Modele paralele SPMD

Modelul SPMD

Comunicarea prin mesaje

Message Passing Interface

Modelul SPMD

Modelul SPMD (Single Program Multiple Data)

- Acesta este un model adecvat calculatoarelor MIMD
- In cele ce urmeaza tratam doar cazul MIMD cu memorie distribuita
- Paradigma SPMD (Single Program Multiple Data): toate procesoarele executa acelasi program, dar fiecare utilizeaza un set propriu de date;
- Important de retinut:
  - Executia instructiunilor nu este sincrona, fiecare procesor executa instructiunile in ritmul sau propriu;
  - Nu toate procesoarele executa aceleasi instructiuni; programele incarcate in memoriile diferitelor procesoare sunt identice dar pot exista diferente la executie.

Adresa unui procesor

- Un program se executa pe *p* procesoare ale unui calculator paralel;
- Valoarea lui *p* este mai mica decat numarul total de procesoare al calculatorului;
- Procesoarele sunt numerotate de la 0 la *p*-1;
- Numerotarea nu este statica, se face in momentul lansarii in executie;
- Pocesoarele pot sa isi afle propria adresa printr-o primitiva a sistemului de operare sau de biblioteca
- Procesoarele pot executa instructiuni diferite in functie de adresa lor
- In descrierea algoritmilor, adresa procesorului care executa programul e utilizata ca o variabila.

Modelul SPMD

Diferentierea la executie

- Notam ***id*** adresa proprie a unui procesor
- **Exemplu** simplu de program

```
(1) id ← adresa proprie
(2) daca id = 3 atunci
        1.  a ←1
(3) altfel
        1.a ← 0
```

- Doar procesorul P_3 executa instructiunea: a ←1
- Toate celelalte executa: a ← 0
- Variabila a este in memoria locala a fiecarui procesor.

Variabile si indici

O variabila a unui program SPMD este multiplicata; fiecare procesor contine un exemplar;

Un procesor nu poate modifica variabilele din memoria altui procesor;

In exemplul de mai sus, se poate interpreta variabilaa cu un vector cu p elemente.

Se vor folosi numai indici globali de programare

Comunicatie prin mesaje

Modele de comunicatie prin mesaje

- Acesta este un model exclusiv pentru calculatoare MIMD cu memorie distribuita
- Deoarece fiecare procesor are propria sa memorie proprie, singura modalitate de comunicare intre procesoare este transmiterea de mesaje;
- Operatia de baza este atunci cand un procesor sursa P_s transmite un mesaj M continand date din memoria sa M_s unui procesor destinatie P_d, care stocheaza datele in memoria sa M_d.

Primitive de baza

- Sunt suficiente doar doua rutine pentru a descrie orice operatie de comunicatie
- Procesorul sursa transmite prin rutina ***send***
- Procesorul destinatie receptioneaza prin rutina ***recv***
- Sintaxa este:

  send(date, dest);

  recv(date, sursa);
- Date este o variabila locala, indicand locul in care se afla datele transmise sau unde se memoreaza datele receptionate; lungimea datelor rezulta din context;
- Se presupune implicit ca datele ocupa o zona continua de memorie;

Comunicatie prin mesaje

Primitive de baza

- Al doilea parametru identifica vecinul cu care se comunica
- Vecinul se noteaza prin adresa (la fel ca im MPI)
- Alternativ, se precizeaza directia in care se gaseste vecinul (daca se precizeaza o topologie a retelei de comunicatie):
  - Pe inel, gila sau tor: stanga, dreapta, sus, jos, vest, est, nord, sud;
  - In cazul hipercublui: dimensiunea pe care se comunica, un numar intre 0 si d-1;

Corectitudinea comunicatiei

- Orice operatie de transmitere a unui procesor este insotita de una de receptie a unui vecin al sau;
- Primitivele ***send*** si ***recv*** trebuie sa apara in perechi, pe ansamblul procesoarelor;
- **Exemplu:** (presupunem ca un procesor P_k trebuie sa transmita vecinilor pe un inel P_{k-1} si P_{k+1}.
  Algoritmul va avea forma:

(1) daca id = k atunci
    (1) ***send*** (M, dreapta);
    (2) ***send*** (M, stanga);
(2) altfel daca id = (k-1) mod p atunci ***recv*** (M, dreapta);
(3) altfel daca id = (k+1) mod p atunci ***recv*** (M, stanga);

Comunicatie prin mesaje

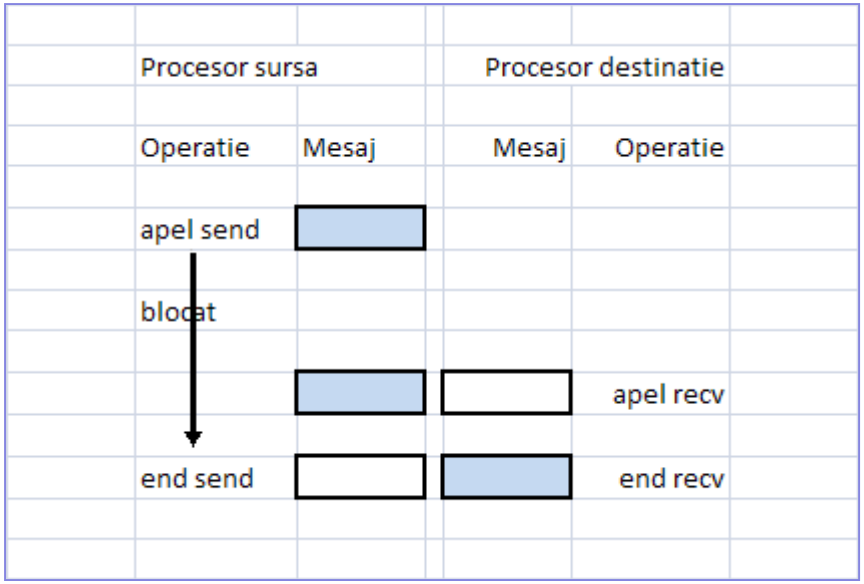
Terminarea locala a comunicatiei

- Presupunem ca procesorul P_s transmite un mesaj procesorului P_d
- Momentele in care P_s apeleaza primitiva ***send***, iar P_a ***recv*** sunt in general diferite
- Transmiterea efectiva a mesajului se face doar dupa ce ambele procesoare au apelat primitivele respective;
- Exista doua posibilitati pentru procesorul care a apelat primul primitiva sa de comunicatie, din momentul apelului si pana cand celalalt procesor apeleaza primitiva pereche:
  - ***Comunicatia blocanta***: asteapta (fara a face nimic);
  - ***Comunicatie non-blocanta***: procesorul poate executa in acest caz si alte operatii;

