Modele de algoritmi paraleli și algoritmi simpli paraleli

Continut

- Modele
  - paralel în date,
  - graful sarcinilor,
  - bazin de lucru,
  - master-sclav,
  - conducta,
  - hibrid;
- Aplicarea modelului paralel în date,
- Calcule în bloc
- Retele de sortare

Modele & Modelul paralel în date

- Un model de algoritm este de obicei o modalitate de structurare a unui alg paralel.
  - prin selectarea unei tehnici de descompunere și cartografiere și
  - prin aplicarea strategiei adecvate de minimizare a interacțiunilor.
- Modelul parallel în date este unul dintre cele mai simple modele de algoritmi.
  - sarcinile sunt mapate static sau semi-static pe procese și fiecare sarcină efectuează operațiuni similare pe date diferite.
  - Acest tip de paralelism care este rezultatul operațiunilor identice aplicate simultan pe diferite elemente de date se numește paralelism de date.
  - Lucrările pot fi realizate în faze, iar datele operate în diferite faze pot fi diferite.
  - În mod obișnuit, fazele de calcul paralele cu date sunt întrerupeți cu interacțiuni pentru a sincroniza sarcinile sau pentru a obține date noi la sarcini.
  - Deoarece toate sarcinile efectuează calcule similare, descompunerea problemei în sarcini se bazează, de regulă, pe partiționarea datelor
    - deoarece o partiționare uniformă a datelor urmată de o mapare statică este suficientă pentru a garanta echilibrul de sarcină

Modelul paralel în date

- Algoritmii paraleli în date pot fi implementate atât în paradigme de spațiu de adresă comună, cât și de transmitere a mesajelor.
  - Spațiul de adrese partiționat într-o paradigmă de transmitere a mesajelor poate permite un control mai bun al plasării și, prin urmare, poate oferi un control mai bun asupra localității.
  - Spațiul adresat în comun poate ușura efortul de programare, mai ales dacă distribuția datelor este diferită în diferite faze ale algoritmului.
- Surplusul interacțiunii din modelul poate fi redus la minimum:
  - prin alegerea unei localități care păstrează descompunerea
  - prin suprapunerea calculului și interacțiunii și
  - prin utilizarea unor rutine de interacțiune colectivă optimizate.
- O caracteristică cheie a modelului este că, pentru majoritatea problemelor, gradul de paralelism al datelor crește odată cu dimensiunea problemei, ceea ce face posibilă utilizarea mai multor procese pentru rezolvarea eficientă a problemelor mai mari..
- Un exemplu de algoritm paralel în date este înmulțirea matricelor dense.

Modelul grafului

- Graful de dependență a sarcinilor poate fi fie banal, ca în cazul înmulțirii matriceale, fie nontrivial.
  - În anumiți algoritmi paraleli, graficul de dependență este utilizat explicit în mapare.
- În modelul graficului de sarcini, interrelațiile dintre sarcini sunt utilizate pentru a promova localitatea sau pentru a reduce costurile de interacțiune.
- Folosit pentru a rezolva probleme în care cantitatea de date asociate sarcinilor este mare în raport cu cantitatea de calcul asociată acestora
  - Lucrul este mai ușor împărtit în paradigme cu spațiu adresabil la nivel global, dar sunt disponibile mecanisme de partajare si în spațiul de adrese disjuncte.
- Printre tehnicile tipice de reducere a interacțiunilor aplicabile modelului:
  - reducerea volumului și frecvenței interacțiunii prin promovarea localității, în timp ce maparea sarcinilor se face pe baza modelului de interacțiune a sarcinilor și
  - folosirea metodelor de interacțiune asincronă pentru a suprapune interacțiunea cu calculul.
- Exemple de algoritmi bazate pe modelul grafului de activități includ:
  - quicksort paralel,
  - factorizare cu matrice rară,
  - algoritmi paraleli derivați prin descompunerea divizăre-și-cucerire.
- Acest tip de paralelism care este exprimat în mod natural prin sarcini independente într-un graf de dependență a sarcinilor se numește paralelism de sarcină.

Modelul bazinului de lucru (modelul grupului de sarcini)

- Caracterizat printr-o mapare dinamică a sarcinilor pe procese pentru balansarea incarcarii în care orice sarcină poate fi realizată de orice proces.
- Nu este dorită o premapare a sarcinilor pe procese.
- Maparea poate fi centralizată sau descentralizată.
- Pointerii pentru sarcini pot fi stocati într-o listă partajată fizic, coadă de prioritate, tabel hash sau arbore sau pot fi stocati într-o structură de date distribuită fizic.
- Lucrul poate fi disponibilă static la început sau ar putea fi generată dinamic;
  - adică, procesele pot genera lucru și o pot adăuga la grupul de lucru global (eventual distribuit).
  - Dacă lucrul este generat dinamic și se utilizează o mapare descentralizată, atunci un alg. de detectare a terminarii este necesar pentru ca toate procesele să poată detecta efectiv finalizarea întregului program.
- În paradigma de transmitere a mesajelor, modelul bazinului de lucru este utilizat de obicei atunci când cantitatea de date asociate sarcinilor este relativ mică în comparație cu calculul asociat sarcinilor.
  - sarcinile pot fi ușor mutate fără a provoca un surplus al interacțiunii prea mare.
- Granularitatea sarcinilor poate fi ajustată pentru a atinge nivelul dorit de schimb între dezechilibrul de încărcare și surplusul genral de accesul la bazinul de lucru pentru adăugarea și extragerea sarcinilor.
- Paralelizarea ciclurilor prin planificare sau metode conexe este un exemplu de utilizare a modelului bazinului de lucru cu mapare centralizată atunci când sarcinile sunt disponibile static.
- Ex. Model al bazinului de lucru in care sarcinile sunt generate dinamic:
  - Căutare în arbore paralelă, cu o structură de date centralizată sau distribuită

Modelul master-sclav (manager-muncitor)

- Unul sau mai multe procese master generează muncă și îl alocă proceselor lucrătorilor.
  - sarcinile pot fi alocate a priori dacă managerul poate estima dimensiunea sarcinilor sau
  - dacă o mapare aleatorie poate face o muncă adecvată de echilibrare a sarcinii.
  - În alt scenariu, lucrătorilor li se atribuie lucrări mai mici la diferite momente.
    - de preferat dacă consumatorul necesită timp pentru a genera muncă și, prin urmare, nu este de dorit ca toți lucrătorii să aștepte până când comandantul a generat toate piesele de lucru.
- În unele cazuri, munca poate fi necesară în faze, iar munca în fiecare fază trebuie să se termine înainte de a putea fi generată lucrarea în următoarele faze.
  - În acest caz, managerul poate determina sincronizarea tuturor lucrătorilor după fiecare fază.
- De obicei, nu se face premaparea procesului; orice lucrător poate face orice job.
- Modelul poate fi generalizat la un model ierarhic sau multi-nivel manager-muncitor:
  - managerul de nivel superior alimentează bucăți mari de sarcini către managerii de nivelul al 2lea, care subdivizează sarcinile între proprii muncitori și pot efectua o parte din munca ei înșiși.
- Acest model este în general la fel de potrivit pt. paradigmele cu spații de adresă partajate sau de transmitere a mesajelor, deoarece interacțiunea este în mod natural în 2 sensuri;
  - managerul știe că trebuie să renunțe la muncă și
  - lucrătorii știu că trebuie să obțină muncă de la manager.
- Trebuie avut grijă:
  - pentru a vă asigura că masterul nu devine un blocaj (se poate întâmpla dacă sarcinile sunt prea mici / rapide)
  - granularitatea sarcinilor: costul lucrărilor >> costul transferului lucrărilor și costul sincronizării.
  - interacțiunea asincronă poate ajuta:
    - interacțiunea suprapusă și calculul asociat cu generarea de muncă de către master.
    - De asemenea, poate reduce timpul de așteptare dacă natura cererilor de la lucrători nu este administrată.

