


Algoritmi paraleli și distribuiți

Sortare paralelă

Mitică Craus

Universitatea Tehnică "Gheorghe Asachi" din Iași

Cuprins

Introducere

- Există o vastă literatură de specialitate având ca subiect sortarea.
- Aceasta se explică prin faptul că sortarea apare ca substask în soluțiile algoritmice ale multor probleme.
- Problema poate fi enunțată astfel:
  Date fiind n elemente $a_0, a_1, \ldots, a_{n-1}$, *dintr-o mulțime U peste care este definită o relație de ordine totală* "$<$", *se dorește renumerotarea lor astfel încât* $a_i < a_j,\ i,j \in \{0,1,\ldots,n-1\},\ i < j$.
- Se presupune, pentru simplitate, că $a_i \neq a_j$, dacă $i \neq j$.

Algoritmul Muller-Preparata - descriere

- Autori: David E. Muller și Franco P. Preparata. Anul publicării: 1975
- Algoritmul este compus din trei faze:

1. Determinarea pozițiilor relative pentru fiecare pereche $\{a_i, a_j\}, i, j = 0, 1, \ldots, n-1$:
   - Notații: $pozitie\_relativa_{a_j}(a_i)$ și $pozitie\_relativa_{a_i}(a_j)$ desemnează poziția lui a_i, respectiv a_j în secvența (a_i, a_j) sortată crescător.
   - Dacă $a_i < a_j$, atunci $pozitie\_relativa_{a_j}(a_i) = 0$ și $pozitie\_relativa_{a_i}(a_j) = 1$.
   - Dacă $a_i > a_j$, atunci $pozitie\_relativa_{a_j}(a_i) = 1$ și $pozitie\_relativa_{a_i}(a_j) = 0$.
2. Calcularea pozitiilor finale ale elementelor a_i, $i = 0, 1, \ldots, n-1$.
3. Plasarea elementelor a_j pe pozitiile finale.

Algoritmul Muller-Preparata - pseudocod

- *Notații*:
  - $A[0..n-1]$ și $P[0..n-1]$ sunt două tablouri, fiecare de dimensiune n.
  - $R[0..2n-2, 0..n-1]$ este un tablou bidimensional de mărime $(2n-1)xn$; $R[j]$ desemnează coloana j.
- *Premise*:
  - Datele de intrare sunt memorate în tabloul $A[0..n-1]$.
  - Pozițiile relative vor fi memorate în tabloul R, în liniile $n-1, n, \ldots, 2n-2$.
  - Pozițiile finale vor fi reținute în tabloul P.

```
SORTARE_PARALELA_MULLER_PREPARATA(A, R, n)
 1  for all i,j : 0 ≤ i,j ≤ n−1
 2  do in parallel/* calcularea pozițiilor relative */
 3      if A[i] < A[j]
 4         then R[i+n−1, j] ← 1
 5         else R[i+n−1, j] ← 0
 6  for all j : 0 ≤ j ≤ n−1
 7  do in parallel/* calcularea pozițiilor finale */
 8      /* Se calculează numărul elementelor care se află în fața elementului a_j */
 9      COMPRIM_ITERATIV(R[j], +)
10      P[j] = R[0, j]
11  for all j : 0 ≤ j ≤ n−1
12  do in parallel/* plasarea pe pozițiile finale */
13      A[P[j]] = A[j]
```

Exemplu de execuție a algoritmului Muller-Preparata

Tabloul R

i \ j / a_j	2	6	3	8
	0	**1**	**2**	**3**
0	0	2	1	3
1	0	1	1	2
2	0	1	0	1
3	0	1	1	1
4	0	0	0	1
5	0	1	0	1
6	0	0	0	0

a_i

Figura 1 : Exemplu de execuție a algoritmului de sortare paralelă Muller-Preparata pentru secvența 2,6,3,8

Corectitudinea

Lema (1)

Pozițiile finale ale elementelor secvenței $a_0, a_1, \ldots, a_{n-1}$ respectă relația de ordine "$<$".

Demonstrație.

În urma calculării numărului elementelor care se află în fața unui element a_i (COMPRIM_ITERATIV (R[j],+)), se obține poziția finală a acestuia, în concordanta cu relația de ordine "$<$". □

Teorema (1)

Algoritmul Muller-Preparata sortează corect o secvență de elemente $a_0, a_1, \ldots, a_{n-1}$ dintr-o mulțime U peste care este definită o relație de ordine totală "$<$".

Demonstrație.

Consecință imediată a lemei 1. □

Complexitatea

Teorema (2)

Complexitatea timp a algoritmului de sortare paralelă Muller-Preparata, implementat pe o masina CREW-PRAM cu $O(\frac{n^2}{\log n})$ unități de procesare, este $O(\log n)$.

Demonstrație.

