IV. Organizarea fizica si modele

Content

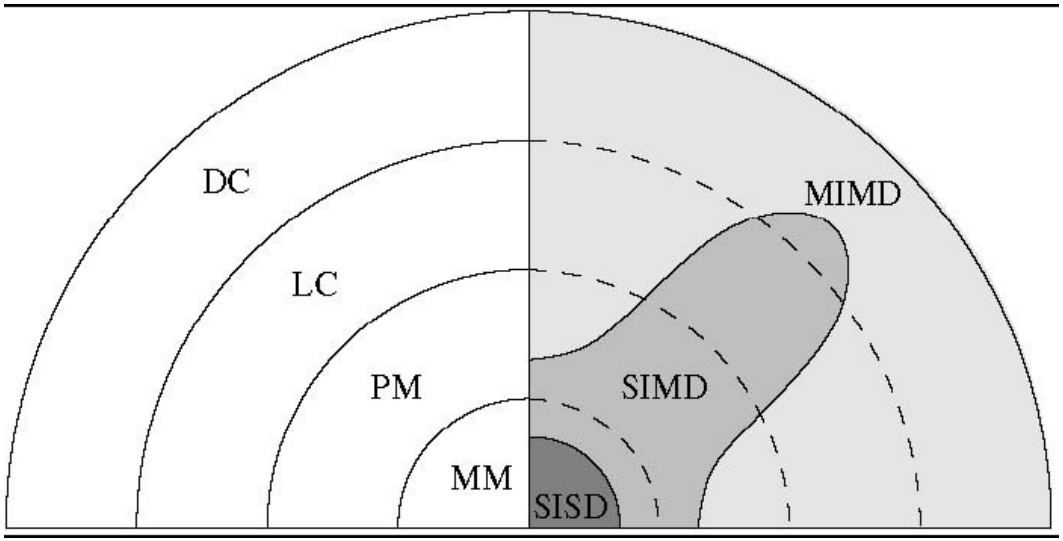
- Organizare fizica:
  - Clasificarea bazata pe raza,
  - Multicore,
  - Cluster,
  - Grid,
  - Trenduri;
- Modele:
  - Modele timpurii,
  - PRAM

Organizare fizica

Clasificare bazată pe rază

- **Masini monoprocesor(MM):** reprezentând în principal computerele SISD (Pcuri vechi, statii de lucru).
  - Includ si masinile SIMD ce contin un processor vectorial.
- **Masini paralele (PM):** construită ca o singură mașină conținând mai multe unități de procesare.
  - Acestea includ arhitecturi SIMD și MIMD și combinații potențiale ale acestora.
- **Clustere locale (LC):** colecții de calculatoare independente adunate în același loc și conectate prin intermediul unei rețele locale
  - Deși sunt intrinsec orientate către MIMD, mașinile SIMD pot fi utilizate la nivelul nodului.
- **Clustere distribuite (DC):** colecții de clustere locale răspândite în întreaga lume și legate între ele prin internet.
  - Aceste sisteme urmează în principal modelul MIMD, dar pot include SIMD.

Legături între clasificarea Flynn și cea bazată pe rază



Exemple de arhitecturi (1/2)

- **Intel Pentium D** –
  - Introdus in 2005,
  - Primul processor dual de la Intel;
  - nucleele au propriile memorii de memorie și accesează memoria comună prin magistrala frontală;
  - lățime de bandă limitată a memoriei în cazul calculelor cu consum intensiv de memorie.
  - conductele lungi permit frecvențe mari de ceas (la momentul scrierii până la 3,73 GHz cu Pentium D 965), dar pot provoca performanțe slabe în cazul if-urilor.
- **Intel Core 2 Duo**
  - succesor a Pentium-ului D
  - Abandonează frecvențele mari de ceas în favoarea unei calculări mai eficiente.
  - La fel ca Pentium D, folosește magistrala frontală pentru acces la memorie de ambele procesoare.
- **AMD Athlon 64 X2 & Opteron** A
  - MD's dual-core CPUs Athlon 64 X2 (single-CPU only) si
  - Opteron (în funcție de model posibil până la 8 CPU într-un singur sistem)
  - procesoare foarte populare pentru clustere Linux.
  - Fiecare nucleu are propriul său canal HyperTransport pentru acces la memorie, ceea ce face ca aceste procesoare să fie potrivite pentru aplicații cu memorie intensă.
- **IBM pSeries**
  - Server și stație de lucru IBM bazată pe procesorul POWER.
  - Procesoare POWER sunt prin design multi-core și dispun de cache-uri mari. IBM construiește sisteme de memorie partajate cu până la 32 de procesoare.

Exemple de arhitecturi (2/2)

- **IBM BlueGene**
  - Arhitectura MPP (massively parallel processing) de la IBM.
  - Utilizează procesoare PowerPC destul de lente de 700 MHz.
  - Aceste procesoare formează sisteme de memorie distribuite foarte mari, foarte integrate, cu rețele de comunicații rapide (un 3D-Torus, precum Cray T3E).
  - Un BlueGene / L cu 131.072 procesoare care oferă o performanță de până la 360 de TeraFLOPS.
- **NEC SX-8**
  - Unul dintre puținele supercomputere vectoriale.
  - Realizează operații vectoriale cu o viteză de 2 GHz, cu opt operații pe ciclu de ceas.
  - Un nod SX-8 este format din opt procesoare, putând fi conectate până la 512 noduri.
- **Cray XT3**
  - un sistem masiv paralel care utilizează CPU Opteron AMD.
- **SGI Altix 3700**
  - ccNUMA sistem care utilizează procesorul Intel Itanium 2.
  - Itanium 2 are cache-uri mari și performanțe bune în punct de plutire.
  - fiind ccNUMA, Altix 3700 este ușor de programat.

