Porazdeljeni sistemi

8.
Večračunalniški sistemi, MPI

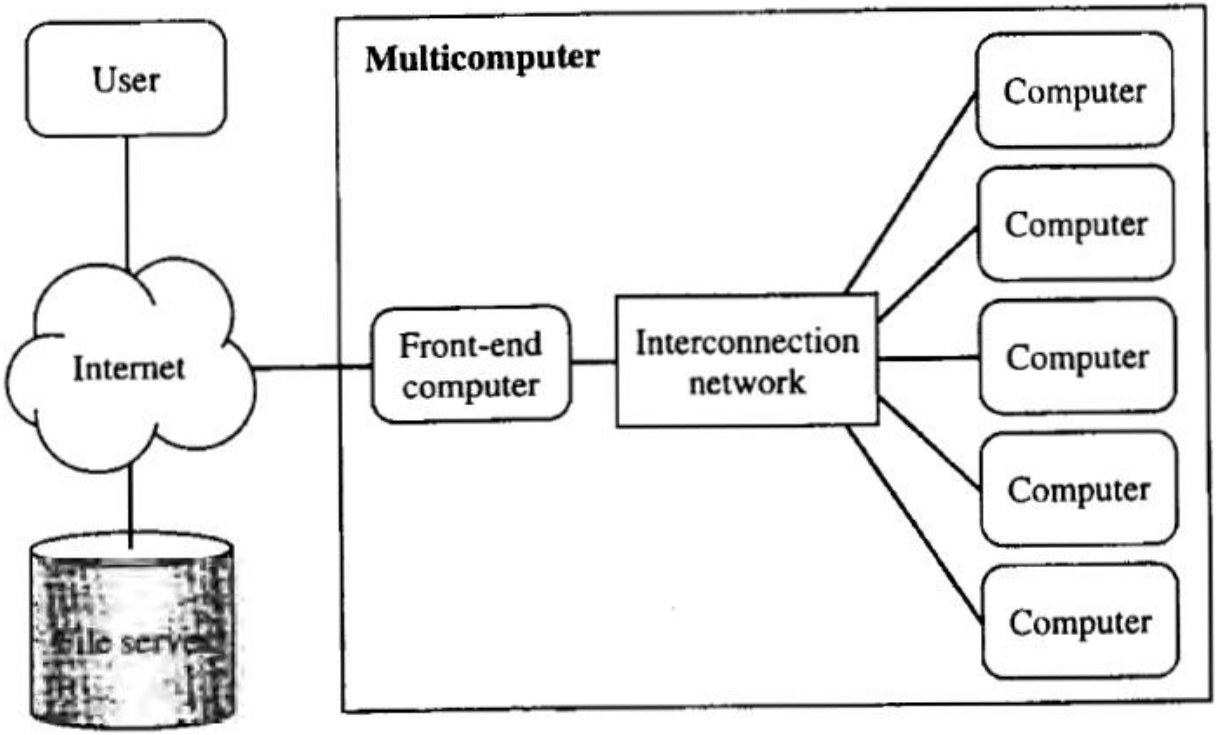
Predavatelj: izr. prof. Uroš Lotrič
Asistent: Davor Sluga

Več-računalniški sistemi

- Za razliko od sistemov z deljenim pomnilnikom so tu pomnilniki nepovezani
- Vsak procesor ima neposreden dostop samo do svojega pomnilnika
  - Isti naslov na dveh procesorjih tako pomeni dve fizično različni pomnilniški lokaciji
- Ker nimajo skupnega pomnilnika, procesorji med seboj komunicirajo s pošiljanjem sporočil
- Rešitve:
  - Namenske:
    - posebno preklopno omrežje z nizko latenco in veliko pasovno širino
    - dobro razmerje med hitrostjo procesorjev in zmožnostjo omrežja
  - Splošne:
    - Običajne masovno proizvedene komponente (procesorji, omrežje)
    - Cenovno zelo ugodne rešitve

Več-računalniški sistemi

- Nesimetrični sistemi
  - Zgrajeni podobno kot procesna polja
    - Front-end za interakcijo z uporabnikom
    - Back-end za računanje