1. Determinarea pozițiilor relative pentru fiecare pereche $\{a_i, a_j\}, i,j = 0,1,\ldots,n-1$: dacă mașina CREW-PRAM este compusă din n^2 unități de procesare, timpul paralel este $O(1)$; dacă numărul unităților de procesare este $\lceil \frac{n^2}{\log n} \rceil$, timpul paralel este $O(\log n)$ (tehnica este aceeași cu cea de la comprimare).
2. Calcularea pozitiilor finale ale elementelor a_i, $i = 0,1,\ldots,n-1$: pentru fiecare i, sunt necesare cel puțin $\lceil \frac{n}{\log n} \rceil$ unități de procesare, pentru a calcula poziția finală a elementului a_i în timpul paralel $O(\log n)$ (vezi complexitatea algoritmului paralel de comprimare). Rezultă un necesar de $n\lceil \frac{n}{\log n} \rceil$ unități de procesare pentru a calcula toate pozițiile finale în timpul paralel $O(\log n)$.
3. Plasarea elementelor a_j pe pozitiile corecte: cu n unități de procesare pentru se obține timpul paralel $O(1)$.




Comentarii

- Relativ la algoritmul secvențial, bazat pe metoda enumerării, care necesita $O(n^2)$ timp, eficiența algoritmului este $E = \frac{O(n^2)}{O(\frac{n^2}{\log n})\log n} = O(1)$.
- Totusi, algoritmul nu este optimal, deoarece cel mai rapid algoritm secvential are timpul de executie de $O(n \log n)$.

Algoritmul Impar-Par (Odd-Even Sort) - descriere

- Este o versiune a algoritmului Bubble.
- Se desfășoară în faze.
  - În fazele impare sunt sortate perechile $\{a_i, a_{i+1}\}$ cu i par.
  - În fazele pare sunt sortate perechile $\{a_i, a_{i+1}\}$ cu i impar.
- Este paralelizabil.

Algoritmul Impar-Par secvențial - pseudocod

- *Notații*: $A[0..n-1]$ este un tablouri de dimensiune n.
- *Premise*: Datele de intrare sunt memorate în tabloul $A[0..n-1]$.

SORTARE_SECVENTIALA_IMPAR_PAR(A, n)

```
1  for faza ← 1 to n
2  do if faza este impara
3        then for i ← 0 to 2⌊n/2⌋ − 2 step 2
4                do COMPARA_SI_INTERSCHIMBA(i, i + 1)
5     if faza este para
6        then for i ← 1 to 2⌊(n−1)/2⌋ − 1 step 2
7                do COMPARA_SI_INTERSCHIMBA(i, i + 1)
```

COMPARA_SI_INTERSCHIMBA(i, j)

```
1  if a_i > a_j
2     then temp ← a_i
3          a_i ← a_j
4          a_j ← temp
5
```

Exemplu de execuție a algoritmului Impar-Par

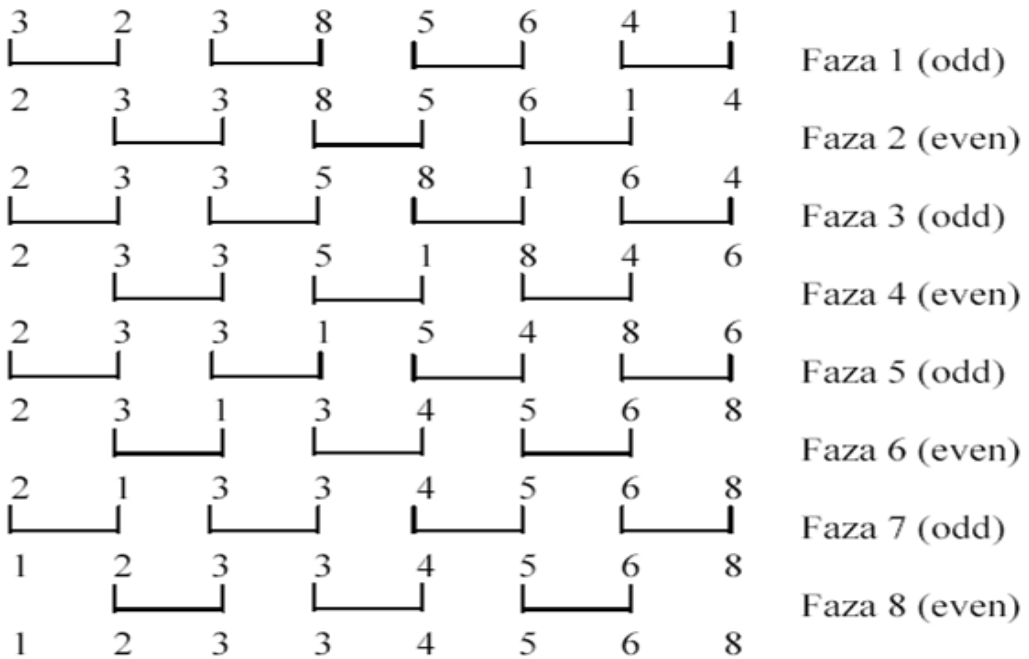


Figura 2 : Exemplu de execuție a algoritmului de sortare Impar-Par pentru $n = 8$

Algoritmul Impar-Par paralel - pseudocod pentru un lanț de unități de procesare

- *Premise*: Inițial, o unitate de procesare p_i memorează elementul a_i în registrul r.