Clustere locale

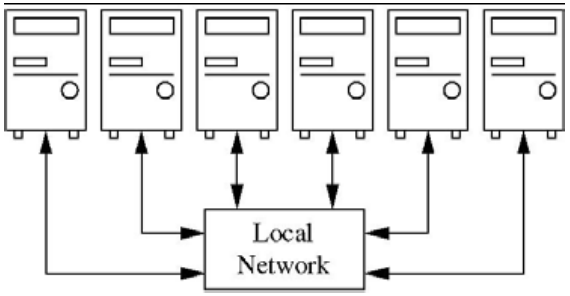
- Un mod practic de a construi MIMD-uri eficiente cu memorie distribuită cu costuri reduse.
- Clusterele au pus la dispoziția celor cu bugete mult mai mici calculele paralele de înaltă performanță.
- Ideea este de a combina componente-off-the-raft (COTS) componente pentru a crea un computer paralel.
  - De exemplu, pe PC-urile care rulează sistemul de operare Linux.
- Rețeaua de comunicații care conectează computerele împreună poate varia de la Gigabit Ethernet la Myrinet și Infiniband, care poate transmite mesaje la un ritm de câteva Ggabits pe secundă (Gbs)
  - Gigabit Ethernet are ~ 100 MB / s și mai ieftin decât Myrinet, dar latența (timpul de călătorie al unui pachet de date) este de 100 mus forGigabit Ethernet > 10 - 20 mus forMyrinet.
    - La o viteză de ceas de 2 GHz, un ciclu durează 0,5 ns. O latență de 10 mus constituie 20.000 de cicluri de timp de călătorie înainte ca pachetul de date să își atingă ținta.
- Posibilitatea de a combina PC-uri și servere cu memorie partajata, de ultimă generație, în clustere.
- Limitari:
  - Comutatoarele cu debit redus pot avea ca rezultat sisteme dezechilibrate și pot deveni un blocaj de performanță major,
    - mai ales când sunt utilizate noduri mai puternice în cluster.

Proiectul Beowulf: primul cluster pentru PC

- Primul cluster PC a fost proiectat în 1994 la Centrul de zbor spațial Goddard NASA pentru a realiza un Gigaflop.
  - 16 PC-uri au fost conectate împreună folosind o rețea Ethernet standard.
  - Fiecare PC a avut un microproc Intel 486 cu performanțe susținute de~ 70 Mflops.
  - A fost construit pentru doar 40.000 de dolari, comparativ cu 1 milion de dolari, care a fost costul pentru un supercomputer echivalent comercial la acel moment.
  - Numit *Beowulf* după eroul timpurilor medievale care l-a învins pe gigantul Grendel.
- În 1997, cercetătorii laboratorului național Oak Ridge au construit un grup Beowulf din multe PC-uri învechite de diferite tipuri:
  - de exemplu, într-o versiune a inclus 75 de PC-uri cu microprocesoare Intel 486, 53 de computere Intel Pentium și cinci stații de lucru rapide Alpha.
  - simulări care produc hărți naționale detaliate de ecoregiuni bazate pe aproape 100 de milioane de grade de libertate.
- Atunci când calculatoarele clusterelor sunt PC-uri care rulează sistemul de operare Linux, aceste clustere se numesc *clustere Beowulf*.
- combinația mai multor computere desktop - cunoscută sub numele de rețea de stații de lucru (NOW) sau grupuri de stații de lucru (COW)
- *soluție furnizor:* constă în furnizarea de facilități de integrare specifice (rafturi și dulapuri) cu un mediu optimizat de rețea și software.