Comunicatie prin mesaje

Comunicatia blocanta

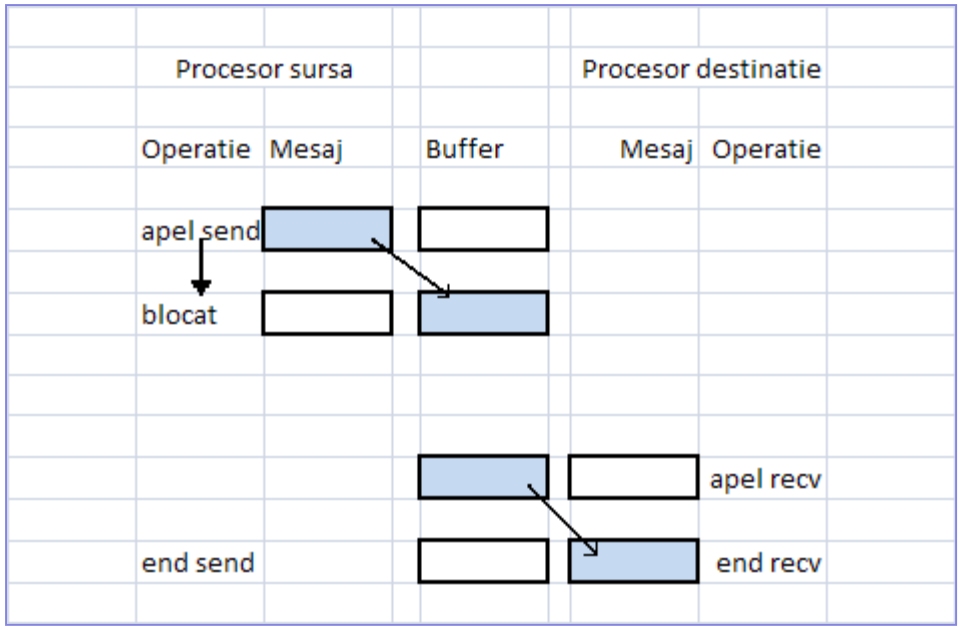
- Al doilea procesor il asteapta pe primul;
- Pe durata asteptarii, acest procesor nu efectueaza alte operatii;
- Comunicatia se numeste blocanta si sincrona (procesoarele vor termina simultan)



Comunicatie prin mesaje

Comunicatia blocanta prin buffere

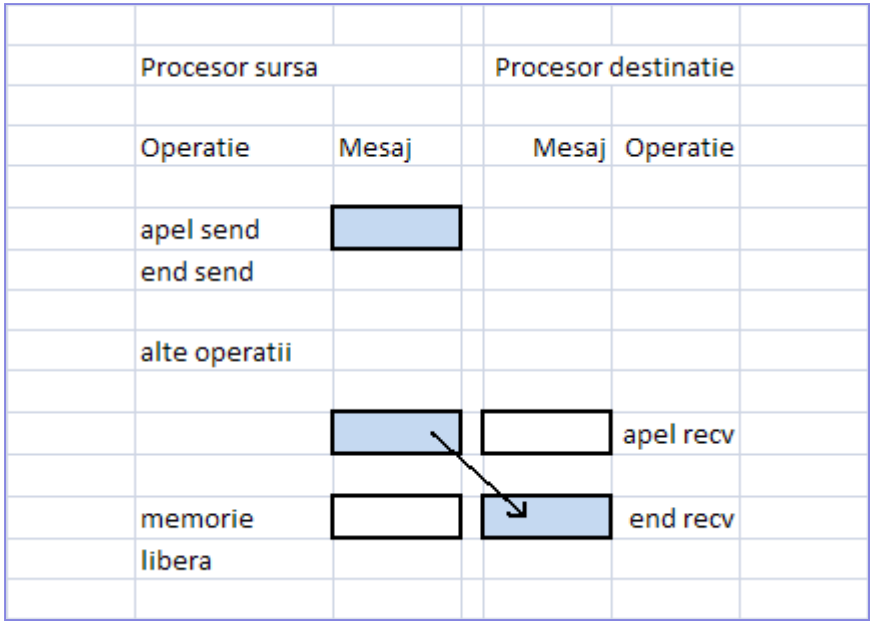
- Se utilizeaza un buffer; primitiva ***send*** muta mesajul din zona sa originala intr-o zona tampon (un buffer); dupa aceasta, executia ***send*** se termina, zona de memorie a mesajului poate fi refolosita;
- Procesorul sursa ramane blocat doar pe durata copierii mesajului in buffer;
- Terminarea rutinei send nu este conditionata de apelul ***recv*** la destinatie;
- La terminarea executiei ***recv***, mesajul este receptionat in zona de memorie alocata acestui scop;