SORTARE_PARALELA_IMPAR_PAR(p_i, r, n)

```
1  for faza ← 1 to n
2  do if faza este impara si 0 ≤ i ≤ 2⌊n/2⌋ − 1
3        then if i este par
4                then trimite lui p_{i+1} valoarea memorata in registrul r
5                     primeste de la p_{i+1} o valoare v
6                     r ← min(r, v)
7                else trimite lui p_{i−1} valoarea memorata in registrul r
8                     primeste de la p_{i−1} o valoare v
9                     r ← max(r, v)
10    if faza este para si 1 ≤ i ≤ 2⌊(n−1)/2⌋
11       then if i este impar
12               then trimite lui p_{i+1} valoarea memorata in registrul r
13                    primeste de la p_{i+1} o valoare v
14                    r ← min(r, v)
15               else trimite lui p_{i−1} valoarea memorata in registrul r
16                    primeste de la p_{i−1} o valoare v
17                    r ← max(r, v)
```

Complexitatea

Teorema (3)

Complexitatea timp a algoritmului de sortare paralelă Impar-Par, implementat pe un lanț de n unități de procesare, este $O(n)$.

Demonstrație.

Timpul paralel pentru fiecare fază este $O(1)$. După n faze algoritmul se termină. □

Comentarii

- Algoritmul de sortare paralelă *Impar-Par* este optimal pentru architecturã: Fiecare unitate de procesare este solicitată $O(n)$ timp
- Costul nu este optimal: (Numarul de unități de procesare) x (timpul paralel) = nx$n = O(n^2)$. Timpul pentru cel mai rapid algoritm secvențial este $O(n \log n)$.
- Algoritmul poate fi implementat și pe o mașină CREW-PRAM.
  Exercițiu: Scrieți un pseudocod pentru sortarea Impar-Par pe o astfel de mașină.

Algoritmul lui Batcher de sortare bitonică - descriere

- Autor: Batcher; Anul publicării: 1968.
- Operația de bază este sortarea unei secvențe bitone.
- Esența problemei sortării unei secvențe bitone este transformarea sortării unei secvențe de bitone lungime n în sortarea a doua secvențe bitone de dimensiune $\frac{n}{2}$.
- Pentru a sorta o secvență de n elemente, prin tehnica sortării unei secvențe bitone, trebuie să dispunem de o secvență bitonă formată din n elemente.
- Observații:
  - Două elemente formează o secvență bitonă.
  - Orice secvență nesortată este o concatenare de secvențe bitone de lungime 2.
- Ideea transformării unei secvențe oarecare în una bitonă: combinarea a două secvențe bitone de lungime $\frac{n}{2}$ pentru a obține o secvență bitonă de lungime n.
- Algoritmul este paralelizabil.

Secvențe bitone

- Secvența bitonă este o secvență de elemente $[a_0, a_1, \ldots, a_{n-1}]$ pentru care
  - există i astfel încât $[a_0, a_1, \ldots, a_i]$ este monoton crescătoare și $[a_{i+1}, \ldots, a_{n-1}]$ este monoton descrescătoare sau
  - există o permutare circulară astfel încât să fie satisfacută condiția anterioară.
- Exemple:
  - [1, 2, 4, 7, 6, 0]; întâi crește și apoi descrește; i = 3.
  - [8, 9, 2, 1, 0, 4]: după o permutare circulară la stânga cu 4 poziții rezultă [0, 4, 8, 9, 2, 1]; i = 3.
- Fie $S = [a_0, a_1, \ldots, a_{n-1}]$ o secventă bitonă,
  - $S_1 = [\min\{a_0, a_{\frac{n}{2}}\}, \min\{a_1, a_{\frac{n}{2}+1}\}, \ldots \min\{a_{\frac{n}{2}-1}, a_{n-1}\}]$ și
  - $S_2 = [\max\{a_0, a_{\frac{n}{2}}\}, \max\{a_1, a_{\frac{n}{2}+1}\}, \ldots \max\{a_{\frac{n}{2}-1}, a_{n-1}\}]$
- Secvențele S_1 si S_2 au proprietățile următoare:
  - Sunt bitone.
  - Fiecare element din S_1 este mai mic decit fiecare element din S_2.