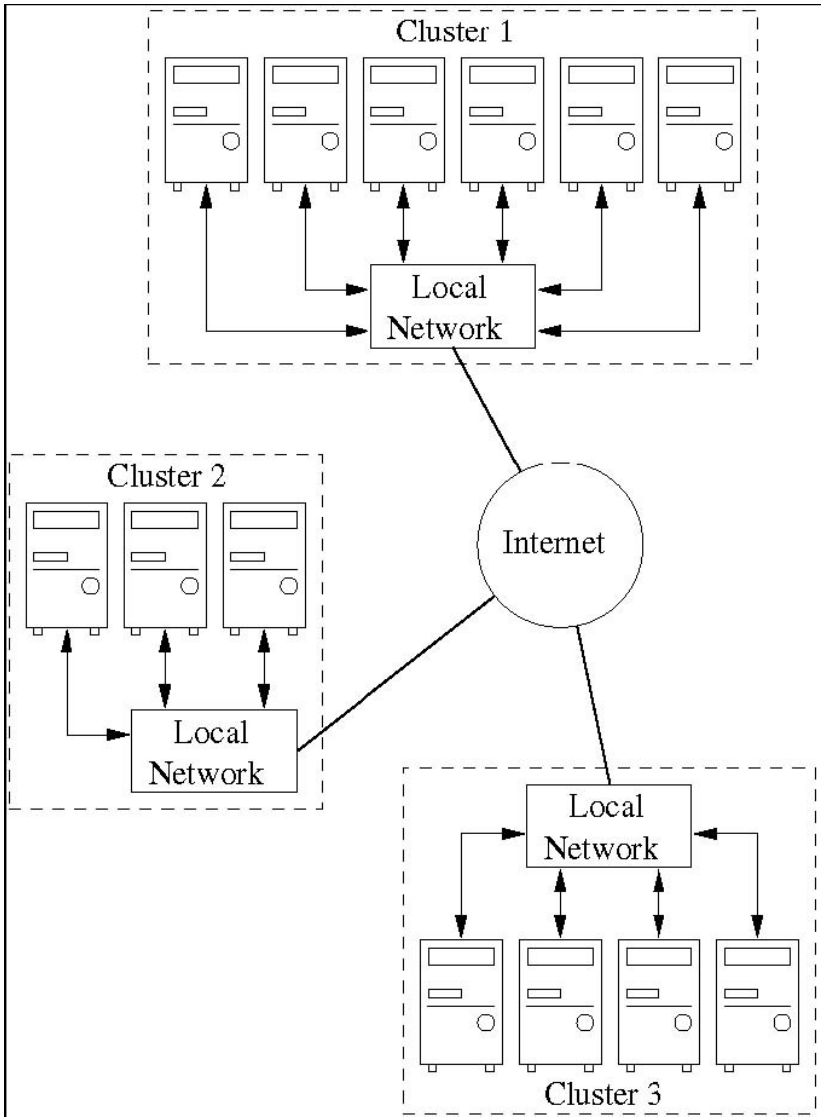
Avantaje & dezavantaje

- Adv:
  - Flexibilitate: magistrala, rețea, RAM pot fi ușor adăugate sau suprimate din system
  - Mai ieftin decât mașinile paralele
- Dez:
  - rețea de interconectare, care este adesea mult mai lentă decât cele complet integrate în mașini paralele.
  - conexiunea fiecărui nod la rețea se face prin magistrala de conexiune a acelui nod, care are adesea lățimi de bandă și / sau latențe restrânse



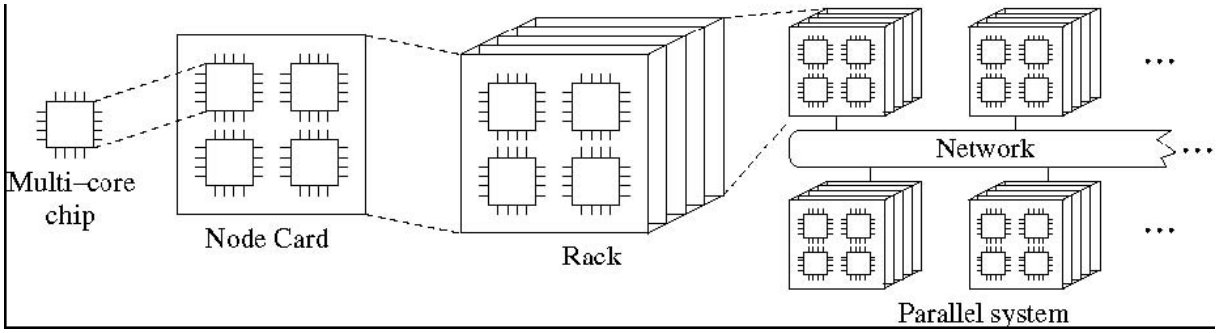
Clustere distribuite - Grid

- Ideea clusterului a fost extinsă la conectarea computerelor din întreaga lume prin Internet
- Un rezultat logical pentru
  - marile îmbunătățiri ale rețelelor vaste din ultimii ani
  - cererea din ce în ce mai puternică pentru sisteme de calcul din ce în ce mai puternice.
- Adv:
  - adunarea foarte mare nr. de masini => o putere de calcul mai mare și o capacitate de memorie mai mare.
- Inconvenientele:
  - comunicațiile între clustere sunt în general mult mai lente decât cele din clusterele locale
  - probleme de securitate



Tendințele configurațiilor utilizate

- Sisteme paralele ierarhice, amestecând memorie distribuită și distribuită
  - Exemplu: IBM BlueGene



- Unii furnizori continuă să dezvolte procesoare specifice pentru a pune în supercomputere cum ar fi procesoarele vectoriale, masive multi-fire sau VLIW (Very Large Instruction Word).
- Reutilizați conceptele paralele care au loc de obicei la nivel de procesor la scara grupurilor mici de procesoare pentru a proiecta procesoare virtuale și mai puternice.
  - Proiecte IBM de Virtual Vector Architecture și procesor de celule.