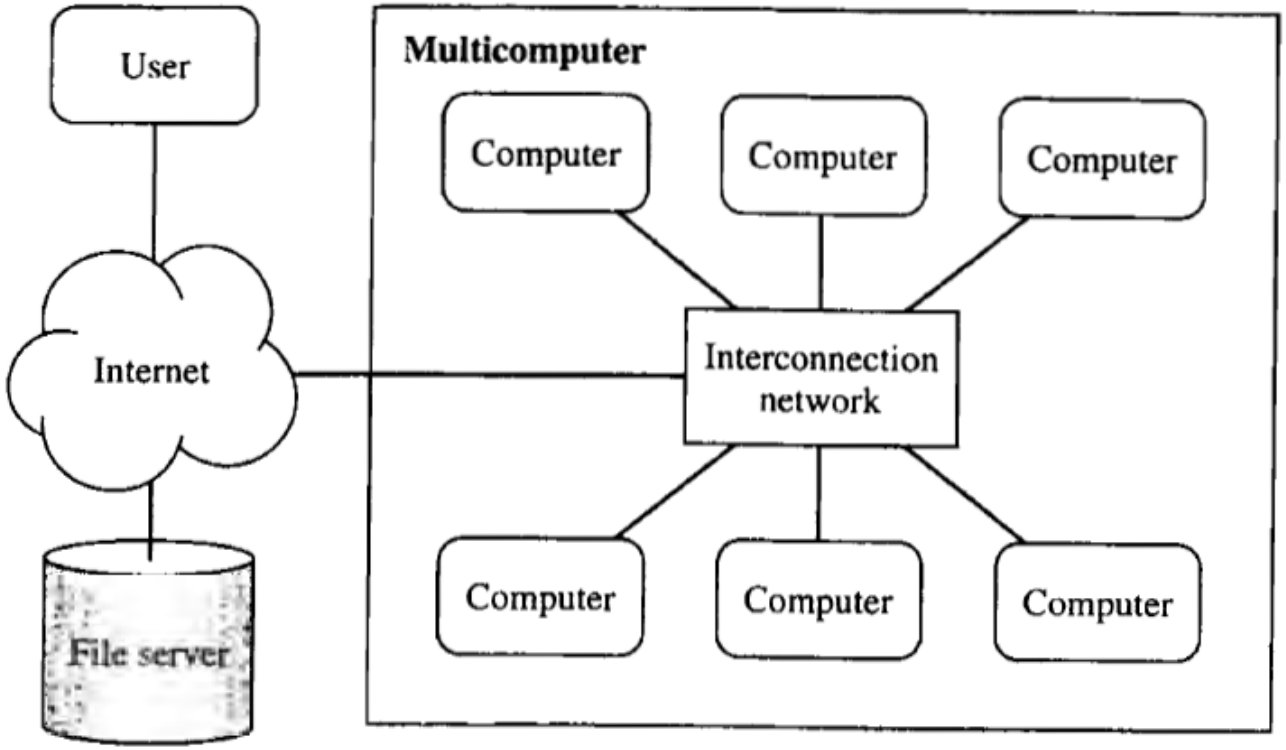


Več-računalniški sistemi

- Nesimetrični sistemi
  - Zgrajeni podobno kot procesna polja
    - Front-end za interakcijo z uporabnikom
    - Back-end za računanje
  - +
    - Ker so namenjeni izključno za izvajanje paralelnih programov imajo na back-end računalnikih nameščen poseben enostaven operacijski sistem
  - -
    - Če odpove front-end, ne moremo uporabljati back-enda
    - Obremenjenost front-end računalnika
    - Težko razhroščevanje programov
    - Dva programa: posebej za front-end in posebej za back-end

Več-računalniški sistemi

- Simetrični sistemi
  - Vsi računalniki imajo enako funkcionalnost

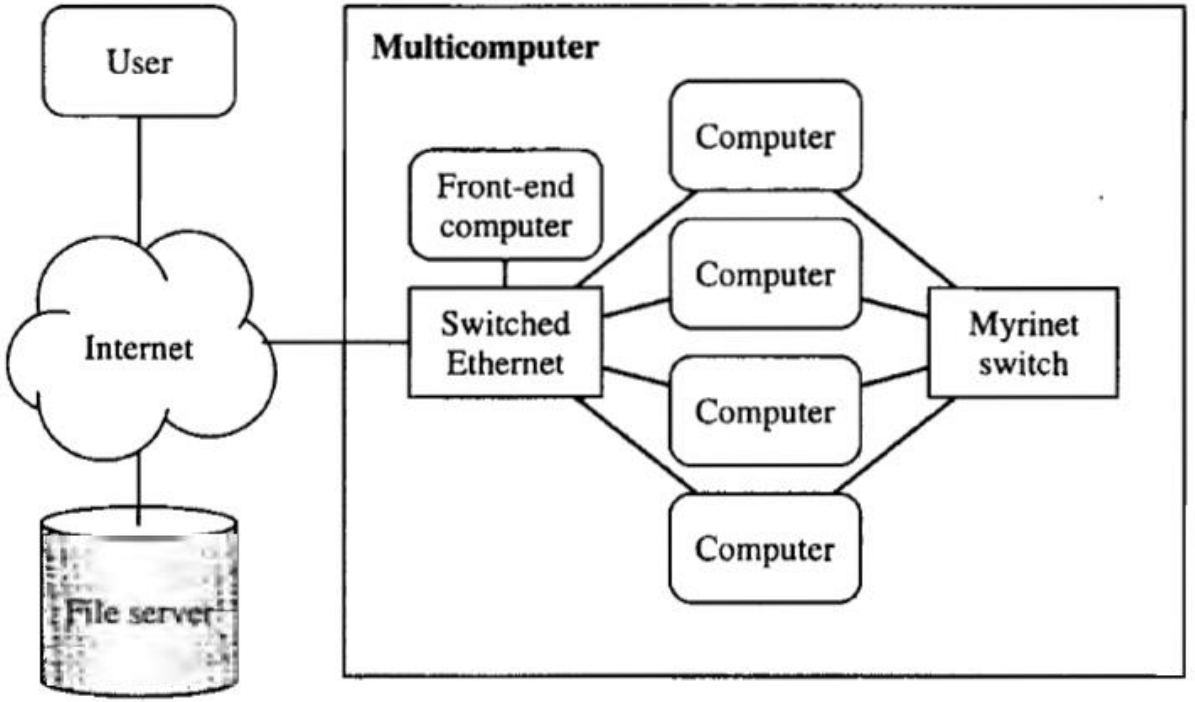


Več-računalniški sistemi

- Simetrični sistemi
  - +
    - Vsi uporabljajo enak operacijski sistem
    - Ni težav s preobremenjenostjo posameznega sistema – če je en zaseden, uporabimo drugega
    - Lažje razhroščevanje (isti OS)
    - Vsi izvajajo isti program
    - Lahko se izvaja hkrati več različnih programov
  - -
    - Stvar ne zgleda kot en sam paralelni stroj
    - Težko je dobro uravnotežiti porazdeljevanje opravil med procesorje
    - Različni programi tekmujejo za vire
      - Več procesov si deli isti procesor → več zgrešitkov v predpomnilniku
      - Počasnejše delovanje

Več-računalniški sistemi

- Najboljša rešitev je mešanica obeh
  - Običajno so vsi računalniki povezani z običajno mrežno povezavo v internet
  - Za medsebojno komunikacijo med računanjem uporabljajo posebno omrežje
    - Myrinet
    - Infiniband