Conducta sau modelul producator-consumator

- Un flux de date este transmis printr-o succesiune de procese, fiecare îndeplinind o sarcină asupra sa.
- Această execuție simultană de diferite programe pe un flux de date se numește paralelism de flux.
- Cu excepția procesului care inițiază conducta, sosirea de noi date declanșează executarea unei noi sarcini de către un proces în conductă.
- Procesele ar putea forma astfel de conducte sub formă de tablouri liniare sau multidimensionale, arbori sau grafice generale cu sau fără cicluri.
- O conductă este un lanț de producători și consumatori.
  - Fiecare proces din conductă poate fi vizualizat ca:
    - un consumator al unei secvențe de elemente de date pentru procesul precedent în conductă și
    - ca producător de date pentru procesul care îl urmează în conductă.
- Conducta nu trebuie să fie un lanț liniar; poate fi un graf direcționat.
- Modelul conductei implică de obicei o mapare statică a sarcinilor pe procese.
- Echilibrarea sarcinii este o funcție a granularității sarcinii.
- Cu cât granularitatea este mai mare, cu atât este mai lungă pentru a umple conducta,
  - pentru declanșarea produsă de primul proces din lanț să se propage până la ultimul proces, menținând astfel o parte din procesele în așteptare.
- O granularitate prea fină poate crește surplusul interacțiunii, deoarece procesele vor trebui să interacționeze pentru a primi date proaspete după calcule mai mici.
- Cea mai frecventă tehnică de reducere a interacțiunilor aplicabilă acestui model este suprapunerea interacțiunii cu calculul.
- Un exemplu de conductă bidimensională este algoritmul de factorizare LU paralel.

Modele hibride

- În unele cazuri, mai mult de un model poate fi aplicabil problemei disponibile, rezultând într-un model de algoritm hibrid.
- Se poate compune un model hibrid:
  - oricare dintre mai multe modele aplicate ierarhic
  - sau mai multe modele aplicate secvențial pe diferite faze ale unui algoritm paralel.
- În unele cazuri, o formulare a algoritmilor poate avea caracteristici ale mai multor un model de algoritm.
  - De exemplu, datele pot curge în mod pipelinat într-un model ghidat de un grafic de dependență de sarcină.
  - Într-un alt scenariu, calculul major poate fi descris printr-un grafic de dependență de sarcină, dar fiecare nod al graficului poate reprezenta un supertask cuprinzând mai multe subtaschi care pot fi potrivite pentru paralelismul paralel cu date sau pipelinate.
- Quicksort paralel este una dintre aplicațiile pentru care un model hibrid este ideal.

Aplicarea modelul parelel în date- ex. 1

- Luați în considerare problema construirii listei tuturor numerelor prime din intervalul [1, n] pentru un număr întreg dat n> 0.
- Un algoritm simplu care poate fi utilizat pentru acest calcul este ciurul lui Eratostene.
  - Începeți cu lista numerelor 1, 2, 3, 4 ,. . . , n reprezentat ca un vector de biți „marcaj” inițializat la 1000. . . 00.
  - În fiecare pas, următorul număr nemarcat m (asociat cu un element 0 în m al vectorului de biți al mărcii) este un prim.
  - Găsiți acest element m și marcați toți multiplii de m începând cu m^2.
  - Cand $m^2 > n$, calculul se oprește și toate elementele nemarcate sunt numere prime.
- Lista numerelor și numarul prim curent sunt stocate într-o memorie partajată care este accesibilă tuturor procesoarelor.
- Un procesor inactiv se referă pur și simplu la memoria partajată, actualizează numarul prim current și folosește indexul privat pentru a parcurge lista și a marca multiplii acelui prim.
  - Diviziunea muncii este astfel autorregulată.
- Fie o abordare paralelă cu date în care vectorul de biți care reprezintă n întregi este împărțit în p segmente de lungime egală, cu fiecare segment stocat în memoria privată a unui procesor.
  - Toate primele ale căror multipli trebuie marcați se află în Procesorul 1, care acționează ca un coordonator;
  - Acesta găsește următorul prim și îl transmite tuturor celorlalte procesoare, care apoi continuă să marcheze numerele din sublistele lor.

Aplicarea modelul parelel în date- ex. 1

- Timpul general al soluției constă acum din două componente:
  - timpul petrecut în transmiterea primelor selectate către toate procesoarele (timpul de comunicare) și
  - timpul petrecut de procesatorii individuali care marchează sublistele lor (timpul de calcul).
- De obicei, timpul de comunicare crește odată cu numărul procesoarelor, deși nu neapărat în mod liniar.
  - Datorită surplusului comunicării, adăugarea mai multor procesoare peste un anumit număr optim nu duce la îmbunătățirea timpului total al soluției sau la o accelerare rapidă.
- Fie soluția paralelă in date, dar cu timpul de I / O de date inclus și în timpul total al soluției.
  - Presupunând simplitatea că timpul I / O este constant și ignorând timpul de comunicare, timpul I / O va constitui o fracțiune mai mare din timpul soluției generale, deoarece partea de calcul este accelerată adăugând tot mai multe procesoare.
  - Dacă I / O durează 100 de secunde, să spunem, atunci există o mică diferență între a face partea de calcul în 1 secundă sau în 0,01 secundă.
  - Astfel de porțiuni „secvențiale” sau „de neegalat” de calcule limitează sever viteza care poate fi obținută cu procesarea paralelă.

Aplicarea modelul parelel în date- ex. 2

- problema distribuirii unui vector x cu n elemente x0, x1 ,. . . , xn − 1 pe procesoare p.
- Distribuție ciclică:
  - **0** : *x*0 *x*5 *x*10 *x*15 *x*20
  - **1** : *x*1 *x*6 *x*11 *x*16 *x*21
  - **2** : *x*2 *x*7 *x*12 *x*17
  - **3** : *x*3 *x*8 *x*13 *x*18
  - **4** : *x*4 *x*9 *x*14 *x*19
- Distribuție bloc:
  - **0** : *x*0 *x*1 *x*2 *x*3 *x*4
  - **1** : *x*5 *x*6 *x*7 *x*8 *x*9
  - **2** : *x*10 *x*11 *x*12 *x*13 *x*14
  - **3** : *x*15 *x*16 *x*17 *x*18 *x*19
  - **4** : *x*20 *x*21
- Distributie bloc-ciclica:
  - Se îmbunătățește distribuția bloc prost echilibrată,
  - Restabilește o localitate de acces la memoria de referință a distribuției ciclice.
    - **0** : *x*0 *x*1 *x*10 *x*11 *x*20 *x*21
    - **1** : *x*2 *x*3 *x*12 *x*13
    - **2** : *x*4 *x*5 *x*14 *x*15
    - **3** : *x*6 *x*7 *x*16 *x*17
    - **4** : *x*8 *x*9 *x*18 *x*19

Aplicarea modelul parelel în date- ex. 3

- Data mining diferă de interogările standard de baze de date, întrucât obiectivul său este de a identifica tendințe și segmentări implicite, mai degrabă decât de a căuta datele solicitate de o interogare directă și explicită.
  - De exemplu, găsirea tuturor clienților care au cumpărat mâncare pentru pisici în ultima săptămână nu reprezintă o extragere a datelor
  - Segmentarea clienților în funcție de relațiile din grupa lor de vârstă, venitul lunar, preferințele în ceea ce privește mâncarea pentru animale de companie, mașini, este.
- Un tip particular de extragere a datelor este explorarea pentru asociații.
  - Scopul este
    - să descopere relații (asociații) între informațiile legate de diferiți clienți și tranzacțiile lor
    - pentru a genera reguli pentru inferența comportamentului clienților.
  - De exemplu, baza de date poate stoca pentru fiecare tranzacție lista articolelor achiziționate în acea tranzacție.
    - Scopul explorarii poate fi determinarea asocierilor între seturile de articole achiziționate în mod obișnuit, care tind să fie achiziționate împreună;
    - De exemplu, probabilitatea condiționată de faptul că un anumit set de articole se găsește într-o tranzacție, dat fiind faptul că un set diferit de elemente se găsește în acea tranzacție.

Aplicarea modelul parelel în date- ex. 3

- Data fiind o bază de date în care înregistrările corespund tranzacțiilor de cumpărare a clienților, fiecare tranzacție are un cod de tranzacție și un set de atribute sau elemente, de ex. articolele cumpărate.
- Primul obiectiv în explorarea pentru asociații este de a examina baza de date și de a determina ce seturi de k elemente, să zicem, se găsesc să apară împreună în mai mult de o fracție de prag dată a tranzacțiilor.
  - Un set de articole (orice dimensiune) care apar împreună într-o tranzacție se numește itemset,
  - Un intemset care se găsește în mai mult de procentul de prag al tranzacțiilor e numit itemset mare.
- Odată ce seturile mari de dimensiuni k sunt găsite cu frecvențele lor de apariție în baza de date a tranzacțiilor, determinarea regulilor de asociere dintre ele este destul de ușoară.
- Prin urmare, problema pe care o luăm în considerare se concentrează pe descoperirea seturilor mari de dimensiuni k și a frecvențelor acestora.
- O modalitate simplă de a rezolva problema este:
  - pentru a determina mai întâi seturile mari de dimensiuni unu.
  - Dintre acestea, se pot construi un set de seturi de articole candidate cu dimensiunea a două elemente și frecvența lor de apariție în baza de date a tranzacțiilor, folosind ideea de bază că dacă un itemset este mare, atunci toate subseturile sale trebuie să fie, de asemenea, mari
  - Rezultă o listă de seturi mari de dimensiuni două.
  - Procesul se repetă până când obținem seturile mari de dimensiuni k.
- Există concurență în examinarea seturilor de articole mari de dimensiunea k-1 pentru a determina seturile de articole candidate cu dimensiunea k și în numărarea numărului de tranzacții din baza de date care conține fiecare dintre seturile candidate.