Tendințele configurațiilor utilizate

- Retele:
  - Gigabit Ethernet a fost utilizat intens în grupuri, dar cu latențe ridicate => altele networks:SCI, Infiniband sau Myrinet.
  - Lățimi de bandă de ordinul Gb/s
  - Latențe de ordinul microsecundei.
  - Flexibilitate mare în topologiile posibile.
- Includerea eterogenității la diferite niveluri ale arhitecturilor paralele.
  - Furnizorii, precum Cray sau SGI, lucrează la sisteme care combină mai multe tipuri de procesoare (vector, scalar)
  - rețelele sunt deja eterogene în toate arhitecturile ierarhice.
- Frontierele dintre diferitele părți ale unui sistem paralel devin din ce în ce mai puțin evidente!
  - Procesoarele tind să devină mini-sisteme multi-procesoare
  - Clustering-ul tinde să fie utilizat la toate nivelurile sistemelor multistrat
    - termenul cluster devine din ce în ce mai inexact în afara oricărui context specific.

Modele de calculatoare paralele

De ce?

- Procesoarele paralele vin în multe variante.
  - Nu este posibil să discutăm toate variantele, inclusiv caracteristicile lor distinctive, punctele forte din contextele aplicației și dezavantajele.
  - ⇒ ne ocupăm adesea cu ***modele abstracte de mașini reale.***
- Beneficiile utilizării modelelor abstracte:
  - teorii independente de tehnologie și tehnici algoritmice care sunt aplicabile unui număr mare de mașini existente și viitoare.
  - Simplitatea conceptuală a unor astfel de modele face ca dezvoltarea algoritmilor și analiza diferitelor compromisuri să fie mai ușor de gestionat.
  - Dacă traducerea automată a acestor algoritmi abstracte în programe eficiente pentru mașini reale este posibilă prin utilizarea unor compilatoare inteligente sau optimizate, atunci aceste modele pot fi într-adevăr utile.
- Dezavantajele includ
  - incapacitatea de a prezice cu exactitate performanța reală
  - tendința de a simplifica prea mult modelele, astfel încât acestea să nu mai reprezinte nicio mașină reală.

Modele timpurii - Prelucrare asociativăAP

- Memorii asociative sau adresabile conținutului (AMs, CAMs),
  - permite accesul celulelor de memorie pe baza conținutului, mai degrabă decât pe locațiile lor fizice din memorie
  - a venit în anii 1950 când progresele în tehnologiile de memorie magnetică și criogenă au permis construirea unor prototipuri de dimensiuni rezonabile.
- Bazat pe încorporarea logicii de procesare simplă în tabloul de memorie
  - ⇒ elimina nevoia de a transfera volume mari de date prin interfața cu lățime de bandă limitată între memorie și procesor (gâtuirea von Neumann).
- Memoriile asociative timpurii au oferit două capacități de bază:
  1. căutare mascată sau căutarea unui anumit model de biți în câmpurile selectate ale tuturor cuvintelor de memorie și marcarea celor pentru care este indicată o potrivire,
  2. scriere paralelă sau stocarea unui model de biți dat în câmpurile selectate ale tuturor cuvintelor de memorie marcate anterior.
  - Aceste funcții + operații logice pe vectori de marcaj (de exemplu, ORing-le) sunt suficiente pentru programarea căutărilor sau chiar operațiuni aritmetice paralele.

Modelul AM/AP

- În ultima jumătate de secol, modelul AM / AP a evoluat prin încorporarea unor capacități suplimentare, astfel încât, în esență, converg cu procesoarele de tip SIMD.
- Exemple timpurii: procesor STARAN Goodyear, produs comercial din anii '70, al cărui design a fost motivat de problema de detectare a conflictelor aeronavei, intensiv în calcul;
  - $O(n^2)$ verificări în perechi necesare pentru evitarea coliziunilor și a ratărilor apropiate pentru n avioane în vecinătatea unui aeroport ocupat.
- Încarnările moderne ale acestui model sunt văzute în cipurile PIM (procesor în memorie),
  - practic cipuri DRAM standard cu un nr mare. procesoare foarte simple adăugate pe căile lor de acces la date,
  - și arhitecturi inteligente RAM (IRAM),
  - avantaje atât în ceea ce privește performanța, cât și consumul de energie.

Modele timpurii - rețele neuronale și automate celulare

- rețele neuronale
  - introduse in anii ‘50,
  - s-a ocupat de procesarea paralelă pentru aplicații de înțelegere a imaginilor
  - dezvoltarea perceptronilor (un dispozitiv asemănător neuronului care se ocupă de procesarea unui singur pixel într-o imagine digitală) în anii ‘40.
  - în anii 1960, o serie de activități de cercetare au pus bazele domeniului modern al *rețelelor neuronale.*
  - introducerea algoritmului de învățare a propagării înapoi a pus rețele neurale pe pista rapidă, astfel încât astăzi sunt utilizate pentru soluționarea problemelor de decizie complexe într-o clasă largă de aplicații.
- Automate celulare
  - o colecție de automate cu state finite identice, care sunt interconectate, prin legăturile lor de intrare-ieșire, în mod regulat, cu tranzițiile de stare ale fiecărui automat controlate de propriul său stat, stările vecinilor la care este conectat și intrări primare, dacă există.
  - *Matricele sistolice,* care stau la baza proiectelor bazate pe VLSI de înaltă performanță în unele zone de aplicație, pot fi privite ca automate celulare.
  - Recent: o reînviere a interesului pentru CA ca modele teoretice ale sistemelor masive paralele și instrumentelor pentru modelarea fenomenelor fizice.