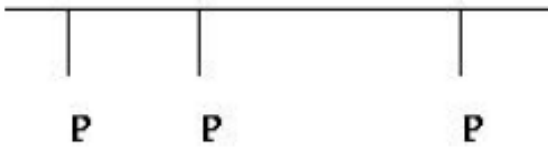


Več-računalniški sistemi

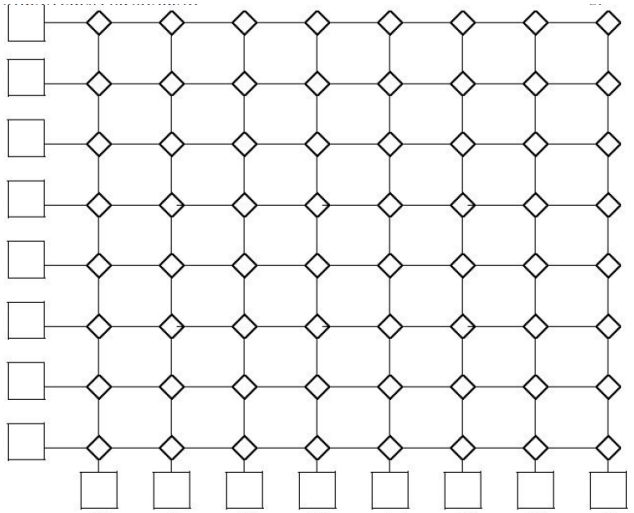
- Gruče in omrežja delovnih postaj
  - Gruče
    - Namenjene so izključno za računanje
    - Računalniki so nameščeni v bližini
    - Strojna in programska oprema kar se da poenotena
  - Omrežja delovnih postaj
    - Namenjena za interaktivno delo z računalnikom, proste vire ponujajo za uporabo drugim
    - Računalniki so razpršeni po organizaciji, območju, svetu (Seti@home)
    - Različni operacijski sistemi, različni uporabniški programi

Povezovalna omrežja

- Večprocesorski sistemi
  - vodila
    - žice si delijo vse naprave,
    - nizka cena, fleksibilnost
    - samo dve napravi na enkrat lahko komunicirata,
    - Več kot je procesorjev, bolj čakajo na dostop do pomnilnika

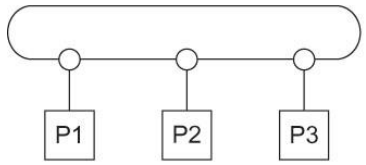


  - Mreža
    - Preklopni sistem
    - Poti med napravami se nastavlja s stikali
      - Kvadrati – n x procesor / n x pomnilnik
      - Diamant – stikala n*n možnih poti
      - Za manjše sisteme
      - n hkratnih povezav

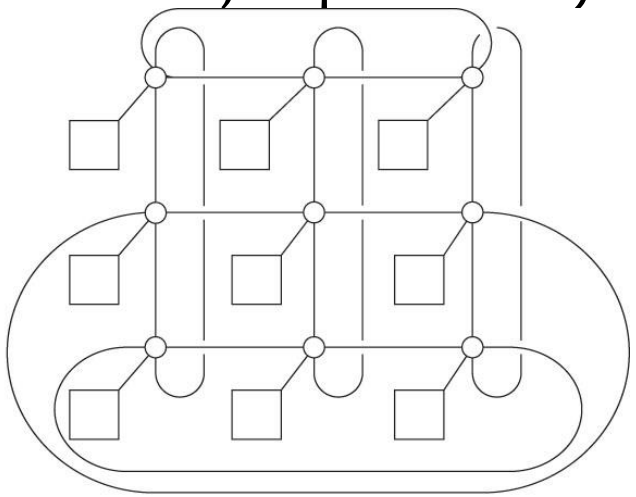


Povezovalna omrežja

- Več-računalniški sistemi: neposredne povezave
  - Vsako stikalo je neposredno vezano na par procesor/pomnilnik
  - dvosmerne povezave
  - Idealna: polno povezana mreža (vsak z vsakim, nepraktična)
  - Obroč
    - Je boljši od vodila – več hkratnih povezav (z dvema sosedoma)
  - 2D torus
    - Še boljši (povezava s štirimi sosedi)

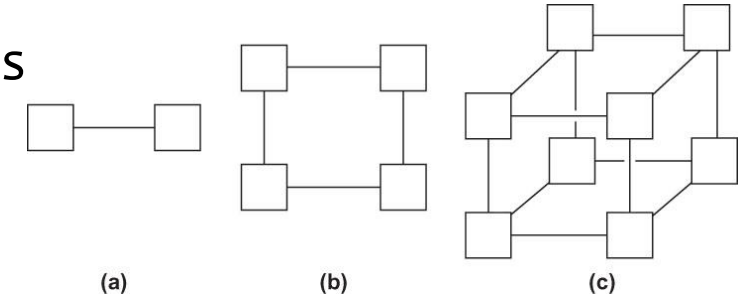


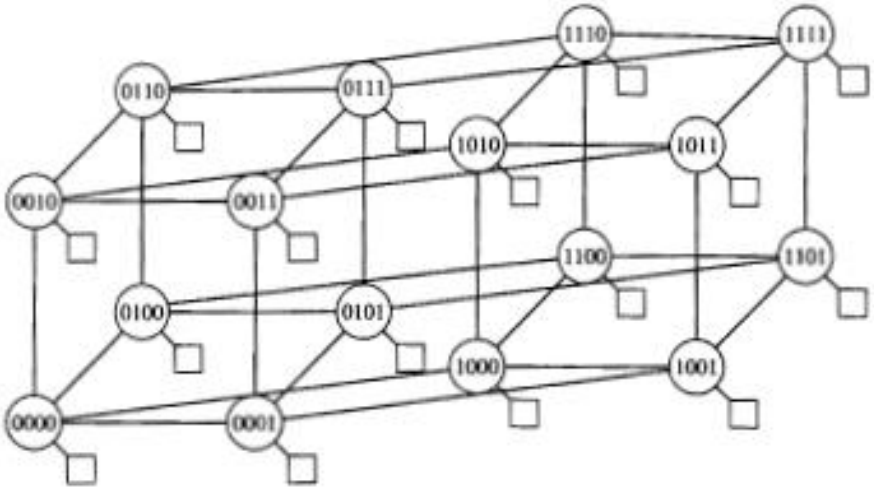
(a)


