iMax, j, result;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
retRequest = malloc(size * sizeof(MPI_Request));
vec = malloc(vecSize * sizeof(int));
if (rank == 0) {
    memset(vec, 0, sizeof(int)*vecSize);
    vec[1987654321 % vecSize] = 42;
}
```

Asinhrona komunikacija

```
MPI_Bcast(vec, vecSize, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Irecv(&result, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &request);
nValues = (vecSize+size-1)/size;
i = rank*nValues;
iMax = i + nValues;
if (iMax > vecSize)
    iMax = vecSize;
for (; i < iMax; ++i) {
    MPI_Test(&request, &done, &status);
    if (done) break;
    if (vec[i] == 42) {
        luckyProcess = rank;
        for (j=0; j<size; ++j) {
            if (j != rank)
                MPI_Isend(&i, 1, MPI_INT, j, tag_success, MPI_COMM_WORLD, &retRequest[j]);
            else
                retRequest[j] = MPI_REQUEST_NULL;
        }
        MPI_Waitall(size, retRequest, MPI_STATUSES_IGNORE);
        break;
    }
};
```

Asinhrona komunikacija

```
if (rank == 0) {
    if (!done)
        MPI_Wait(&request, &status);
    for (j = 1; j < size; ++j) {
        if (j > 1) {
            MPI_Irecv(&result, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &request);
            MPI_Wait(&request, &status);
        }
        if (status.MPI_TAG == tag_success) {
            i = result;
            luckyProcess = status.MPI_SOURCE;
        }
    }
    printf("P:%d found it at index %d\n", luckyProcess, i);
} else {
    if (luckyProcess != rank) {
        MPI_Isend(&i, 1, MPI_INT, 0, tag_fail, MPI_COMM_WORLD, &retRequest[j]);
    } else {
        MPI_Request_free(&request);
    }
}
free(retRequest);
free(vec);
MPI_Finalize();
```

Asinhrona komunikacija

- Optimizacija je draga (nepotrebno zapletena koda)
- Pokriti je treba vse možnosti:
 - Noben proces ne najde iskanega števila
 - Več procesov najde iskano število
 - Glavni / podrejeni najde
- Vsi procesi pošljejo rezultat glavnemu
 - Tisti, ki najde – vrne rezultat
 - Tisti, ki ne najde – sporoči da je končal
 - Tudi tisti, ki jih je ustavil drug proces