Modelul PRAM model - abstractizare

- Modelul teoretic utilizat pentru calculatoarele convenționale sau secvențiale (clasa SISD) este cunoscut sub numele de *mașina cu acces aleatoriu* (RAM)
  - nu trebuie confundat cu mem-ul cu acces aleatoriu, care are același acronim
- Versiunea paralelă a RAM, PRAM (bob de mazăre), constituie un model abstract al clasei procesoarelor paralele cu memorie globală.
- Abstractizarea constă din
  - ignorând detaliile rețelei de interconectare procesor-la-memorie și
  - considerând că fiecare procesor poate accesa orice locație de memorie din fiecare ciclu de mașină, independent de ce fac alte procesoare.
- Exemplu:
  - Algoritmii PRAM ar putea implica afirmații de genul

"for 0 <= *i* < *p,* Processor *i* adds the contents of memory location 2*i* + 1 to the memory location 2*i*" (diferite locații accesate de diferitele procesoare)

"each processor loads the contents of memory location *x* into its Register 2"

(aceeași locație accesată de toate procesoarele).

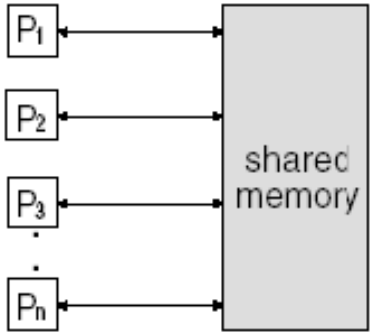
- Problema mai multor procesoare care încearcă să scrie într-o locație de memorie comună trebuie rezolvată într-un fel.
  - Se pot folosi diverse scheme de inhibare, prioritate sau combinare atunci când se încearcă operațiuni de scriere simultană într-o locație comună.

PRAM – funcționalitate?

- se presupune că un singur procesor este activ inițial.
  - În fiecare etapă de calcul, fiecare proces activ poate citi și scrie în memoria comună și poat activa, de asemenea, un alt procesor.
- modelul abstract PRAM poate fi SIMD sau MIMD.
  - Varianta SIMD: toate echipamentele obțin aceleași instrucțiuni în fiecare ciclu de mașină;
  - din cauza adresării indexate și indirecte (bazate pe registre), acestea execută adesea operațiunea care le este transmisă pe date diferite.
- Procesoarele partajează un ceas comun, dar pot executa instrucțiuni diferite în fiecare ciclu.
- modelul PRAM este foarte teoretic.
  - Dacă s-ar construi o PRAM fizică, conectivitatea procesor-memorie ar trebui realizată de o rețea de interconectare.
  - Deoarece locațiile de memorie sunt prea numeroase pentru a fi atribuite porturi individuale într-o rețea de interconectare, blocuri de locații de memorie (sau module) ar trebui să partajeze un singur port de rețea.
    - Fiecare ciclu de instrucțiuni ar trebui să consume O (log p) în timp real.

PRAM ca model ideal

- De obicei, se presupune că PRAM este văzută ca o memorie globală de *dimensiuni nelimitate*, care este uniform accesibilă pentru toate procesoarele.
- Să presupunem un PRAM ca un computer cu memorie partajată cu procesoare *p* și o memorie globală de *m* cuvinte.
  - Procesoarele sunt conectate la memorie printr-un set de comutatoare.
  - Aceste comutatoare determină cuvântul de memorie accesat de fiecare procesor.
  - Fiecare procesor *p* din ansamblu poate accesa oricare dintre cuvintele de memorie, cu condiția ca un cuvânt să nu fie accesat de mai mult de un procesor simultan.
  - Pentru a asigura o astfel de conectivitate, numărul total de comutatoare trebuie să fie O(*mp*).
    - Pentru o dimensiune rezonabilă a memoriei, construirea unei rețele de comutare cu această complexitate este foarte scumpă.
    - ⇒ Modelele PRAM de calcul sunt imposibil de realizat în practică.



Acces la memorie în modelul PRAM

- Toate procesoarele au acces de citire și scriere la o memorie globală partajată
  - În PRAM accesul poate fi simultan.
  - Fiecare procesor teoretic poate accesa memoria globală partajată într-o unitate de timp neîntreruptă.
  - Fiecare procesor poate efectua în paralel diverse operații aritmetice și logice.
- Modelul PRAM are algoritmi de citire simultan și exclusiv.
  - Algoritmii de citire simultană au voie să citească aceeași bucată de memorie simultan, fără corupția datelor.
  - Algoritmii de citire exclusivă sunt folosiți pentru a se asigura că niciun procesor nu a citit vreodată aceeași locație de memorie în același timp.
- Modelul PRAM are, de asemenea, alg. de scriere concomitentă și exclusivă.
  - Algoritmii de scriere simultană permit procesoarelor multiple să scrie în memorie
  - Algoritmii de scriere exclusivă asigură că niciun procesator nu scrie în aceeași memorie în același timp.