(b)

Povezovalna omrežja

- Več-računalniški sistemi: neposredne povezave
  - Hiperkocka
    - Več povezav kot obroč ali torus
    - Zmogljivejša stikala
    - (precej uporabljana)
    - n = 2^d procesorjev, n stikal
    - Kako poslati sporočilo iz vozlišča 0101 na vozlišče 001
      - Razlika je na dveh mestih → preko dveh vozlišč
      - 0101 → 0001 → 0011 ali 0101 → 0111 → 0011

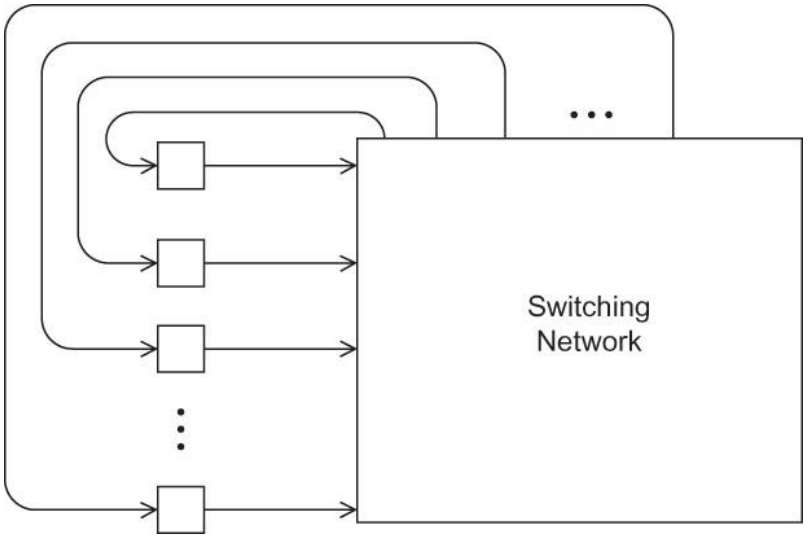




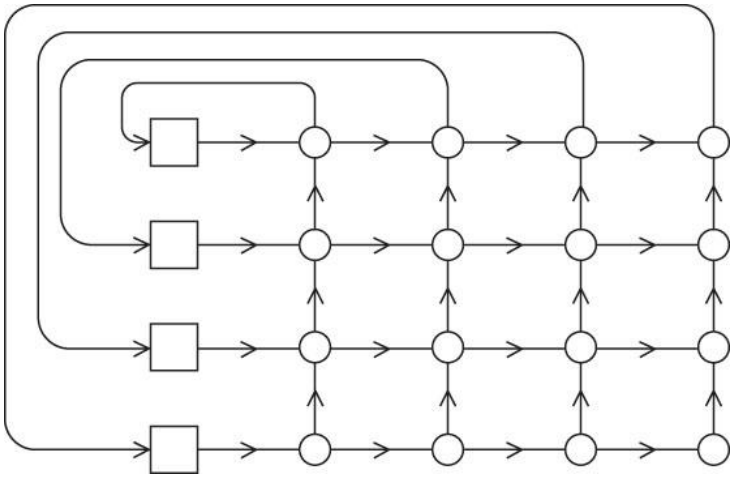
Povezovalna omrežja

- Več-računalniški sistemi:

  preklopna omrežja
  - Stikala niso direktno vezana na procesor
  - Enosmerne povezave



Mreža:
Enosmerne povezave, poljubni pari procesorjev p*p stikal

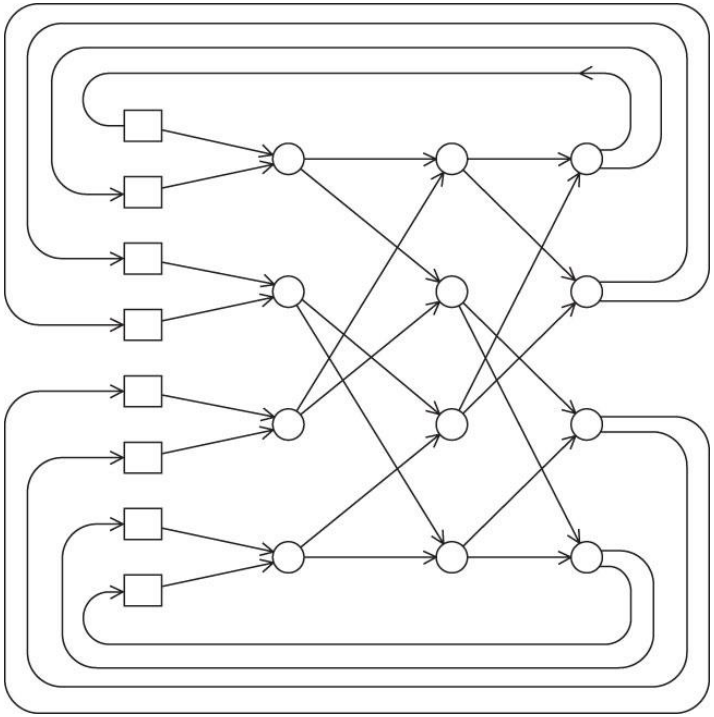


Povezovalna omrežja

- Več-računalniški sistemi:

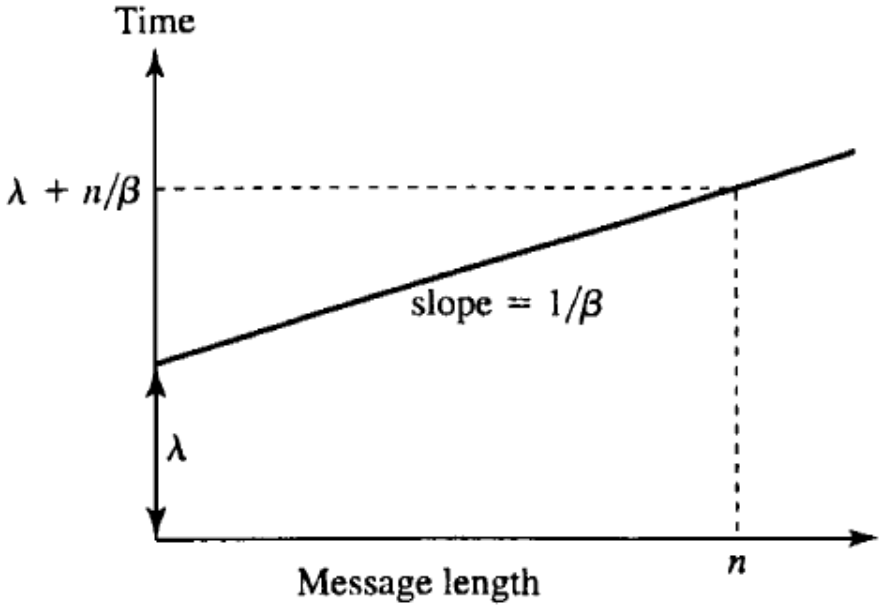
  preklopna omrežja
  - Stikala niso direktno vezana na procesor
  - Omega
    - Če 0 pošilja 6,

      potem ne more 1 pošiljati 7
    - Cenejša izvedba 2*p*log(p) stikal



Povezovalna omrežja

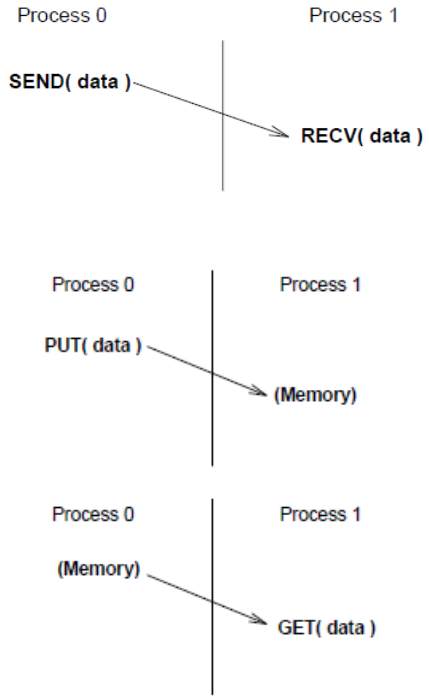
- Cena pošiljanja sporočil po mreži (Gigabit Ethernet)
  - Latenca (λ): 100 us
  - Pasovna širina (β): 1000 Mbit/s
  - Čas prenosa: $t_p = \lambda + \beta^{-1} \cdot [\text{velikost paketa}]$



Delitev paralelnega računanja

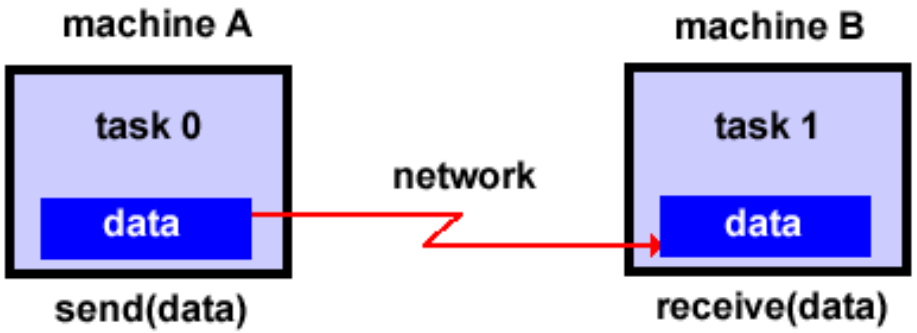
- Strojna oprema
  - Deljeni pomnilnik
  - Porazdeljeni pomnilnik
  - Oboje se uporablja pri SIMD/MIMD programskih modelih

- Komunikacije
  - Sodelovanje:
    - Pri pošiljanju in sprejemanju sporočil sodelujeta oba partnerja
    - Vsaka sprememba v spominu sprejemnika je narejena z njegovim sodelovanjem
  - Enostransko pošiljanje:
    - Pisanje in branje iz oddaljenega pomnilnika
    - Ni treba čakati na drug proces (+)
    - Sinhronizacija je lahko enostavna ali zelo zapletena (-)



Več-računalniški sistemi: paralelni programi

- Model izmenjevanja sporočil
  - Programer mora sporočila eksplicitno poskrbeti za oddajanje in sprejemanje sporočil



Več-računalniški sistemi: paralelni programi

- Model izmenjevanja sporočil
  - Program se razdeli na več procesov, ki uporabljajo vsak svoj lokalni pomnilnik
  - Procesi izmenjujejo podatke s komunikacijo, ki vključuje oddajanje in sprejemanje sporočil
  - Več procesov se lahko izvaja na enem samem fizičnem računalniku kot tudi na poljubnem številu različnih računalnikov

Več-računalniški sistemi: paralelni programi

- Model izmenjevanja sporočil
  - Običajno lahko te modele uporabljamo v obliki knjižnic in pripadajočega izvajalnega okolja
  - Razmah po 1980
  - Leta 1992 je ustanovljen MPI Forum, katerega osnovna naloga je standardizacija modelov za izmenjavanje sporočil