Sabloane pentru blocuri de calcule

1. Calcule in semigrup (reducere, fan-in)
2. Calcul paralel al prefixului
3. Ruatarea pachetelor
4. Difuzarea si multi-difuzare
5. Sortare

Calcule in semi-grup (reducere, fan-in)

- Fie o un operator binar asociativ;
  - adica, (x o y) o z= x o (y o z) pentru toti x, y, z in S.
- Un semigrup este o pereche (S, o), unde S este o multime de elemente pe care este definit o.
- *Calculul in semigrup* (numit si *reducere* sau *fan-in*) este definit astfel:
- Data fiind o lista de valori x_0, x_1, . . . , x_{n-1}, se calculeaza x_0 ox_1 o. . . o x_{n-1} .
- Exemple comune pentru operatorul o includ +, *, ∩, max, min.
- Operatorul o poate fi sau nu comutativ,
  - sdica poate sau nu sa satisfaca x o y = y o x
  - în timp ce algoritmul paralel poate calcula bucăți ale expresiei folosind orice schemă de partiționare, bucățile trebuie să fie combinate în cele din urmă de la stânga la dreapta.

Calculul paralel al prefixului si rutarea pachetelor

- **Calcul paralel al prefixului.**
- Aceleasi notatii ca in slideul anterior,
- Un calcul de prefix in paralel este definit ca evaluând simultan toate prefixele expresiei x_0 ox_1 o. . . o x_{n-1}
  - adica, x_0, x_0 o x_1, x_0 o x_1 o x_2, . . . , x_0 o x_1 o. . . o x_{n-1} .
  - prefixul *i* a expresiei este $s_i = x_0$ ox_1 o. . . o x_i.

- **Rutarea pachetelor.**
  - Un pachet de informații se află la Procesorul i și trebuie trimis la Procesorul j.
  - Problema constă în dirijarea pachetului prin procesoare intermediare, dacă este nevoie, astfel încât să ajungă la destinație cât mai repede posibil.
  - Problema devine mai dificilă când mai multe pachete se află la diferite procesoare, fiecare având propria destinație.
  - În acest caz, rutele pachetelor pot interfera între ele, pe măsură ce trec prin procesoare intermediare comune.
  - Când fiecare procesor are cel mult un pachet de trimis și un pachet de primit, problema de rutare a pachetelor se numește comunicare unu-la-unu sau rutare 1-1.

Difuzare si sortare

- **Broadcasting.**
  - Având în vedere o valoare cunoscută la un anumit procesor i, difuzați-o la toate procesoarele cât mai repede posibil, astfel încât la final, fiecare procesor să aibă acces la „sau să știe”.
  - Aceasta este uneori denumită comunicare individuală.
  - Cazul mai general al acestei operațiuni, adică comunicarea unu-la-mulți, este cunoscut sub numele de multicasting.
  - Din punct de vedere al programării, realizăm alocările xj: = a pentru 1 ≤j ≤p (difuzare) sau pentru j în G (multidifuzare), unde G este grupul multicast și xj este o variabilă locală în procesorul j.

- **Sortare.**
  - În loc să sortăm un set de înregistrări, fiecare cu o cheie și elemente de date, ne concentrăm pe sortarea unui set de chei pentru simplitate.
  - Problema noastră de sortare este definită astfel: Data fiind o listă de n chei x0, x1,. . . , xn – 1 și o ordine totală ≤ asupra valorilor cheii, rearanjați cele n chei ca o ordine nedescrescatoare.
  - Avem în vedere doar sortarea cheilor în conversie, într-o manieră simplă, într-una pentru sortarea cheilor în ordine neascendentă sau pentru sortarea înregistrărilor.

Algoritmi pentru tablouri liniare– calcul in semigrup

- Conectare in linie
- Un caz special al calculului in semigrup este determinarea maximului.
  - Fiecare procesor p deține inițial o valoare, iar scopul nostru este ca fiecare procesor să cunoască cea mai mare dintre aceste valori.
  - O variabilă locală, maximă până acum, poate fi inițiată la valoarea datelor proprii ale procesorului.
  - În fiecare etapă, un procesor trimite valoarea maximă până în prezent celor doi vecini.
  - Fiecare procesor, la primirea valorilor de la vecinii săi stânga și dreapta, își stabilește valoarea maximă până acum la cea mai mare dintre cele trei valori, adică max (stânga, proprie, dreaptă).
  - În cel mai rău caz, sunt necesari pași de comunicare p - 1 (fiecare implicând trimiterea valorii unui procesor către ambii vecini) și aceiași pași de comparație fără trei căi.
  - Acesta este cel mai bun pe care îl putem spera, având în vedere că diametrul unui tablou liniar al procesorului p este D = p - 1 (legătură inferioară bazată pe diametru).
- Pentru un calcul general de semigrup,
  - procesorul de la capătul stâng al tabloului (cel fără vecin stânga) devine activ și trimite valoarea datelor la dreapta (inițial, toate procesoarele sunt inactive sau inactive).
  - La primirea unei valori de la vecinul său stâng, un procesor devine activ, aplică operația de semigrup o la valoarea primită de la stânga și la propria valoare de date, trimite rezultatul la dreapta și devine din nou inactiv.
  - Valul de activitate se propagă spre dreapta până când procesor cel mai din dreapta obține rezultatul dorit.
  - Rezultatul calculului este apoi propagat spre stânga la toate procesoarele. În total, sunt necesare 2p - 2 etape de comunicare.

Algoritmi pentru tablouri liniare– prefix paralel

- Presupunem că rezultatul prefixului ith este obținut la procesorul ith, $0 \leq i \leq p - 1$.
- Algoritmul general al semigrupului:
  - realizează mai întâi un semigrup de calcul și
  - apoi face o difuzare a valorii finale către toate procesoarele.
  - Astfel, avem un alg. Pt. calculul prefixului paralel cu p-1 pași de comunicare / combinare.
- O variantă a calculului prefixului paralel, în care procesorul i se încheie cu rezultatul prefixului până la valoarea (i - 1), este uneori utilă.
  - Acest calcul al prefixului diminuat poate fi efectuat la fel de ușor dacă fiecare procesor deține valoarea primită de la stânga, decât cea pe care o trimite spre dreapta.
- Până acum, am presupus că fiecare procesor deține un singur element de date.
  - Extinderea algoritmilor semigrupului și a prefixului paralel la cazul în care fiecare procesor deține inițial mai multe elemente de date este simplu.
- Într-o calculare paralelă a prefixului cu fiecare procesor care deține inițial două elemente de date, algoritmul este format din:
  - fiecare procesor care efectuează un calcul prefix pe propriul set de date cu dimensiunea n / p (acest lucru necesită n / p - 1 pași de combinare),
  - făcând apoi un calcul al prefixului paralel diminuat pe tabloul liniar ca mai sus (p-1 pași de comunicare / combinare) și
  - în cele din urmă, combinarea prefixului local rezultat din acest ultim calcul cu prefixele calculate local (n / p pași de combinare).
  - În total, sunt necesare 2n / p + p– 2 pași de combinare și pași de comunicare p - 1.

Algoritmi pt. tablouri liniare–rutarea pachetului si difuzare

- ***Rutarea pachetului.***
  - Pentru a trimite un pachet de informații de la Procesorul i la Procesorul j pe un tablou liniar, pur și simplu atașăm o etichetă de rutare cu valoarea j - i la acesta.
  - Semnul unei etichete de rutare determină direcția în care ar trebui să se deplaseze (+ = dreapta, - = stânga) în timp ce amploarea acesteia indică acțiunea care trebuie efectuată (0 = elimirare pachet, non-zero = trimit pachet).
  - La fiecare înaintare, mărimea etichetei de rutare este - cu 1.
  - Mai multe pachete originare de la diferite procesoare pot curge spre dreapta și spre stânga în pauză, fără a interveni vreodată între ele.