Submodele ale PRAM

1. **Exclusive-read, exclusive-write (EREW)** PRAM.
   - Accesul la o locație de memorie este exclusiv.
   - Nu sunt permise operațiuni simultane de citire sau scriere.
   - Cel mai slab model PRAM: concurență minimă în accesul la memorie.
   - Cea mai realistă dintre cele patru submodele.
2. **Concurrent-read, exclusive-write (CREW)** PRAM.
   - Accesul cu citire multipla la o locație de memorie este permis.
   - Mai multe accesuri de scriere la o locație de memorie sunt serializate.
   - Submodel implicit: asumat când nu se spune nimic despre submodel,
3. **Exclusive-read, concurrent-write (ERCW)** PRAM.
   - Accesul multiplu la scriere este permis la o locație de memorie,
   - Accesul multiplu la citire este serializat.
4. **Concurrent-read, concurrent-write (CRCW)** PRAM.
   - Acces multiplu la citire și scriere la o locație de memorie comună.
   - Acesta este cel mai puternic model PRAM.
   - Cel mai puțin restrictiv submodel,
   - Solicita un mecanism de rezolvare a conflictelor pentru a defini scrieri concomitente.

Protocoale pentru scrierea concomitentă

- Comun:
  - scrierea simultană este permisă dacă toate valorile pe care procesatorii încearcă să le scrie sunt identice.
- Arbitrar:
  - un procesor arbitrar are voie să procedeze cu operația de scriere, iar restul eșuează.
- Prioritate:
  - toate procesoarele sunt organizate într-o listă de prioritate predefinită, iar procesorul cu cea mai mare prioritate reușește, iar restul eșuează.
- Suma:
  - suma tuturor cantităților este scrisă (modelul de soluționare a conflictelor bazate pe sumă poate fi extins la orice operator asociat, definit pe cantitățile scrise).

CRCW PRAM - clasificari

- Nedefinit (CRCW-U) :
  - În cazul scrierii multiple, valoarea scrisă este nedefinită.
- Detectare (CRCW-D):
  - Se scrie un cod special care reprezintă „coliziunea detectată”.
- Comum (CRCW-C):
  - Scrieri multiple sunt permise numai dacă toate stochează aceeași valoare.
  - Numit uneori submodelul *cu scriere consistentă.*
- Aleator (CRCW-R):
  - Valoarea scrisă este aleasă la întâmplare dintre cele oferite.
  - Acesta este uneori numit submodel *scriere-arbitrară.*
- Prioritate (CRCW-P):
  - Procesorul cu cel mai mic indice reușește să-și scrie valoarea.
- Max/Min (CRCW-M):
  - Cea mai mare / cea mai mică dintre valorile multiple este scrisă.
- Reducere:
  - Suma aritmetică (CRCW-S), AND AND (CRCW-A), XOR logic (CRCW-X), o altă combinație a valorilor multiple este scrisă.

Ordonarea submodelelor

- O modalitate de a ordona submodele este prin puterea lor de calcul.
- Două submodele PRAM sunt la fel de puternice dacă fiecare îl poate imula pe celălalt cu o încetinire a factorului constant.
- Un submodel PRAM este (strict) mai puțin puternic decât un alt submodel (notat cu simbolul „<”) dacă există probleme pentru care primul necesită pași de calcul semnificativ mai mulți decât al doilea.
- Exemplu:
  - CRCW-D PRAM < CRCW-M (maxim),
    - CRCW-M poate găsi cel mai mare număr dintr-un vector A de dimensiunea p într-o singură etapă:
      - Procesorul i citește A [i] și îl scrie pe o locație convenită x, care va păstra valoarea maximă pentru toate procesoarele
    - CRCW-D are nevoie de cel puțin O (log n) pași.
- Relația „mai puțin puternică sau egală” dintre submodele poate fi definită în mod similar.

  EREW < CREW < CRCW-D < CRCW-C < CRCW-R < CRCW-P
  - EREW poate simula submodelul CRCW cu cel mult încetinirea logaritmică.
  - Un procesor P CRCW-P (prioritate) poate fi simulat de un procesor *p* EREW PRAM cu un factor de încetinire a O (log *p*).

Alg.1: Difuzarea datelor - cum?

- Un procesor trebuie să trimită o valoare a datelor tuturor celorlalte procesoare.
  - În CREW / CRCW, difuzarea banală: proc trimiterea scrie valoarea datelor într-o locație de memorie, toate procesoarele citind acea valoare de date în ciclul următor.
    => difuzarea simplă se face în O (1) pași.
- Multidifuziunea în cadrul grupurilor este la fel de simplă dacă fiecare procesor își cunoaște apartenența la grupuri și doar membrii fiecărui grup citesc datele:
  - Difuzarea integrală: fiecare dintre procesatoarele p trebuie să trimită o valoare a tuturor celorlalte procesoare - se poate face prin *p* operații de difuzare separate în pași O (*p*).
- Schema de mai sus este clar aplicabilă difuzării în modelul EREW.
- Cea mai simplă schemă pentru difuzarea EREW este:
  - se fac copii ale valorii datelor, să spunem într-un vector de difuzare *B* de lungime *p,*
  - apoi fiecare procesor citeaste propria copie accesând *B*[*j*].
  - Inițial, Processor *i* scrie valoarea datelor în *B*[0].
  - ***Dublarea recursivă*** este utilizată pentru a copia B [0] în toate elementele lui B în O (log 2 p)
  - In final, procesosul *j,* 0 ≤ *j* < *p,* citeste *B* [*j*] pentru a obține valoarea datelor difuzate de Procesor *I*