MPI: uvod

- MPI – Message Passing Interface
  - je specifikacija za razvijalce in uporabnike knjižnic. Ne gre za točno določeno knjižnico!
  - MPI je *de facto* standard za izmenjevanje sporočil
  - Definirana je za jezike C/C++ in Fortran
  - Zgodovina
    - MPI-1: 1994
    - MPI-2: 1996

MPI: uvod

- Razlogi za uporabo MPI
  - Zrel in dobro razumljen produkt
  - Dobro prilagojen na strojno opremo
  - Uporabljen v mnogih aplikacijah
  - Podprt s standardom
  - Prenosljiv na nivoju kode
  - Množica uporabnih funkcij v knjižnicah
  - Je zelo dostopen

MPI: uvod

- Kaj nam ponuja?
  - Komunikacija točka-točka
    - Strukturirana sporočila, lahko z lastnimi podatkovnimi tipi
    - Režimi
      - Sinhronizacija
      - Predpomnenje
  - Skupinska komunikacija
    - Vgrajene operacije in operacije, ki jih definira uporabnik
    - Množica funkcij za prenos podatkov
    - Definiranje podskupin
  - Topologija je lahko prilagojena aplikaciji
    - Vgrajena podpora za mreže in grafe
  - Spremljanje delovanja
    - Možnost za opazovanje sistema z drugimi orodji

MPI: uvod

- Je velik ali majhen?
  - Je velik
    - MPI-1 128 funkcij, MPI-2 152 funkcij
  - Je majhen
    - Večino problemov rešimo s 6 funkcijami
  - Je ravno pravšen
    - Za uporabo nam ni treba poznati vsega
    - Kadar potrebujemo več, lahko poiščemo in izkoristimo

MPI: uvod

- Paralelizem po specifikaciji MPI je ekspliciten. Programer je odgovoren za
  - identifikacijo paralelnih delov algoritma in
  - implementacijo algoritma z uporabo konstruktov MPI
- Po specifikaciji MPI-1 je število procesov statično, po specifikaciji MPI-2 pa se jih da dinamično spreminjati
- Za prenos podatkov je potrebno sodelovanje vseh vpletenih procesov
  - oddajanje sporočila s strani enega procesa mora dopolnjevati sprejemanje sporočila s strani ostalih procesov

MPI: kje začeti?

- Programska oprema
  - MPICH (http://www-unix.mcs.anl.gov/mpi)
  - LAM/MPI (http://www.lam-mpi.org
  - DeinoMPI (http://mpi.deino.net)
- Razširitve za druge programske jezike
  - Niso standardizirane
  - MPIJava (http://aspen.ucs.indiana.edu/pss/HPJava/mpiJava.html)
  - MPIPython (http://code.google.com/p/mpi4py/)
  - MPI.NET (http://www.osl.iu.edu/research/mpi.net)

MPI: kje začeti?

- Uporabnik poskrbi, da ima vsak proces dostop do
  - izvršilne datoteke, vhodnih datotek in izhodnih datotek; dve rešitvi:
    - dostop do mrežnih virov (network share)
    - kopiranje datotek na vozlišča in nazaj
  - Procesi morajo imeti pravico, da se zaženejo preko mreže (izklopljen ali ustrezno nastavljen požarni zid)
- S klicem ukaza *mpiexec (mpirun)* vsak proces začne izvajati svoj program
- Običajno je program en sam
  - Za različne funkcionalnosti programa na različnih procesih poskrbimo z vejitvami na podlagi oznake procesa

MPI: šest osnovnih funkcij

- Preprost MPI program

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

int main(int argc, char **argv)
{
    int size, rank;

    MPI_Init(&argc, &argv);

    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    printf("Sem proces %d od %d.\n", rank, size);

    MPI_Finalize();