Algoritmul lui Batcher de sortare bitonică - pseudocod

- *Notații*:
  - $A[0..n-1]$ este un tablou unidimensional de dimensiune n.
  - (A,i,d) definește segmentul $A[i..i+d-1] = A[i],...,A[i+d-1]$.
  - s este un parametru binar care specifică ordinea crescătoare ($s=0$) sau descrescătoare ($s=1$) a cheilor de sortare.
  - COMPARA_SI_SCHIMBA(x,y,s) desemnează sortarea a două elemente x si y ordinea indicată de parametrul s.
- *Premise*: Inițial, $A[0..n-1]$ conține secvența de sortat.
- *Apel*: SORTARE_BATCHER$(A,0,n,0)$ sau SORTARE_BATCHER$(A,0,n,1)$.

```
SORTARE_BATCHER(A, i, d, s)
1  if d = 2
2     then (A[i], A[i+1]) ← COMPARA_SI_SCHIMBA(A[i], A[i+1], s)
3     else /* sortare crescătoare a unei secvențe S de lungime d/2 */
4          SORTARE_BATCHER(A, i, d/2, 0)
5          /* sortare descrescătoare a secvenței S', care urmează lui S, de lungime d/2 */
6          SORTARE_BATCHER(A, i + d/2, d/2, 1)
7          /* sortarea secvenței bitone SS', de lungime d */
8          SORTARE_SECVENTA_BITONA(A, i, d, s)
```

Sortarea unei secvențe bitone - pseudocod

- *Premise*: Inițial, segmentul $A[i..i+d-1]$ conține o secvență bitonă de lungime d;

```
SORTARE_SECVENTA_BITONA(A, i, d, s)
 1  if d = 2
 2     then (A[i], A[i+1]) ← COMPARA_SI_SCHIMBA(A[i], A[i+1], s)
 3     else /* Construirea secvențelor bitone S1 și S2, de lungime d/2 */
 4          for all j : 0 ≤ j < d/2
 5          do in parallel
 6             (A[i+j], A[i+j+d/2]) ← COMPARA_SI_SCHIMBA(A[i+j], A[i+j+d/2], s)
 7          /* sortarea secvenței bitone S1, de lungime d/2 */
 8          SORTARE_SECVENTA_BITONA(A, i, d/2, s)
 9          /* sortarea secvenței bitone S2, de lungime d/2 */
10          SORTARE_SECVENTA_BITONA(A, i+d/2, d/2, s)
```

Exemplu de sortare a unei secvențe bitone

Original sequence	3	5	8	9	10	12	14	20	95	90	60	40	35	23	18	0
1st Split	3	5	8	9	10	12	14	0	95	90	60	40	35	23	18	20
2nd Split	3	5	8	0	10	12	14	9	35	23	18	20	95	90	60	40
3rd Split	3	0	8	5	10	9	14	12	18	20	35	23	60	40	95	90
4th Split	0	3	5	8	9	10	12	14	18	20	23	35	40	60	90	95

Figura 3 : Exemplu de sortare a unei secvențe bitone

Corectitudinea

Lema (2)

Dacă algoritmul lui Batcher sortează orice secvență de chei de sortare binare, atunci sortează orice secvență de chei de sortare numere reale oarecare.

Demonstrație.

Fie $f : \mathrm{R} \to \mathrm{R}$ o funcție monotonă. Astfel, $f(a_i) \leq f(a_j)$ dacă și numai dacă $a_i \leq a_j$. Evident, dacă algoritmul lui Batcher transformă secvența $[a_1, a_2, \ldots, a_n]$ în secvența $[b_1, b_2, \ldots, b_n]$, atunci va transforma secvența $[f(a_1), f(a_2), \ldots, f(a_n)]$ în secvența $[f(b_1), f(b_2), \ldots, f(b_n)]$. Astfel, dacă în secvența $[b_1, b_2, \ldots, b_n]$ există un indice i pentru care $b_i > b_{i+1}$, atunci în secvența $[f(b_1), f(b_2), \ldots, f(b_n)]$ vom avea $f(b_i) > f(b_{i+1})$.

Fie acum f o funcție monotonă definită astfel: $f(b_j) = \begin{cases} 0 & ,\text{dacă } b_j < b_i \\ 1 & ,\text{dacă } b_j \geq b_i \end{cases}$

În aceste condiții, secvența $[f(b_1), f(b_2), \ldots, f(b_n)]$ va fi o secventă binară nesortată deoarece $f(b_i) = 1$ și $f(b_{i+1}) = 0$. Rezultă că algoritmul lui Batcher eșuează în sortarea secvenței binare $[f(a_1), f(a_2), \ldots, f(a_n)]$. Deci, dacă algoritmul lui Batcher eșuează în sortarea unei secvențe de chei de sortare numere reale oarecare, atunci există o secvența binară care nu va fi sortată în urma aplicăriii algoritmului lui Batcher. □

Corectitudinea -continuare

Teorema (4)

Algoritmul lui Batcher sorteaza cele n elemente ale secvenței memorate în tabloul $A[0..n-1]$, *în ordinea crescătoare* $(s=0)$ *respectiv descrescătoare* $(s=1)$ *a cheilor de sortare.*

Demonstrație.

Este suficient să demonstrăm corectitudinea procedurii SORTARE_SECVENTA_BITONA pentru cazul binar (Lema 2). Procedăm prin inducție după lungimea d a secvențelor procesate.

Dacă $d=2$, evident procedura SORTARE_SECVENTA_BITONA transformă secvența inițială $S_{init}=A[i..i+d-1]$ într-o secvență sortată S_{fin}.