Making *p* copies of *B*[0] by recursive doubling
for *k* = 0 to log2 *p* – 1 Processor *j,* 0 ≤ *j* < *p,* do
Copy *B*[*j*] into *B*[*j* + 2*k*]
Endfor

Alg.1: Difuzarea datelor in PRAM

- Nota: în Pasul *k* din procesul de dublare recursivă de mai sus, trebuie să fie active doar primele procesoare 2*k*.
- Algoritmul complet de difuzare EREW cu această prevedere

EREW PRAM algorithm for broadcasting by Processor *I*
Processor *i* write the data value into *B*[0]
s := 1
while *s* < *p* Processor *j,* 0 ≤ *j* < min(*s, p* – *s*), do
    Copy *B*[*j*] into *B*[*j* + *s*]
    s := 2*s*
endwhile
Processor *j,* 0 ≤ *j* < *p,* read the data value in *B*[*j*]

- Parametrul s poate fi interpretat ca „span” al elementelor deja modificate sau „pas” pentru operația de copiere.
- O(log *p*)-pasi: este optimal pentru EREW PRAM.

Alg.2: Difuzarea integrală în PRAM

- Pentru a efectua o difuzare toti-la-toti,
  - fiecare procesor transmite o valoare pe care o deține la fiecare dintre celelalte procesoare p-1
  - Procesorul *j* scrie valoarea sa în *B[j],* mai degrabă decât în *B[0].*
  - Într-o etapă de acces la memorie, toate valorile care urmează să fie transmise sunt înscrise în vectorul de difuzare *B.*
  - Fiecare procesor citește apoi celelalte valori p-1 din accesele de memorie *p*-1.
  - Asigurați-vă că toate citirile sunt exclusive: Procs j începe să citească valorile începând cu *B*[*j* + 1], înfășurându-se la *B*[0] după citire*B*[*p* – 1].

EREW PRAM data broadcasting without redundant copying.

EREW PRAM algorithm for all-to-all broadcasting

Processor *j,* 0 ≤*j* < *p,* write own data value into *B*[*j*]

for *k* = 1 to *p* – 1 Processor *j,* 0 ≤*j* < *p,* do

Read the data value in *B*[(*j* + *k*) mod *p*]

endfor

Alg. 3: Algoritmul de sortare naiv

- Dat cu un vector de date *S* de lungime *p,*
- Se calculeaza rangul *R[j]* al elementului de date *S[j]* și apoi se stocheaza *S[j]* în *S*[*R*[*j*]].
- Rangul *R*[*j*] al *S*[*j*]
  - = total nr. elemente de date care sunt mai mici decât *S*[*j*],
  - calculat de fiecare proces care examinează toate celelalte elemente de date și determina numărul element *S*[*l*] < *S*[*j*].
- Fiecare element de date trebuie să primească un rang unic => legături rupte prin utilizarea proc ID.
  - Dacă procesoarele *i* și *j* (*i* <*j*) dețin valori de date egale, valoarea din Procesorul *i* este „mai mică” în scopuri de clasare.

Naive EREW PRAM sorting algorithm using all-to-all broadcasting
Processor *j,* 0 ≤□*j* < *p,* write 0 into *R*[*j*]
for *k* = 1 to *p* – 1 Processor *j,* 0 ≤□*j* < *p,* do
    l := (*j* + *k*) mod *p*
    if *S*[*l*] < *S*[*j*] or *S*[*l*] = *S*[*j*] and *l* < *j*
        then *R*[*j*] := *R*[*j*] + 1
    endif
endfor
Processor *j,* 0 ≤□*j* < *p,* write *S* [*j*] into *S*[*R*[*j*]]

- Nu este optim prin faptul că O (p²) calculi sunt implicate, semnificativ mai mare decât O(*p* log *p*) necesar pentru sortarea elementelor *p* pe un singur procesor.

Semigrupuri si calculul prefixelor

- **Calculul subgrupurilor**:
  - este definit pe baza unui operator binar asociativ o.
  - trivial pentru un CRCW PRAM al soiului „reducere”
  - Exemple:
    - calcularea sumei aritmetice (logice AND, logice XOR) a valorilor *p*, una pe procesor,
      - trivial pentru CRCW-S (CRCW-A, CRCW-X) PRAM;
        - acesta poate fi realizat într-un singur ciclu de fiecare proc, scriind valoarea corespunzătoare într-o locație comună, care va reține apoi suma aritmetică a tuturor valorilor.
  - Schema de dublare recursivă poate fi utilizată pe un EREW PRAM
    - singura diferență care apare în ultima etapă de difuzare.

- **Calculul prefixelor**
  - constă în prima fază a calculului semigrupului.
  - Paradigma divizării și cuceririi:
    - Problemă compusă din două subprobleme:
    1. calcularea rezultatelor imparite *s*1,*s*3,*s*5,...
    2. calcularea egal-index.results *s*0,*s*2,*s*4,...
    - Primul subproblem este rezolvat după cum urmează.
      - Perechile de elemente consecutive din lista de intrare (x0 și x1, x2 și x3, x4 și x5 și așa mai departe) sunt combinate pentru a obține o listă cu jumătate din dimensiune.
    - Efectuarea calculului paralel prefix pe această listă obține valori pentru toate rezultatele imparite.
    - Rezultatele par indexate sunt apoi găsite într-o singură etapă PRAM prin combinarea fiecărei intrări indexate cu rezultatul impare imediat precedent.
    - Timpul total de calcul este dat de recurență $T(p) = T(p/2) + 2$ cu solutia $T(p) = 2 \log 2\ p$.