    return 0;
}
```

MPI: šest osnovnih funkcij

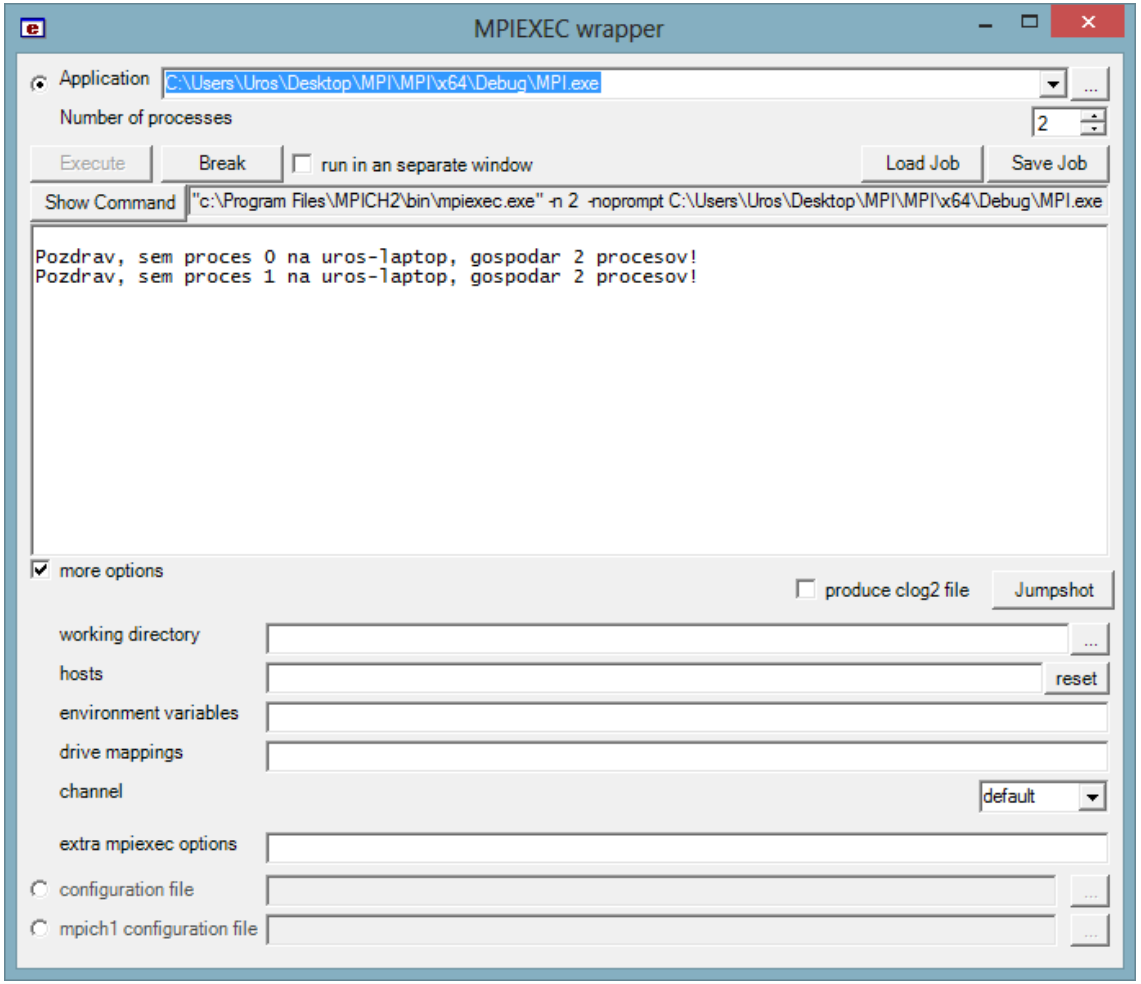
- Preprost MPI program
  - Zaglavje `"mpi.h"` definira funkcije, konstante in podatkovne tipe
  - Skoraj vse funkcije MPI vračajo celoštevilčno kodo napake
  - Vsa imena v knjižnici MPI se začnejo s predpono MPI_
    - Konstante so zapisane z velikimi črkami
    - Podatkovni tipi in funkcije imajo veliko začetnico za predpono MPI, ostale črke so male
  - Funkcije, ki niso iz knjižnice MPI, se izvajajo lokalno.
    - Funkcija `printf` v prejšnjem primeru tako teče na vsakem procesu.
    - Okolje MPI omogoča zbiranje sporočil iz vseh vozlišč.

MPI: šest osnovnih funkcij

- Preprost MPI program
  - `int MPI_Init(int *argc, char **argv);`
    - Inicializira okolje MPI in vzpostavi povezave med procesi.
    - Pred njo ne smemo poklicati nobene druge funkcije MPI
    - Argumenta argc in argv uporabi zato, da iz ukazne vrstice prebere posebne nastavitve okolja
    - Argumente iz ukazne vrstice zagotovo dobi samo proces 0!
  - `int MPI_Finalize(void)`
    - Zapre povezave
    - Je zadnja funkcija MPI, ki jo pokličemo.
    - Za njo ne smemo uporabiti nobene funkcije iz knjižnice MPI

MPI: šest osnovnih funkcij

- MPICH2
  - http://www.mpich.org

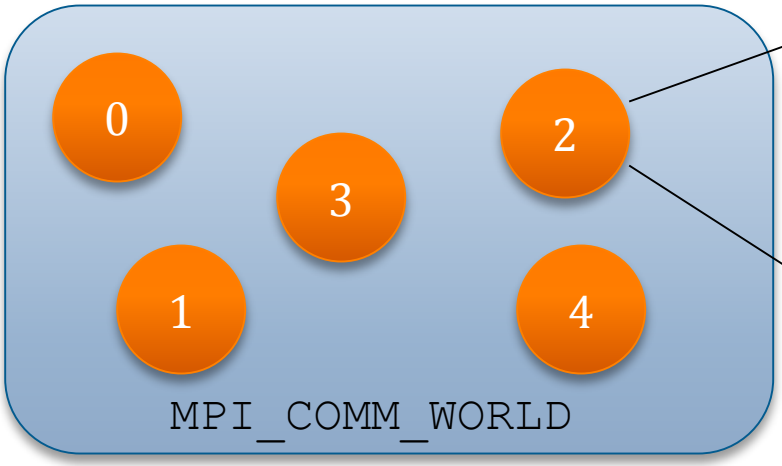


MPI: šest osnovnih funkcij

- Kako procesom razdeliti naloge?
  - `int MPI_Comm_size(MPI_Comm comm, int *size)`
    - pove na koliko procesih se izvaja program
  - `int MPI_Comm_rank(MPI_Comm comm, int *rank)`
    - Pove oznako izbranega procesa, rank = 0… size-1
- Komunikator
  - Je skupina procesov, ki se med seboj lahko izmenjuje sporočila
  - Lahko definiramo več komunikatorjev, ki predstavljajo ločene podskupine procesov
  - Privzeti komunikator `MPI_COMM_WORLD` vključuje vse procese
  - V komunikatorju se procesi ločujejo po ranku

MPI: šest osnovnih funkcij

- Komunikator
  - Primer




