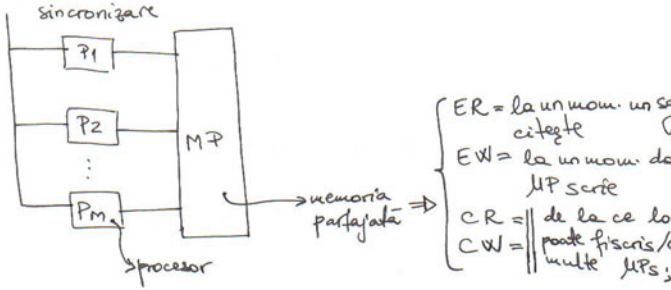
SPM. 17

Modele teoretice de algoritmi paraleli

Un model general permite aflarea unei părți din parametrii.

• Modelul PRAM (Parallel Random Access Memory):

→ ignoră total timpul de acces la memoria comună;
→ model teoretic, bun la analiza situațiilor deadlock, la analiza timpilor de procesare și comunicare, timpul de acces la memorie nefiind inclus;



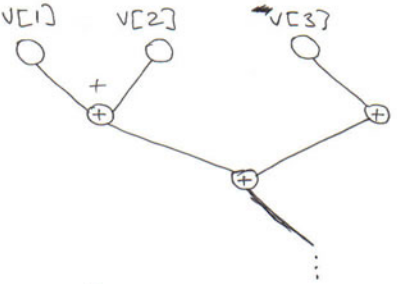
EREW PRAM ⇒ cea mai exclusivă variantă
CREW PRAM ⇒ implementează excluderea mutuală
ERCW PRAM ⇒ nu ar folosi la nimic (model teoretic)
CRCW PRAM ⇒ o variantă interesantă (la un mom. dat m. multe μPs. pot citi sau scrie o locație de mem. în același timp)

CRCW PRAM:
- common PRAM → toate μPs. au aceeași val. de scris;
- arbitrary PRAM → dc. m. multe μPs. vor să scrie aceeași locație de mem., se alege una arbitrară;
- minimum PRAM → μP care are prioritate mai mică ajunge să pună val.;
- combinational PRAM → într-o cerere de scriere simultană combină valorile

• Modelul bazat pe arbori binari (work-depth):

→ nu ține seama de arhitectura mașinii, ci dă importanță algoritmului;
→ model util pt. evaluarea nr. de operații și lungimea (drumul) cel mai lung în algoritm;

Ex.: avem de calculat suma unui vector:



O astfel de modelare permite calculul timpului de execuție și nr. de operații.

Exemple de alg. de calcul ||

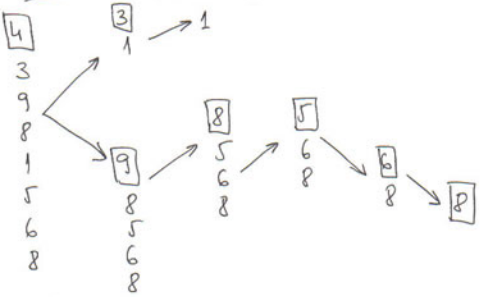
Alg. sincroni se fol. cu precădere în multicalculator.
În cazul multi)ITS se fol. alg. asincroni ⇒ fiecare μP își face bucățica lui, independent de celelalte.

(1) Alg. asincroni ptr. multiprocesoare:

a) înmulțirea matr.: → alg. f. bun ptr. implem. multiprocesor

```
struct global_memory
    { shared float A[M,M], B[M,M], C[M,M]; }
task ()
{ int k;
  GET_NEXT_INDEX(k); // procedură care citește și increm. k, iar când
                     // k=n devine (-1);
  while (k>0)
    { for (i=1; i<=M; i++)
        { c(i,k)=0;
          for (j=1; j<=M; j++)
              c(j,k) = c(i,k) + A(i,j)*B(j,k);
        }
      GET_NEXT_INDEX(k);
    }
}
```