Comunicatie prin mesaje

Comunicatie non-blocanta

- Primitivele send si recv se termina imediat dupa apel, fara ca zona de memorie a mesajului sa fie libera la send sau sa contina mesajul la recv;
- Primitivele au doar rolul de a initia comunicatia, transferul efectiv fiind realizat mai tarziu la un nivel inferior;
- Dupa terminarea executiei send sau recv, procesorul poate executa si alte operatii, in paralel sau concurent cu comunicatia;
- Si in modul non-blocant comunicatia se poate realiza sincron sau prin buffer;



Standardul MPI

- ***Message Passing Interface*** (***MPI***) este un standard descriind primitive de comunicatie – in contextul unui model de programare SPMD
- Este implementat pe toate calculatoarele MIMD si pe multe alte arhitecturi, inclusiv retele de calculatoare;
- Pentru Windows:
  - OpenMPI
  - MPICH2
  - WMPI
- Pentru Linux, Unix:
  - MPI
- Avantaje:
  - Portabilitate;
  - Comunicatia este implementata eficient pe fiecare calculator;
- Se pot testa si depana programe in medii putin costisitoare (din punct de vedere al resurselor de calcul necesare, de ex. desktop computer sau o retea mica) rulandu-le pe supercalculatoare in momentul in care sunt aproape sigur functionale;
- Rutinele MPI se executa in cadrul unui grup de procesoare, numit comunicator;
- Initial, exista doar comunicatorul **MPI_COMM_WORLD** continand toate procesoarele pe care se executa programul;

Standardul MPI

- In cadrul unui comunicator sunt *p* procesoare, numerotate de la 0 la *p-1*.
- Un procesor poate afla numarul de procesoare din comunicator si adresa proprie (numita rank in MPI) prin:

  int MPI_Comm_size(MPI_Comm com, int *p);

  int MPI_Comm_rank(MPI_Comm com, int *my_id);

MPI_Comm este tipul predefinit pentru comunicatoare;

- **com** este comunicatorul in care se afla procesorul;
- In variabilele p si my_id rutinele returneaza valorile numarului de procesoare, respectiv a adresei procesorului apelant;
- Toate rutinele MPI intorc un intreg care caracterizeaza succesul executiei;

Structura unui program MPI

- Un program MPI are forma obisnuita a unui program secvential;
- Rutinele **MPI_Init** si **MPI_Finalize** se apeleaza inaintea si respectiv dupa orice alte rutine MPI;
- **MPI_Init** primeste argumentele argc si argv, acestea avand sau nu o semnificatie in functie de implementarea MPI
- Programatorul este liber sa foloseasca rutinele MPI potrivite cu algoritmul pe care il implementeaza.

Standardul MPI

Rutine de comunicatie MPI

Oricare doua procesoare isi pot transmite mesaje (topologia virtuala este graful complet conectat);

Principalele rutine de comunicatiei sunt:

MPI_Send (void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)

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

Prin ***MPI_Send*** un procesor trimite procesorului dest mesajul aflat in memoria locala la adresa ***buf***, de lungime ***count*** si avand tipul ***datatype***.

Mesajului ii este asociata o eticheta (tag) care il personalizeaza;

Transmisia mesajului se face in cadrul comunicatorului com, care trebuie sa contina si procesorul sursa, si cel destinatie;

MPI_Recv receptioneaza un mesaj de lungime cel mult ***count***, trunchiind eventual mesajul transmis; tipul de date poate diferi de la sursa la destinatie; variabila ***status*** da informatii suplimentare dupa receptie, de ex. lungimea efectiva a mesajului.