- ***Difuzare.***
  - Dacă Procesorul i dorește să difuzeze o valoare a tuturor procesoarelor, trimite un mesaj rbcast (a) (citește difuzarea la dreapta) către vecinul său drept și un mesaj lbcast (a) către vecinul său stâng.
  - Orice procesor care primește un mesaj rbcast (a), pur și simplu copiază valoarea a și transmite mesajul către vecinul său drept (dacă există).
  - În mod similar, primirea unui mesaj lbcast (a) determină copierea locală a lui a și mesajul transmis către vecinul din stânga.
  - Cel mai rău caz nr. etapelor de comunicare pentru difuzare este p - 1.

Algorithms for linear array – sorting

- If the key values are already in place, one per processor, then an algorithm known as *odd–even transposition* can be used for sorting.
  - A total of *p* steps are required.
  - In an odd-no. step, odd-no. processors compare values with their even-no. right neighbors.
  - The two processors exchange their values if they are out of order.
  - Similarly, in an even-numbered step, even-numbered processors compare–exchange values with their right neighbors.
  - Worst case: largest key value in P0 and must move all the way to the other end of the array.
    - This needs *p* – 1 right moves.
  - One step must be added because no movement occurs in the first step.
  - Of course one could use even–odd transposition, but this will not affect the worst-case time complexity of the algorithm.
- Note that the odd–even transposition algorithm uses *p* processors to sort *p* keys in *p* compare–exchange steps.
- In most practical situations, the number *n* of keys to be sorted (the *problem size*) is greater than the number *p* of processors (the *machine size*).
- The odd–even transposition sort algorithm with *n*/*p* keys per processor is as follows.
  - First, each processor sorts its list of size *n*/*p* using any efficient sequential sorting algorithm.
  - Next, the odd–even transposition sort is performed as before, except that each compare–exchange step is replaced by a merge–split step in which
    - the two communicating processors merge their sublists of size *n*/*p* into a single sorted list of size 2*n*/*p* and
    - then split the list down the middle, one processor keeping the smaller half and the other, the larger half.

Algorithms for binary trees

- ***Semigroup Computation.***
  - Each inner node receives two values from its children (if each of them has already computed a value or is a leaf node),
  - applies the operator to them, and passes the result upward to its parent.
  - After ceil(log2 *p*) steps, the root processor will have the computation result.
  - All procs are then notified of the result through a broadcasting operation from the root.
  - Total time: 2ceil(log2*p*) steps.
- ***Parallel Prefix Computation.***
  - Can be done optimally in 2ceil(log 2 *p)* steps
  - Consists of an upward propagation phase followed by downward data movement.
  - The upward propagation phase is identical to the upward movement of data in semigroup computation.
  - At the end of this phase, each node will have the semigroup comp.res.for its subtree.
  - The downward phase is as follows.
    - Each processor remembers the value it received from its left child.
    - On receiving a value from the parent, a node passes the value received from above to its left child & the combination of this value and the one that came from the left child to its right child.
  - The root is viewed as receiving the identity element from above and thus initiates the downward phase by sending the identity element to the left & the value received from its left child to the right.
  - At the end of the downward phase, the leaf processors compute their respective res.

Algoritmi pentru tablouri liniare– sortare

- Putem folosi un algoritm similar cu sortarea bulelor:
  - permite elementelor mai mici din frunze să „buleze” mai întâi către procesorul rădăcină,
  - permițând astfel rădăcinii să „vadă” toate elementele de date în ordine nedescendentă.
  - rădăcina trimite apoi elementele către nodurile frunzelor în ordinea corectă.
- Înainte de a descrie partea algoritmului care se ocupă de balonarea ascendentă a datelor, să ne ocupăm de mișcarea descendentă mai simplă.
  - Această mișcare în jos este coordonată cu ușurință dacă fiecare nod cunoaște numărul nodurilor frunzelor din subtree stânga.
  - Dacă ordinea de rang a elementului primit de sus (păstrat într-un contor local) nu depășește numărul de noduri de frunze la stânga, atunci elementul de date este trimis la stânga.
  - În caz contrar, este trimis în dreapta.
- Presupune implicit că datele trebuie să fie sortate de la stânga la dreapta în frunze.
- Se poate realiza mișcarea ascendentă a datelor din algoritmul de sortare de mai sus:
  - Inițial, fiecare frunza are un singur element de date și toate celelalte noduri sunt goale.
  - Fiecare nod interior are spațiu de stocare pentru 2 valori, care migrează în sus din subarbore sale din stânga și din dreapta.

    if you have 2 items
    then do nothing
    else if you have 1 item that came from the left (right)
        then get the smaller item from the right (left) child
        else get the smaller item from each child
- Algoritmul de sortare de mai sus necesită timp liniar în numărul de elemente care trebuie sortate.

Algoritmi pentru grile 2d

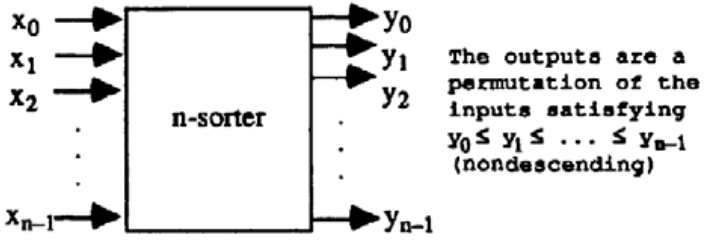
- ***Calcule in semigrup.***
  - Efectuați calculul semigrup în fiecare rând și apoi în fiecare coloană.
  - De exemplu, la găsirea maximului unui set de valori p, stocate cate una pe procesor,
    - maximele de rând sunt calculate mai întâi și puse la dispoziția fiecărui procesor din rând.
    - apoi maximele de coloane sunt identificate.
  - Același proces poate fi utilizat pentru calcularea sumei numerelor p.
  - Pentru un calcul general de semigrup cu o operație noncomutativă, numerele p trebuie să fie stocate în ordine pe rând pt.ca acest algoritm să funcționeze corect.
- ***Parallel Prefix Computation.***
  - Poate fi realizat în trei faze, presupunând că procesoarele (și valorile stocate) sunt indexate în ordine rând-major:
    1. faceți un calcul prefix paralel pe fiecare rând,
    2. efectuați un calcul al prefixului paralel diminuat în coloana din dreapta și
    3. difuzați rezultatele din coloana din dreapta la toate elementele din rândurile respective și combinați cu valoarea prefixului rândului calculat inițial.
  - De exemplu, în sumele prefixate,
    - sumele din prefixul din primul rând sunt calculate de la stânga la dreapta.
    - procesoarele din coloana cea mai dreaptă rețin sumele de rând.
    - un calcul al prefixului diminuat în această ultimă coloană produce suma tuturor rândurilor precedente din fiecare procesor.
    - combinând suma tuturor rândurilor precedente cu sumele prefixului rândului, rezultă sumele totale ale prefixului.

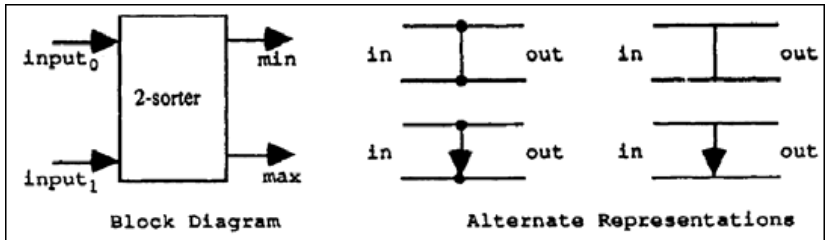
Algoritmi pentru grile 2d

- ***Difuzare.***
  - In 2 faze:
  1. transmite pachetul către fiecare procesor din rândul nodului sursă și
  2. difuzat în toate coloanele.
  - Dacă mai multe valori sunt difuzate de un procesor, atunci mișcările de date necesare pot fi canalizate, astfel încât fiecare difuzare suplimentară necesită doar un pas suplimentar.
- ***Sortare.***
  - versiunea simplă a unui algoritm de sortare cunoscut sub numele de shearsort.
  - Algoritmul constă din plafon (log 2r) + 1 faze într-o grilă 2D cu r rânduri.
  - În fiecare fază, cu excepția ultimei, toate rândurile sunt sortate independent într-o ordine de tip sarpe:
    - rândurile cu numere pare 0, 2, ... de la stânga la dreapta,
    - rândurile impare cu numere impare 1, 3,. . . de la dreapta la stânga.
  - Apoi, toate coloanele sunt sortate independent de sus în jos.
  - În faza finală, rândurile sunt sortate independent de la stânga la dreapta.
  - Necesarul algoritmului de forfecare trebuie să compare pașii de schimb pentru sortarea în rândul major.