Vom demonstra că procedura SORTARE_SECVENTA_BITONA transformă o secvență binară S_{init} de tipul $0^r1^t0^v$ sau $1^r0^t1^v (r+t+v=d)$ într-o secvență S_{fin} sortată în ordinea indicată de valoarea lui $s, (\forall) d \geq 2$

Pasul paralel 6 transformă secvența S_{init} într-o secvență S_{temp} conform figurilor 2 și 3.

Se observă că în toate cazurile, secvența rezultată S_{temp}, este formată din două sub-secvențe bitone S^1_{temp} și S^2_{temp}, fiecare de lungime $\frac{d}{2}$. O secvență este de tipul S_{init} iar cealaltă secvență conține numai cifre 0 sau numai cifre 1. Dacă $s=0$, cheia maximă din S^1_{temp} este mai mică sau egală cu cheia minimă din S^2_{temp}. Dacă $s=1$, cheia minimă din S^1_{temp} este mai mare sau egală cu cheia maximă din S^2_{temp}.

Conform ipotezei de inducție, procedura SORTARE_SECVENTA_BITONA transformă secvențele S^1_{temp} și S^2_{temp} în două secvențe S^1_{fin} și S^2_{fin}, sortate crescător $(d=0)$ sau descrescător $(s=1)$. Dacă $s=0$, cheia maximă din S^1_{fin} este mai mică sau egală cu cheia minimă din S^2_{fin} deci secvența $S^1_{fin}S^2_{fin}$ este crescătoare. Daca $s=1$, cheia minimă din S^1_{fin} este mai mare sau egală cu cheia maximă din S^2_{fin} deci secvența $S^1_{fin}S^2_{fin}$ este descrescătoare. □

Corectitudinea -continuare

s	$p+q(p,q \leq \frac{d}{2})$	Secvența inițială	Secvența rezultată
0	$\leq \frac{d}{2}$	$0^{\frac{d}{2}-p}1^p1^q0^{\frac{d}{2}-q}$	$0^{\frac{d}{2}}1^q0^{\frac{d}{2}-(p+q)}1^p$
0	$\leq \frac{d}{2}$	$0^{\frac{d}{2}}0^k1^{p+q}0^m$	$0^{\frac{d}{2}}0^k1^{p+q}0^m$
0	$\leq \frac{d}{2}$	$0^k1^{p+q}0^m0^{\frac{d}{2}}$	$0^{\frac{d}{2}}0^k1^{p+q}0^m$
0	$\leq \frac{d}{2}$	$1^p0^{\frac{d}{2}-p}0^{\frac{d}{2}-q}1^q$	$0^{\frac{d}{2}}1^p0^{\frac{d}{2}-(p+q)}1^q$
0	$\leq \frac{d}{2}$	$1^{\frac{d}{2}}1^p0^{\frac{d}{2}-(p+q)}1^q$	$1^p0^{\frac{d}{2}-(p+q)}1^q1^{\frac{d}{2}}$
0	$\leq \frac{d}{2}$	$1^p0^{\frac{d}{2}-(p+q)}1^q1^{\frac{d}{2}}$	$1^p0^{\frac{d}{2}-(p+q)}1^q1^{\frac{d}{2}}$
0	$> \frac{d}{2}$	$0^{\frac{d}{2}-p}1^p1^q0^{\frac{d}{2}-q}$	$0^{\frac{d}{2}-p}1^{(p+q)-\frac{d}{2}}0^{\frac{d}{2}-q}1^{\frac{d}{2}}$
0	$> \frac{d}{2}$	$1^p0^{\frac{d}{2}-p}0^{\frac{d}{2}-q}1^q$	$0^{\frac{d}{2}-q}1^{(p+q)-\frac{d}{2}}0^{\frac{d}{2}-p}1^{\frac{d}{2}}$
1	$\leq \frac{d}{2}$	$0^{\frac{d}{2}-p}1^p1^q0^{\frac{d}{2}-q}$	$1^q0^{\frac{d}{2}-(p+q)}1^p0^{\frac{d}{2}}$
1	$\leq \frac{d}{2}$	$0^{\frac{d}{2}}0^k1^{p+q}0^m$	$0^k1^{p+q}0^m0^{\frac{d}{2}}$
1	$\leq \frac{d}{2}$	$0^k1^{p+q}0^m0^{\frac{d}{2}}$	$0^k1^{p+q}0^m0^{\frac{d}{2}}$
1	$\leq \frac{d}{2}$	$1^p0^{\frac{d}{2}-p}0^{\frac{d}{2}-q}1^q$	$1^p0^{\frac{d}{2}-(p+q)}1^q0^{\frac{d}{2}}$
1	$\leq \frac{d}{2}$	$1^{\frac{d}{2}}1^p0^{\frac{d}{2}-(p+q)}1^q$	$1^{\frac{d}{2}}1^p0^{\frac{d}{2}-(p+q)}1^q$
1	$\leq \frac{d}{2}$	$1^p0^{\frac{d}{2}-(p+q)}1^q1^{\frac{d}{2}}$	$1^{\frac{d}{2}}1^p0^{\frac{d}{2}-(p+q)}1^q$
1	$> \frac{d}{2}$	$0^{\frac{d}{2}-p}1^p1^q0^{\frac{d}{2}-q}$	$1^{\frac{d}{2}}0^{\frac{d}{2}-p}1^{(p+q)-\frac{d}{2}}0^{\frac{d}{2}-q}$
1	$> \frac{d}{2}$	$1^p0^{\frac{d}{2}-p}0^{\frac{d}{2}-q}1^q$	$1^{\frac{d}{2}}0^{\frac{d}{2}-q}1^{(p+q)-\frac{d}{2}}0^{\frac{d}{2}-p}$