```
MPI_Comm_size(MPI_COMM_WORLD, &world);
// vrne world = 5

MPI_Comm_rank(MPI_COMM_WORLD,  &id);
// vrne id = 2
```

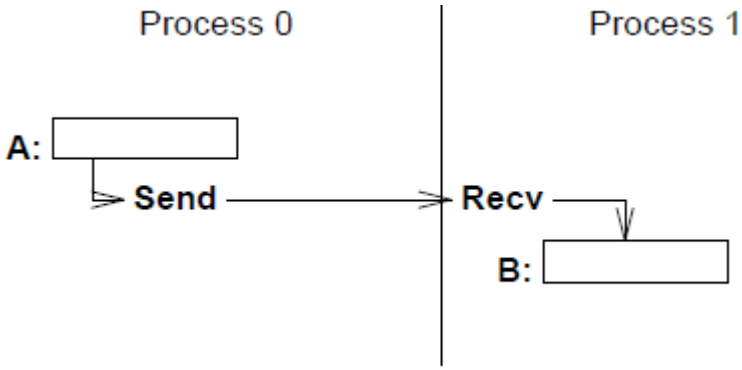
MPI: šest osnovnih funkcij

- Prvi primer

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

int main(int argc, char **argv)
{
	int rank, size;

	MPI_Init(&argc, &argv);
	MPI_Comm_size(MPI_COMM_WORLD, &size);
	MPI_Comm_rank(MPI_COMM_WORLD, &rank);
	printf("Sem proces %d od %d.\n", rank, size);
	MPI_Finalize();

	return 0;
}
```

MPI: šest osnovnih funkcij

- Pošiljanje in sprejemanje sporočil
  - En proces (A) odda sporočilo, drugi ga sprejme (B)



  - Komu je namenjeno sporočilo?
  - Kje so oddani in kje prejeti podatki?
  - Koliko podatkov pošiljamo?
  - Kakšen tip podatkov pošiljamo?
  - Kako sprejemnik spozna podatke?

MPI: šest osnovnih funkcij

- Pošiljanje in sprejemanje sporočil
  - Pri pošiljanju se navede začetni naslov, podatkovni tip in število podatkov
  - Podatkovni tipi so lahko
    - osnovni, podprti v programskih jezikih,
    - sestavljeni kot
      - neprekinjeno polje nekega podatkovnega tipa,
      - razpršena polja,
      - splošne strukture
  - Zakaj uvedba lastnih podatkovnih tipov in podajanje števila podatkov namesto dolžine polja?
    - Različni sistemi imajo različne predstavitve števil
    - S podporo osnovnih podatkovnih tipov v MPI je lažje zagotoviti prenosljivost med različnimi sistemi
    - Ker že navedemo podatkovni tip, je dovolj da dodamo samo še število elementov, ki jih želimo prenesti

MPI: šest osnovnih funkcij

- Pošiljanje in sprejemanje sporočil
  - Podatkovni tipi

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	

MPI: šest osnovnih funkcij

- Pošiljanje in sprejemanje sporočil
  - Ciljni proces je določen s komunikatorjem in rankom
  - Kot proces, od katerega želimo sprejeti sporočilo, lahko navedemo tudi `MPI_ANY_SOURCE`
  - Označevanje sporočil
    - Vsako sporočilo označimo z označbo (tag)
    - Označbe so cela števila v območju od 0… 32767
    - Z nadzorom nad označbami lahko poskrbimo za pravilen vrstni red prihajajočih sporočil
    - Kot označbo sporočila, ki ga želimo sprejeti, lahko navedemo tudi `MPI_ANY_TAG`

MPI: šest osnovnih funkcij

- Pošiljanje in sprejemanje sporočil
  - Funkcija za pošiljanje

```
MPI_Send(void *message,
         int count,
         MPI_Datatype datatype,
         int destination,
         int tag,
         MPI_Comm comm);
```

    - `message` – kazalec na sporočilo v pomnilniku
    - `count` – število elementov v sporočilu
    - `datatype` – podatkovni tip elementov
    - `destination` –proces, ki mu je sporočilo namenjeno
    - `tag` – označba sporočila
    - `comm` komunikator

MPI: šest osnovnih funkcij

- Pošiljanje in sprejemanje sporočil
  - Funkcija za sprejemanje

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

    - `message` – kazalec na sporočilo v pomnilniku
    - `count` – število elementov v sporočilu
    - `datatype` – podatkovni tip elementov
    - `source` – proces, ki je poslal podatke
    - `tag` – označba sporočila
    - `comm` komunikator
    - `status` status

MPI: šest osnovnih funkcij

- Pošiljanje in sprejemanje sporočil
  - Status

```
MPI_Status status;
MPI_Recv(..., &status);

recv_tag = status.MPI_TAG;
recv_source = status.MPI_SOURCE;
```

  - Če nas status ne zanima, lahko pri klicu funkcije uporabimo `MPI_STATUS_IGNORE`
  - Vrednosti `status.MPI_TAG` in `status.MPI_SOURCE` sta uporabni v primeru, ko pri klicu funkcije `MPI_recv` uporabimo `MPI_ANY_TAG` ali `MPI_ANY_SOURCE`.

MPI: šest osnovnih funkcij

- Drugi primer

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

#define BUF_SIZE    80

int main(int argc, char* argv[])
{
    int     myid, size;
    char    buffer[BUF_SIZE];
    int     i;

    MPI_Init(&argc, &argv);

    MPI_Comm_rank(MPI_COMM_WORLD, &myid);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
```

MPI: šest osnovnih funkcij

- Drugi primer (nadaljevanje)

```
    if (myid == 0)
    {
        printf("Pozdrav, sem proces %d, gospodar %d procesov.\n", myid, size);
        for (i=1; i<size; i++)
        {
            MPI_Recv(buffer, BUF_SIZE, MPI_CHAR,
                     MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD,
                     MPI_STATUS_IGNORE);
            printf("%s", buffer);
        }
    }
    else
    {
        sprintf(buffer, "Pozdravljen, sem proces %d\n", myid);
        MPI_Send(buffer, strlen(buffer)+1, MPI_CHAR,
                 0, myid, MPI_COMM_WORLD);
    }

    MPI_Finalize();

    return 0;
}
```

MPI: šest osnovnih funkcij

- Mnogokrat nam zadošča že spodnjih šest funkcij
  - `MPI_Init`
  - `MPI_Comm_size`
  - `MPI_Comm_rank`
  - `MPI_Send`
  - `MPI_Recv`
  - `MPI_Finalize`
- MPI je preprost!