Retele de sortare

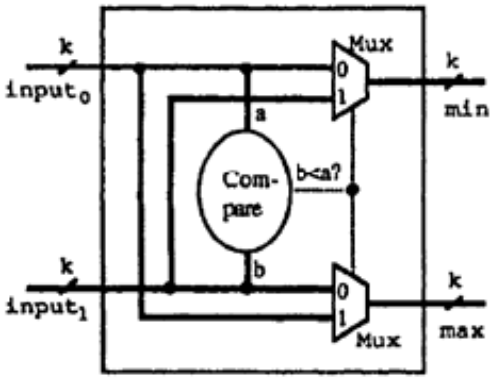
- Proiectele la nivel de circuit pentru procesare paralelă sunt în mod necesar specifice problemei sau cu scop special.
- O rețea de sortare este un circuit care primește n intrări și le permite să producă n ieșiri, astfel încât ieșirile să satisfacă y0 ≤y1 ≤ y2 ≤. . . in-1.
- Consultați o astfel de rețea de sortare n-input n-output ca n-sorter
- La fel cum mulți algoritmi de sortare se bazează pe compararea și schimbul de perechi de chei, putem construi un n-sorter din blocuri de construcție cu 2 sortări.
- Un 2-sorter compară cele două intrări ale sale (fie input0 și input1) și le comandă la ieșire, schimbând ordinea, dacă este nevoie, punând valoarea mai mică, min (input0, input1), înainte de valoarea mai mare, max (input0, input1).

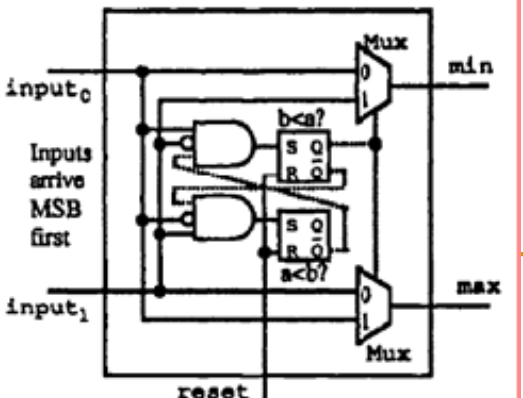




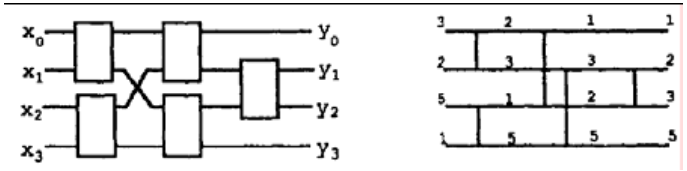
Realizarea hardware a 2-sortatorului

- Dacă vedem intrările ca fiind numere întregi nesemnate care sunt furnizate 2-sorter în formă de bi-paralel, atunci 2-sorter poate fi implementat folosind un comparator și doi multiplexori 2 la 1, așa cum se arată în fig. din stânga.
- Când cheile sunt lungi sau când trebuie să implementăm o rețea de sortare cu multe intrări pe un singur cip VLSI, intrarea în paralel cu biți devine imposibilă, având în vedere limitările pinului.
- Fig. din dreapta înfățișează o realizare hardware în serie de biți a sortatorului cu 2 flip-flops.
  - Flip-flops-urile sunt resetate la 0 la început.
  - Această stare reprezintă până acum cele două intrări egale. Celelalte două stări sunt 01 (intrarea superioară este mai mică) și 10 (intrarea inferioară este mai mică).
  - În timp ce sortatorul 2 este în starea 00 sau 01, intrările sunt transmise direct la ieșiri.
  - Când starea se schimbă la 10, intrările sunt schimbate, cu intrarea de sus dirijată spre ieșirea inferioară și invers.





Corectitudinea unui n-sortator



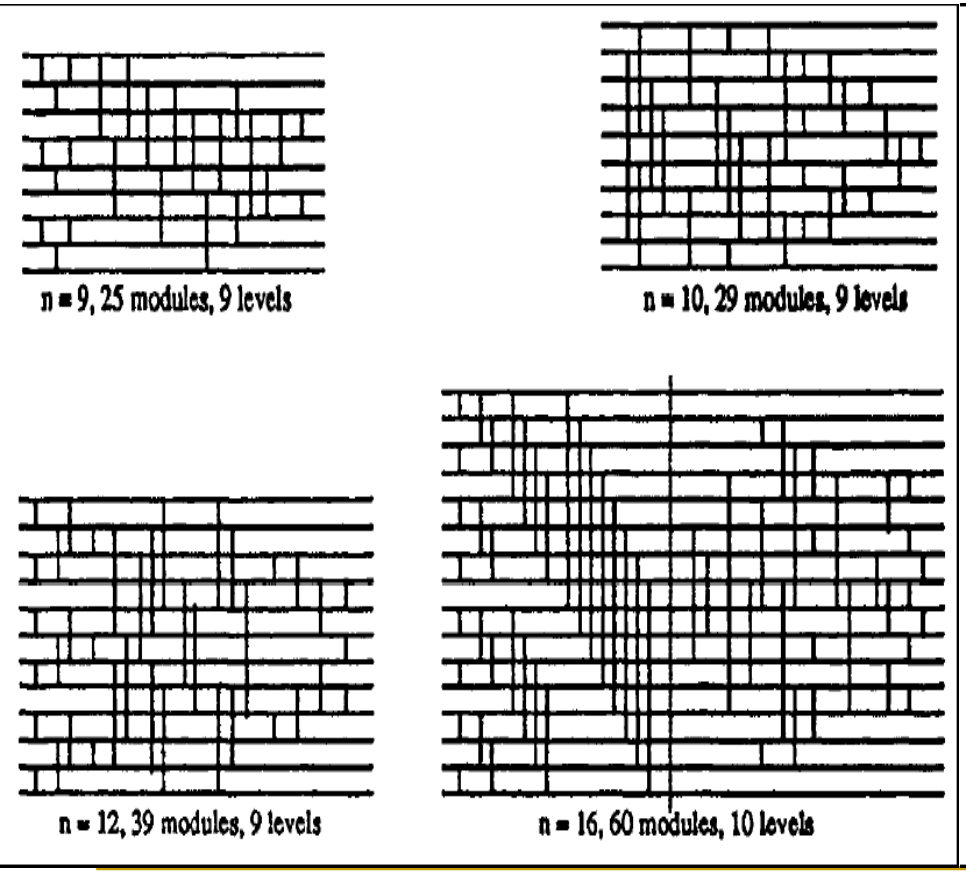
- Fig ilustrează un 4-sortator construit din blocuri de clădire cu 2 sortatoare: diagrama blocului și reprezentarea schematică.
- Reprezentarea schematică a sortatorului 4 arată valorile datelor purtate pe toate liniile atunci când se aplică secvența de intrare 3, 2, 5, 1.
- Cum verificăm dacă circuitul este de fapt un 4-sortator valid?
- Principiul zero-unu ne permite să facem acest lucru cu mult mai puțină muncă.
  - Un n-sortator evalid dacă ordonează corect toate secvențele 0/1 de lungime n.
- Folosind principiul zero-unu, corectitudinea sortatorului 4 poate fi verificată testându-l pentru cele 16 secvențe posibile de 0/1 de lungime 4.
  - Rețeaua sortează clar 0000 și 1111.
  - Sortează toate secvențele cu un singur 0, deoarece 0 „bule” până la linia superioară.
  - În mod similar, un singur 1 s-ar „scufunda” până la linia de jos.
  - Partea rămasă din dovadă tratează secvențele 0011, 0101, 0110, 1001, 1010, 1100, toate ducând la ieșirea corectă 0011.

Cel mai bun n-sortator?