Figura 4 : Corectitudinea

Corectitudinea -continuare

s=0

0000111111	1110000000	0000000000	1110111111
0000000000	0011111000	0000000000	0011111000
0011111000	0000000000	0000000000	0011111000
1111110000	0000000011	0000000000	1111110011
1111111111	1100000111	1100000111	1111111111
1100000111	1111111111	1100000111	1111111111
0011111111	1111000000	0011000000	1111111111
1111111000	0000001111	0000001000	1111111111

s=1

0000111111	1110000000	1110111111	0000000000
0000000000	0011111100	0011111100	0000000000
0011111100	0000000000	0011111100	0000000000
1111110000	0000000111	1111110111	0000000000
1111111111	1111000111	1111111111	1111000111
1111000111	1111111111	1111111111	1111000111
0111111111	1110000000	1111111111	0110000000
1111100000	0011111111	1111111111	0011100000

Figura 5 : Corectitudinea

Implementarea algoritmului lui Batcher pe rețele de sortare

- Dacă se derecursivează algoritmul lui Batcher, se constată că sortarea unei secvențe de $n = 2^m$ elemente constă în m faze de sortare a unei secvențe bitone: $SSB_0, SSB_1, \ldots, SSB_{m-1}$
- În faza SSB_k, $k \in \{0, 1, \ldots, m-1\}$, se realizează sortarea secvențelor bitone S_i^{2d} formate din perechile de secvențe consecutive $S_i^d S_i'^d$ de lungime $d = 2^k$, S_i^d fiind sortată crescător și $S_i'^d$ descrescator.
- Secvențele S_i^{2d} cu numărul de ordine i par sunt sortate crescător. Cele cu i impar sunt sortate descrescător.

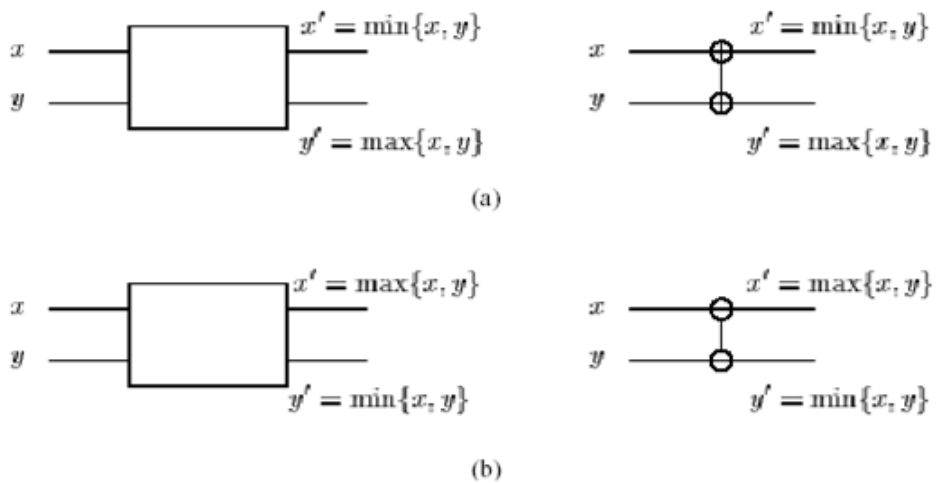


Figura 6 : Comparatori pentru sortarea a două elemente

Rețea de comparatori care transformă o secvență oarecare în una bitonă
Fazele $0, 1, \ldots, m-2$

- Intrare: o secvență oarecare; Ieșire: o secvență bitonă.