Înmulțirea matricelor

- Dat fiind matricile *m x m A* și *B*, cu elementele *a[i, j]* și *b[i, j]*, produsul lor *C* poate fi obținut cu un algoritm secvențial O(m^3).
- Dacă PRAM are procesoare $p = m^3$, atunci înmulțirea matricei se poate face în timp O (log *m*)
  - un procesor calculează *a[i,k] x b[k,j]* și apoi permit grupurilor de *m* proc-uri să efectueze m-input + s (semigrup comp) în timp O (log m).
  - Deoarece de obicei nu ne interesează procesarea paralelă pentru înmulțirea matricei, cu excepția cazului în care *m* este destul de mare, aceasta nu este o soluție practică!
- Presupunem că PRAM are $p = m^2$ procesoare.
  - În acest caz, înmulțirea matricei se poate face în timp O(*m*) folosind un procesor pentru a calcula fiecare element *c[i,j]* al matricei *C.*
  - Procesor responsabil de calcul *c[i,j]*
    - citește elementele din linia *i* în *A* și elementele din coloana *j* din *B,*
    - își înmulțește elementele *k*th corespunzătoare,
    - adaugă fiecare dintre produsele astfel obținute la un total final *t.*
  - În paralel, ciclurile *i* și *j* din algoritmul secvențial.
  - Etichetați procesoarele m^2 cu doi indici *(i,j)*, fiecare variind de la 0 la $m - 1$, mai degrabă decât cu un singur indice cuprins între 0 și $m^2 - 1$.

Implementarea CREW a A x B

PRAM matrix multiplication algorithm using m^2 processors
Processor (i, j), $0 \leq i, j < m$, do
begin
    $t := 0$
    for $k = 0$ to $m - 1$ do
    $t := t + a[i,k]b[k,j]$
    endfor
    $cij := t$
end

- Într-o iterație dată a ciclului k,
  - Toate procesoarele (i, y), $0 \leq y < m$, acceseaza același element $a[i,k]$ a lui A
  - Toate procesoarele (x, j) acceseaza același element $b[j,k]$ a lui B.

A x B utilizand *m* procesoare

- Înmulțirea matricei se poate face în O(m^2)
  - Procesorul *i* calculeaza *m* elemente din linia *i* al matricei *C* a produsului.
  - Procesorul *i* va
    - citi elementele din linia *i* din A și elementele tuturor coloanelor din B,
    - înmulți elementele lor *k* corespunzătoare,
    - adăuga fiecare dintre produsele astfel obținute la *t.*

Parallelize the *i* loop in the sequential algorithm.
PRAM matrix multiplication algorithm using *m* processors
for *j* = 0 to *m* – 1 Processor *i,* 0 ≤ *i* < *m,* do
t := 0
for *k* = 0 to *m* – 1 do
t := *t* + *a[b,i] b[k,j]*
endfor
cij := *t*
Endfor

- Fiecare procesor citește un rând diferit de A => nu se încearcă citiri simultane.
- *B:* toate *m* procesoare accesează același element *b[k,j]* în același timp.
- Atât pentru $p = m^2$ cât și pentru $p = m$ algoritmi eficienți cu viteze liniare.

A x B utilizand mai putin de *m* procesare

- Alg. naiv:
  - Putem permite procesorului i să calculeze un set de rânduri m / p în matricea de rezultate C precum *i, i + p, i + 2p, . . . , i + (m/p – 1)p.*
  - Paralelizarea ciclul in *i*, deoarece aceasta este de preferat
    - daca se considera ciclul in *k* -- are dependențe de date
    - daca se considera ciclul in *j* -- implică *m* sincronizări ale procesoarelor, o dată la sfârșitul fiecărei iterații, asumând modelul SPMD
  - Dezavantaj: fiecare element B este obținut *m / p* de ori, cu doar două operații aritmetice (+, x) efectuate pentru fiecare astfel de element.
- Înmulțirea blocului matricial:
  - crește raportul de acces la calcul la memorie;
  - se împart *m x m* matricile *A, B* și *C* în *p* blocuri de mărime *q x q.*
  - se înmulțesc matricile m x m folosind matricea x cu procesoare, unde termenii din instrucțiunea algoritmului *t:=t+a[i,k]*b[k,j]* sunt acum matricile *q x q*
  - Procesorul *(i,j)* calculează blocul *(i,j)* al matricei *C* rezultat.
  - Fiecare calcul multiplu-adăugare pe blocurile *q x q* are nevoie de $2q^2 = 2m^2/p$ de acces de memorie pentru a citi blocurile și operațiile aritmetice $2q^3$.
  - *q* operațiile aritmetice sunt efectuate pentru fiecare acces la memorie și se va obține o performanță mai bună ca urmare a localității îmbunătățite.