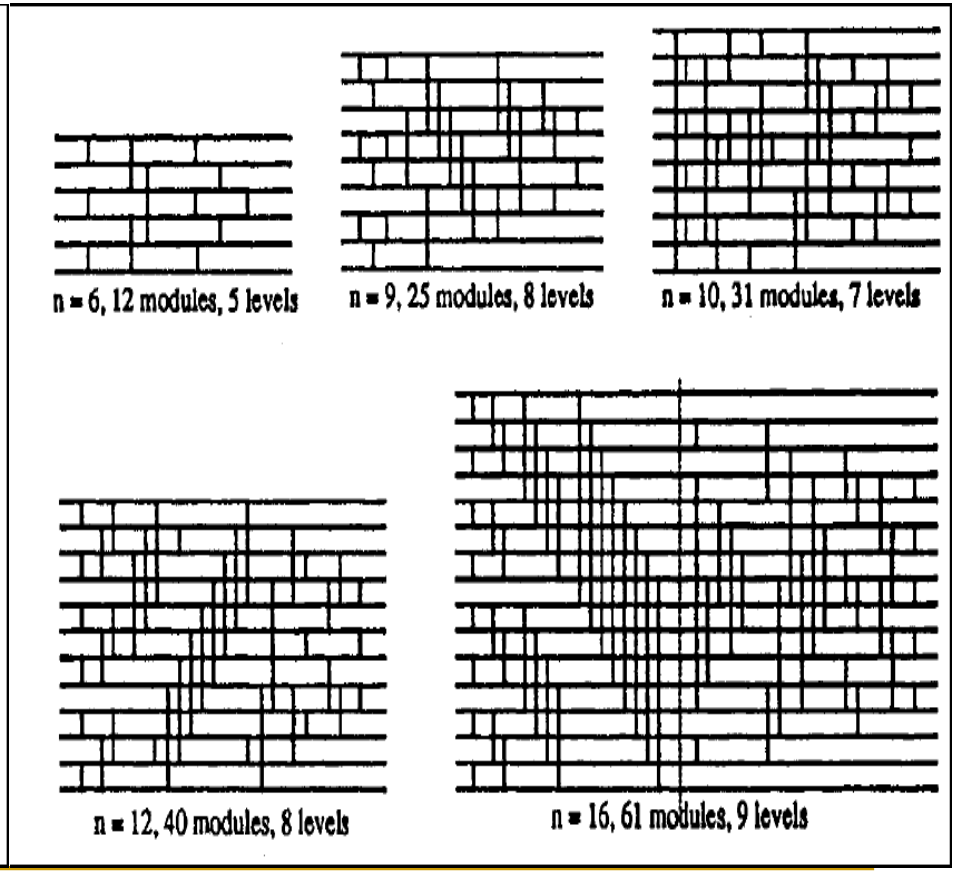
- Ce inseamna “cel mai bun”?
- Metrici
  - Cost: numărul total de blocuri de 2-sortatori utilizate în proiectare
  - Întârziere: numărul de 2-sortatori pe drumul critic de la intrare la ieșire
- Obiective:
  1. Minimizarea costul x întârziere ar fi adecvată.
  2. Dacă putem reproiecta o rețea de sortare, astfel încât să fie cu 10% mai rapidă, dar doar cu 5% mai complexă, se consideră că reproiectarea este eficientă din punct de vedere al costurilor și se spune că circuitul rezultat este rentabil în timp (sau cel puțin mai mult decât cea original).

Exemple:

- Retele de sortare cu cost redus

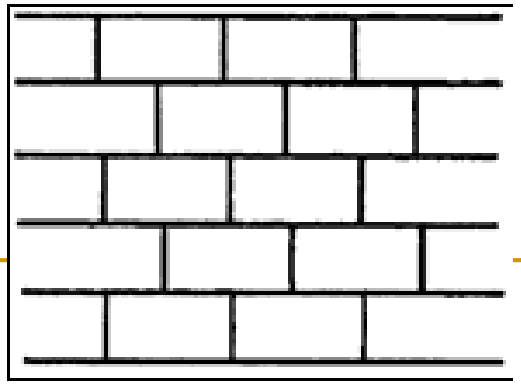


- Retele de sortare rapide



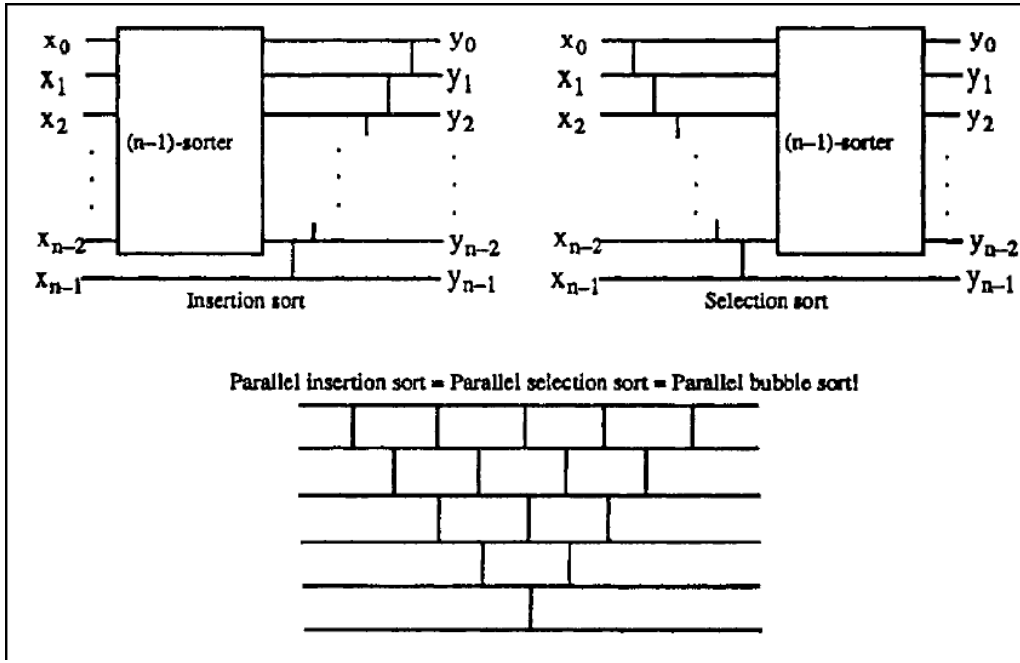
Cel mai bun n-sortator in general?

- Modelele cu cele mai mici costuri sunt cunoscute doar pentru n mici și încă nu există o metodă generală pentru derivarea sistematică a desenelor low cost
- Cele mai rapide modele posibile sunt cunoscute doar pentru n mici; rețelele de sortare eficiente din punct de vedere al timpului sunt chiar mai dificile.
- Există multe modalități de proiectare a rețelelor de sortare, ceea ce duce la rezultate diferite în ceea ce privește meritul.
  - De exemplu, Fig. Arată un 6-sortator al cărui design se bazează pe algoritmul de sortare a transpoziției impar-par discutat în legătură cu sortarea pe o linie liniară de procesoare.
    - Acest design „perete de cărămidă” oferă avantaje în ceea ce privește ușurința cablurilor (deoarece firele sunt scurte și nu se încrucișează).
    - Este destul de ineficient, deoarece folosește n x floor(n / 2) module și are n unități de întârziere.
    - Costul x întârziere este O(n^3).



Retele de sortare prin design recursiv?

- O modalitate de a sorta n intrări este de a sorta primele n - 1 intrări, să spunem, apoi de a insera ultima intrare în locul propriu.
- Ex:
  - Sortarea prin insertie (Fig. stanga)
  - Sortarea prin selectie (Fig. dreapta) –sortarea bulelor
- Ineficient:
  - cost O(n^2)
    (> O(n log n) prin divizare)
  - intarziere O(n)
    (> O(log n) prin divizare)



Rețea de sortare bazată pe inserare sau selecție + recursivitate

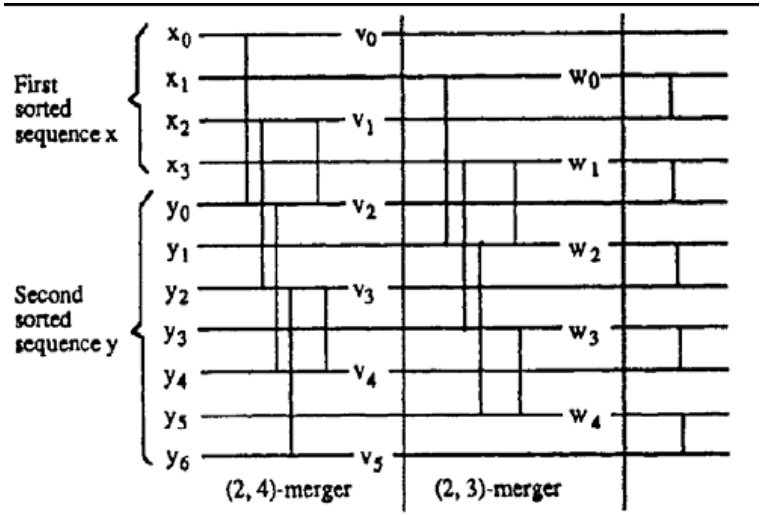
Retele practice de sortare: reteaua Batcher