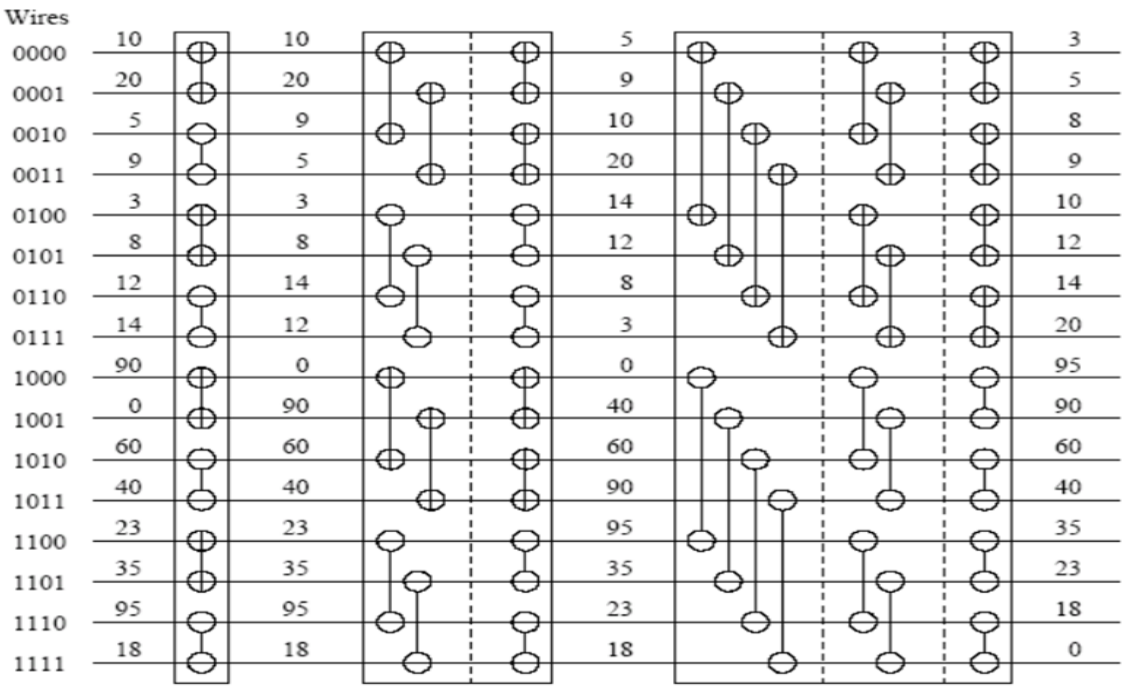


Figura 7 : Rețea de comparatori care transformă o secvență oarecare în una bitonă (R_1); $n = 16$

Rețea de sortare a unei secvențe bitone - Faza $m-1$

- Intrare: o secvență bitonă; Ieșire: o secvență sortată.