Standardul MPI

Moduri de comunicatie

- MPI prevede rutine de comunicatie pentru toate tipurile de comunicatie descrise anterior: blocanta sau non-blocanta, prin buffer sau sincron;
- Modul de implementare a rutinelor de baza (standard) **MPI_Send**, **MPI_Recv** nu este precizat;
- Pentru comunicatia non-blocanta:

  MPI_Wait asteapta terminarea transmisiei sau receptiei;

  MPI_Test verifica daca transmisia sau receptia s-au terminat sau nu;

Comunicatie	blocanta	non-blocanta
standard	MPI_Send MPI Recv	MPI_Isend MPI Irecv
prin buffer	MPI_Bsend	MPI_Ibsend
sincrona	MPI_Ssend	MPIIssend

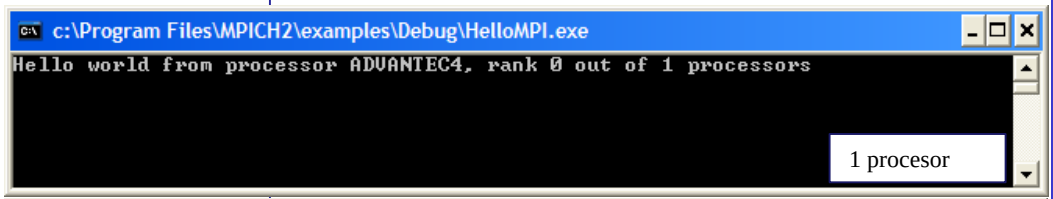
Standardul MPI

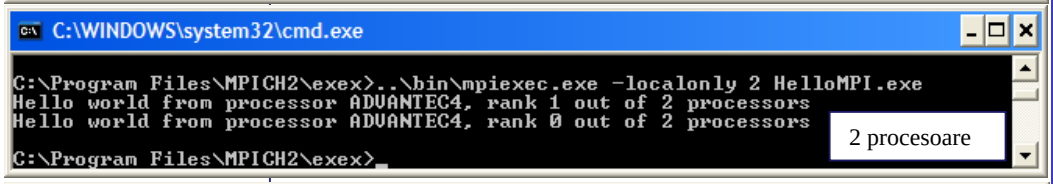
MPI Hello World!

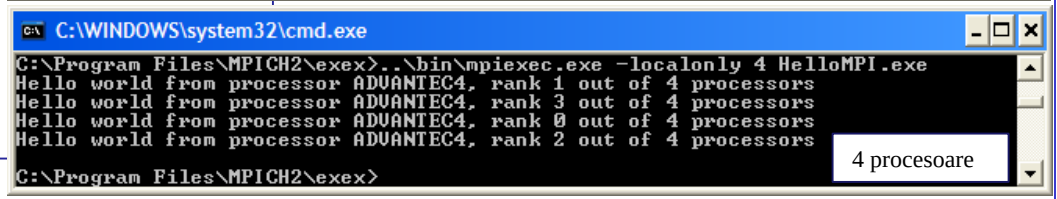
HelloMPI.c

(Global Scope)

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

int main(int argc, char** argv) {
  int world_size;
  int world_rank;
  int name_len;
  // Get the name of the processor
  char processor_name[MPI_MAX_PROCESSOR_NAME];

  // Initialize the MPI environment
  MPI_Init(&argc,&argv);

  // Get the number of processes
  MPI_Comm_size(MPI_COMM_WORLD, &world_size);

  // Get the rank of the process
  MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);


  MPI_Get_processor_name(processor_name, &name_len);

  // Print off a hello world message
  printf("Hello world from processor %s, rank %d"
         " out of %d processors\n",
         processor_name, world_rank, world_size);

  // Finalize the MPI environment.
  MPI_Finalize();

  getch();
}
```







Standardul MPI

Alte functii MPI

- Comunicatie globala: operatii de comunicatie implicand toate procesoarele dintr-un grup (comunicator): sincronizare, difuzare, distributie;
- Alte operatii globale: calculul maximului unor valori distribuite tuturor procesoarelor unui grup, calculul sumei;
- Topologii virtuale. Pentru cresterea performantelor unui program MPI pe un anumit calculator, programatorul poate descrie o topologie virtuala care sa corespunda topologiei reale a calculatorului.
- Procese: se pot creea dinamic, deci se pot imbina paralelismul si concurenta