- Suboptimala: Cost x intarziere = $O(n \log^4 n)$
- Dar rețelele de sortare ale lui Batcher sunt destul de eficiente.
- Încercările de a proiecta rețele mai rapide sau mai puțin complexe pentru valorile specifice ale n au obținut doar îmbunătățiri marginale față de construcția lui Batcher atunci când n este mare.
- Exemple:

1. Sortarea bazată pe tehnica de fuziune impar-par
2. Sortare bazată pe secvențe bitonice

Sortarea bazată pe tehnica de fuziune impar-par

- Un (*m* ,*m'*)-merger este un circuit care îmbină două secvențe sortate de lungimi *m* si *m'* într-o singură secvență sortată de lungime *m* + *m'.*
- Fie două secvențe sortate $x_0<=x_1<=\ldots x_{m-1}$, $y_0<=y_1<=\ldots y_{m'-1}$
- Fuziunea pară - par uniformă se realizează prin comasarea separată a elementelor indexate și impar a celor două liste:
  - $x_0,x_2,\ldots x_{2ceil(m/2)-2}$ si $y_0,y_2,\ldots y_{2ceil(m'/2)-2}$ sunt contopite pentru a obține $v_0,v_1,\ldots v_{ceil(m/2)+ceil(m'/2)-1}$
  - $x_1,x_3,\ldots x_{2ceil(m/2)-1}$ si $y_1,y_3,\ldots y_{2ceil(m'/2)-1}$ sunt contopite pentru a obține $w_0,w_1,\ldots w_{ceil(m/2]+ceil(m'/2)-1}$
- Dacă acum comparăm - schimbăm perechile de elemente $w_0:v_1$, $w_1:v_2$, $w_2:v_3$,.... secvența rezultată $v_0w_0v_1w_1v_2w_2\ldots$ va fi complet sortata.
- v_0, despre care se știe că este cel mai mic element în general, este exclus din operațiunile finale de schimb-comparație
- Ex: (4, 7)-merger din Fig. folosește 16 module și are o întârziere de 4 unități

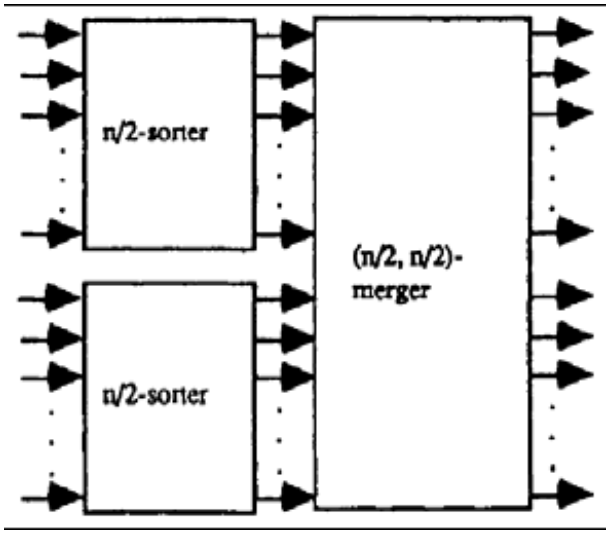


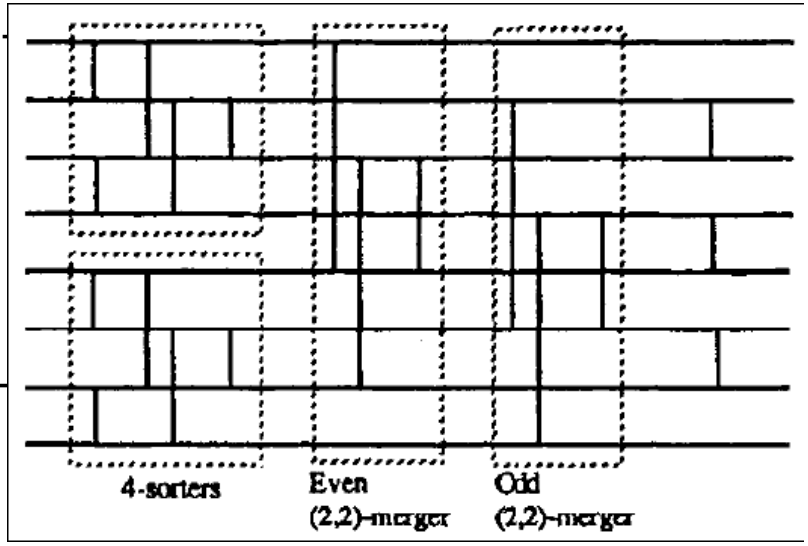
Cele trei segmente de circuit, separate prin linii verticale, corespund unui (2, 4) -merger pentru intrări indexate par, un (2, 3) -merger pentru intrări indexate impar, și operațiunile de comparare-schimb in paralel.

Fiecare dintre fuziunatorii mici pot fi proiectată recursiv. De exemplu, (2, 4) - merger este format din două (1, 2) -merger pentru intrări par- și impar-par, urmate de două comparații paralele - operațiuni de schimb.

Sortare recursivă bazată pe fuziune impar-par

- Înarmat cu un circuit eficient de fuziune, putem proiecta un n-sorter recursiv de la două n/2-sortere și o (n /2, n/2)-merger, ca în fig.
- 4-sorter din slide-ul anterior este o instanță a acestui design:
  - este format din două 2-sortatoare
  - urmat de un (2, 2) -merger
  - la rândul său, construit din două (1, 1) -mergere și o singură etapă de comparație-schimb.
- Un exemplu mai mare, corespunzător unui 8-sortator: Fig.
  - 4-sortatori sunt utilizati pentru a sorta prima și a doua jumătate a intrărilor separat,
  - Listele sortate apoi fuzionate de un (4, 4) -merger compus dintr-un (2, 2) –merger par, un (2, 2) -merger impar și o etapă finală a trei comparatori.



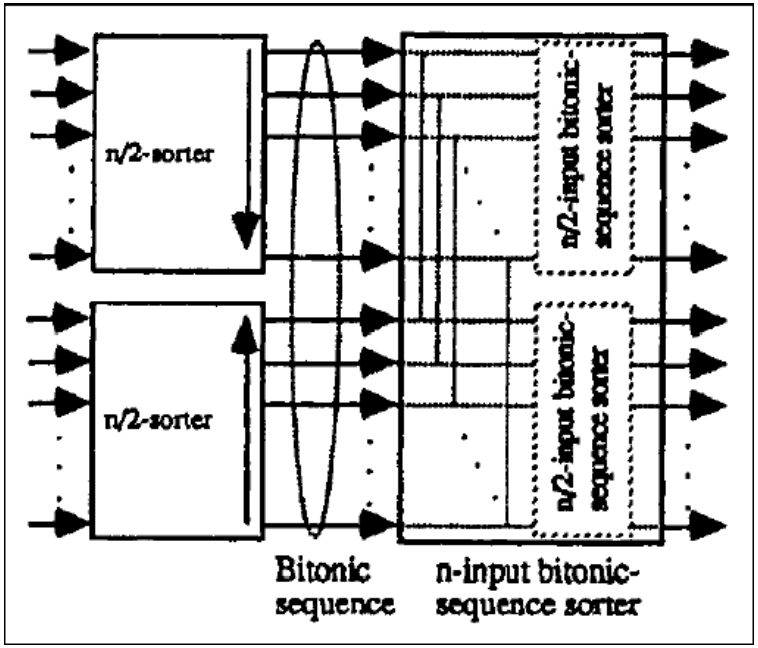


Intarziere si cost la reteaua de sortare Batcher

- Fuziunea impar-par (m, m) a lui Batcher, când m este o putere de 2, se caracterizează prin următoarea întârziere și cost:
  - $C(m)=2C(m/2)+m-1=(m-1)+2(m/2-1)+4(m/4-1)+\ldots. = m \log m +1$
  - $D(m)=D(m/2)+1=\log m+1$
  - Cost x delay = $O(m \log^2 m)$
- Rețelele de sortare Batcher bazate pe tehnica de fuziune impar-par sunt caracterizate de următoarea întârziere și cost:
  - $C(n)=2C(n/2)+(n/2)(\log(n/2))+1=n (\log n)^2/2$
  - $D(n)=D(n/2)+\log(n/2)+1=D(n/2)+\log n=\log n (\log n+1)/2$
  - Cost x delay = $O(n \log^4 n)$