b) sortare Quicksort:




```
task ( )
{ P = A(1);
  for (i=2; i<=N; i++)
      if (A(i)<=P)
        { B(j) = A(i);
          j++;
        }
      else { C(k) = A(i);
             k++;
           }
  task(B);
  task(C);
  BARRIER  //aștept să se termine
  for (i=1; i<j; i++)
      D(i) = B(i);
  D(j+1) = P;
  for(i=1; i<k; i++)
      D(j+1+i) = C(i);
  return D;
}
```

c) Eliminare Gaussiană:

$$\begin{cases} x_1 + 2x_2 - x_3 = -7 \\ 2x_1 - x_2 + x_3 = 7 \\ -x_1 + 2x_2 + 3x_3 = -1 \end{cases} \Rightarrow \begin{bmatrix} \textcircled{1} & 2 & -1 & -7 \\ 2 & -1 & 1 & 7 \\ -1 & 2 & 3 & -1 \end{bmatrix}$$

$-\sqrt{\sqrt{-}}$

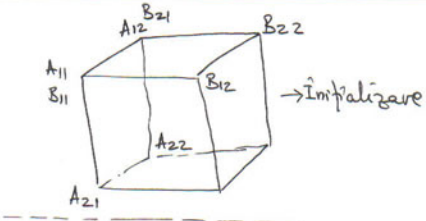
$$\begin{bmatrix} \textcircled{1} & 2 & -1 & -7 \\ 2 & -1 & 1 & 7 \\ -1 & 2 & 3 & -1 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 2 & -1 & -7 \\ 0 & \textcircled{-5} & 3 & 21 \\ 0 & 4 & 2 & -8 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 2 & -1 & -7 \\ 0 & -5 & 3 & 21 \\ 0 & 0 & 22/5 & \dots \end{bmatrix}$$

```
Struct global_memory
    { shared float a(n,m);
      shared int p=0, next_row;
    }
task()
{ int k;
  while (p<m-1)
    { if (proces_id ==1)
        { p = p+1;
          next_row = p+1;
        }
      BARRIER;
      k = Fetch&Add (next_row, 1);
      while (k<=m)
        { mult = -a(k,p)/a(p,p);
          for (i=p; i<=m; i++)
              a(k,i) = a(k,i) + mult * a(p,i);
          k = Fetch&Add (next_row, 1);
        }
    }
}
```

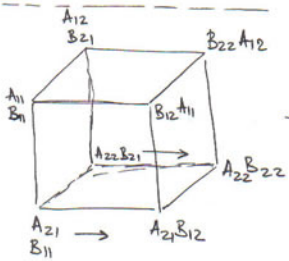
(2) Alg. sincroni pt. multicalculatoare:
(granularitate grosieră)

a) înmulțirea de matrice:

$$C = A * B = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} * \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$ → nu transmitem tot ⇒ timpul de calcul nu mai e așa mare



→ Inițializare



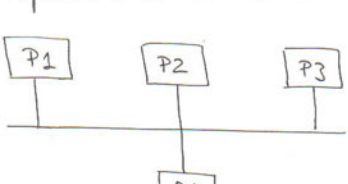
→ A în dreapta, B în jos; fiecare înmulțește ce a primit

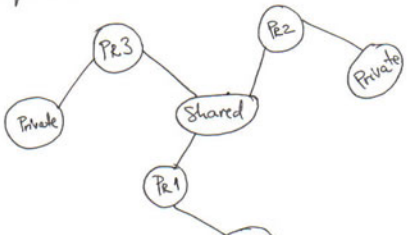
inițializare minimală

SPM.18

Standardul MPI

Open MPI se adresează sist. multiprocesor.





Fiecare fir de execuție beneficiază de un set de variabile proprii. Ptr. comunicație ∃ memoria partajată.

Oricum am face implementarea, MPI vor avea o zonă privată de memorie și vor partaja o zonă comună, atât la progr. multitasking cât și la progr. pe proces.

Stdrdul. MPI e un stdrd. deschis → www.openmp.org

Standardul își propune:

→ standardizarea directivelor de calcul || (C/C++ & Fortran) ⇒ nu se lucrează cu reg. critice, bariere, etc → toate instrucțiunile sunt aduse în aceste directive;