Secvente bitonice

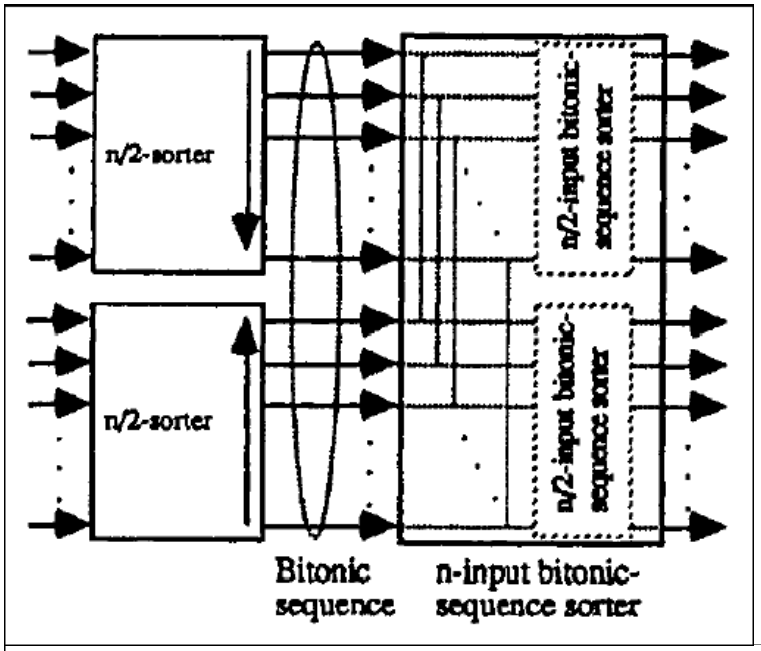
- O secvență bitonică este definită ca una care:
  - “se ridică apoi cade” $x_0<=x_1<=\ldots<=x_i>=x_{i+1}>=x_{i+2}>=\ldots>=x_{n-1}$,
  - “cade apoi se ridică” $x_0>=x_1>=\ldots>=x_i<=x_{i+1}<=x_{i+2}<=\ldots x_{n-1}$
  - sau se obține din primele două categorii prin deplasări sau rotații ciclice.
- Exemple
  - 1 3 3 4 6 6 6 2 2 1 0 0 se ridică apoi cade
  - 8 7 7 6 6 6 5 4 6 8 8 9 cade apoi se ridică
  - 8 9 8 7 7 6 6 6 5 4 6 8 secventa anterioara, rotita la dreapta cu 2 pozitii

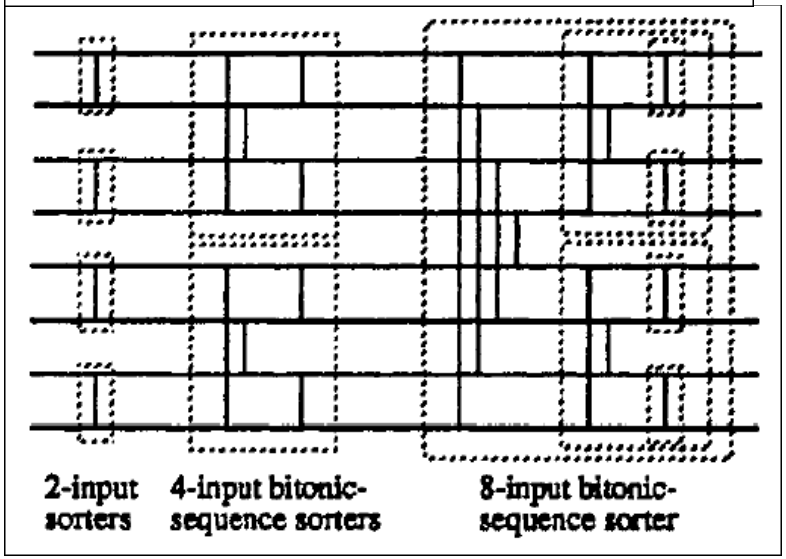


- Batcher a observat că dacă sortăm prima jumătate și a doua jumătate a unei secvențe în direcții opuse, așa cum este indicat de săgețile verticale din Fig, secvența rezultată va fi bitonică și poate fi astfel sortată printr-un sortator special de secvențe bitonice.
- Se dovedește că un sortator de secvențe bitonice cu n intrări are aceeași întârziere și cost cu un (n / 2, n / 2)-merger impar-par.
- Prin urmare, sortatoarele bazate pe noțiunea de secvențe bitonice (sortatoare bitonice) au aceeași întârziere și costuri ca cele bazate pe fuzionarea impar-par.

Sortator de secvente bitonice

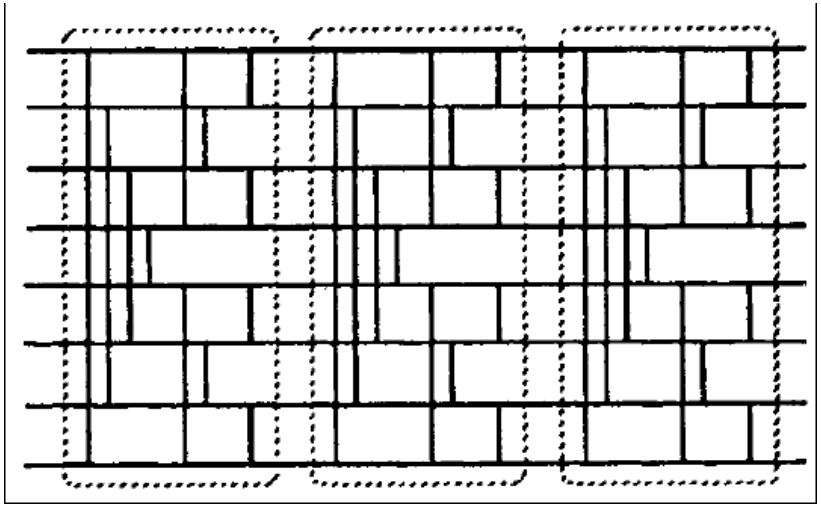
- Un sortator de secvențe bitonice poate fi proiectat pe baza afirmației că:
  - Dacă într-o secvență bitonică, comparăm-schimbăm elementele din prima jumătate cu cele din a doua jumătate, așa cum este indicat de comparatorii punctate din Fig. atunci:
    - fiecare jumătate din secvența rezultată va fi o secvență bitonică și
    - fiecare element din prima jumătate nu va fi mai mare decât orice element din a doua jumătate.
  - Astfel, cele două jumătăți pot fi sortate independent de sortatoare de secvențe bitonice mai mici pentru a finaliza procesul de sortare.
- Putem inversa direcția de sortare în n-sortator dacă reglăm în mod corespunzător conexiunile comparatoarelor punctate din Fig.1
- Fig.2 este prezentată o rețea de sortare bitonică completă cu opt intrări





Retele de sortare balansate periodic

- O clasă de rețele de sortare care au aceeași întârziere și cost ca rețelele de sortare Batcher
- Un *n*-sorter de acest tip constă din log *n* etape identice, fiecare avand sortatori de secvente bitonice cu (log *n*)-etape si *n*-input.
- Astfel, întârzierea și costul unui n-sorter de acest tip sunt $(\log n)^2$ si $n (\log n)^2/2$.



- Fig. prezintă un exemplu cu opt intrări:
  - întârziere mai mare (9 vs. 6) și cost mai mare (36 vs. 19) în comparație cu un 8-sortatorul Batcher
  - dar oferă următoarele avantaje:
  1. Structura este regulată și modulară (aspect VLSI mai ușor).
  2. Sunt posibile implementări mai lente, dar mai economice, prin reutilizarea blocurilor
  3. Utilizarea unui bloc suplimentar oferă toleranță la unele defecțiuni (schimburi ratate).
  4. Utilizarea a două blocuri suplimentare oferă toleranță la orice singur defect (un schimb ratat sau incorect).
  5. Trecerile multiple printr-o rețea defectă pot duce la sortarea corectă (degradare grațioasă).
  6. Proiectarea cu un singur bloc poate fi tolerată la erori adăugând o etapă suplimentară la bloc.