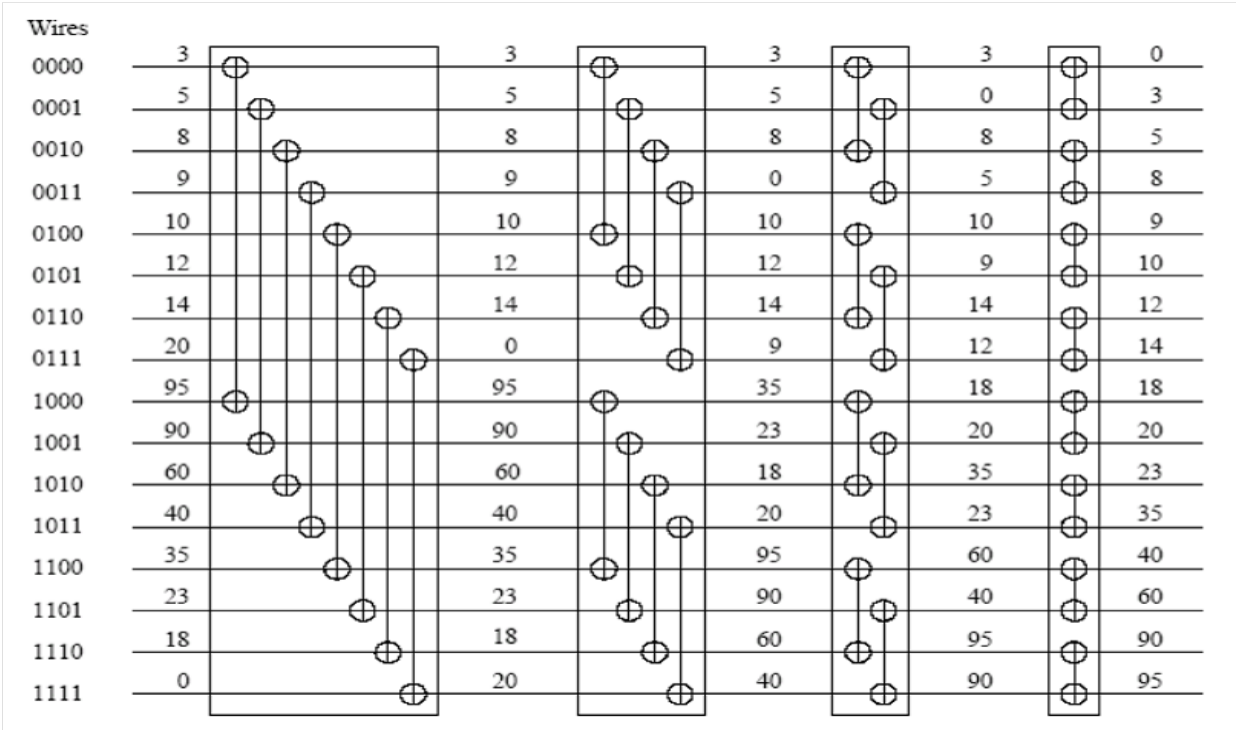


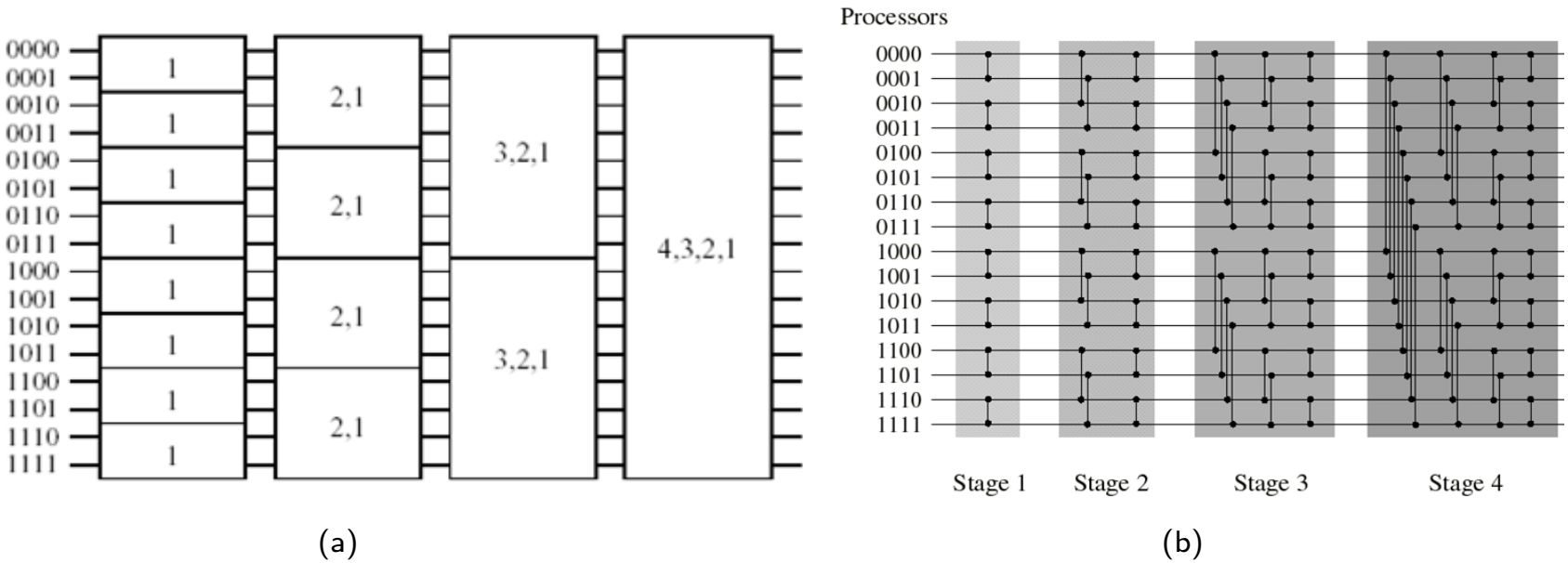
Figura 8 : Rețea de sortare a unei secvențe bitone (R_2); $n = 16$

Implementarea algoritmului lui Batcher pe hipercub

- Cubul binar multidimensional este o arhitectură ideală pentru implementarea algoritmului lui Batcher.
- Reamintim:
  - Dacă se derecursivează algoritmul lui Batcher, se constată că sortarea unei secvențe de $n = 2^m$ elemente constă în m faze de sortare a unei secvențe bitone: $SSB_0, SSB_1, \ldots, SSB_{m-1}$
  - În faza SSB_k, $k \in \{0, 1, \ldots, m-1\}$, se realizează sortarea secvențelor bitone S_i^{2d} formate din perechile de secvențe consecutive $S_i^d S_i^{\prime d}$ de lungime $d = 2^k$, S_i^d fiind sortată crescător și $S_i^{\prime d}$ descrescator.
  - Secvențele S_i^{2d} cu numărul de ordine i par sunt sortate crescător. Cele cu i impar sunt sortate descrescător.
- Execuția fazei SSB_k pe hipercub necesită utilizarea succesivă a dimensiunilor $D_k, D_{k-1}, \ldots, D_1, D_0$.
- Planificarea utilizării dimensiunilor pentru o sortare completă poate fi reprezentată astfel:
  - SSB_0 : D_0
  - SSB_1 : D_1, D_0
  - SSB_2 : D_2, D_1, D_0

    ...
  - SSB_{m-1} : $D_{m-1}, D_{m-2}, \ldots, D_0$

Planificarea dimensiunilor = Fazele sortării prin rețele de sortare Batcher



(a) (b)

Figura 9 : (a) Planificarea dimensiunilor (b)Fazele sortării prin rețele de sortare Batcher

Comunicarea pe hipercub în timpul ultimei faze a algoritmului lui Batcher

Fiecare linie continuă reprezintă o operație de interschimbare.

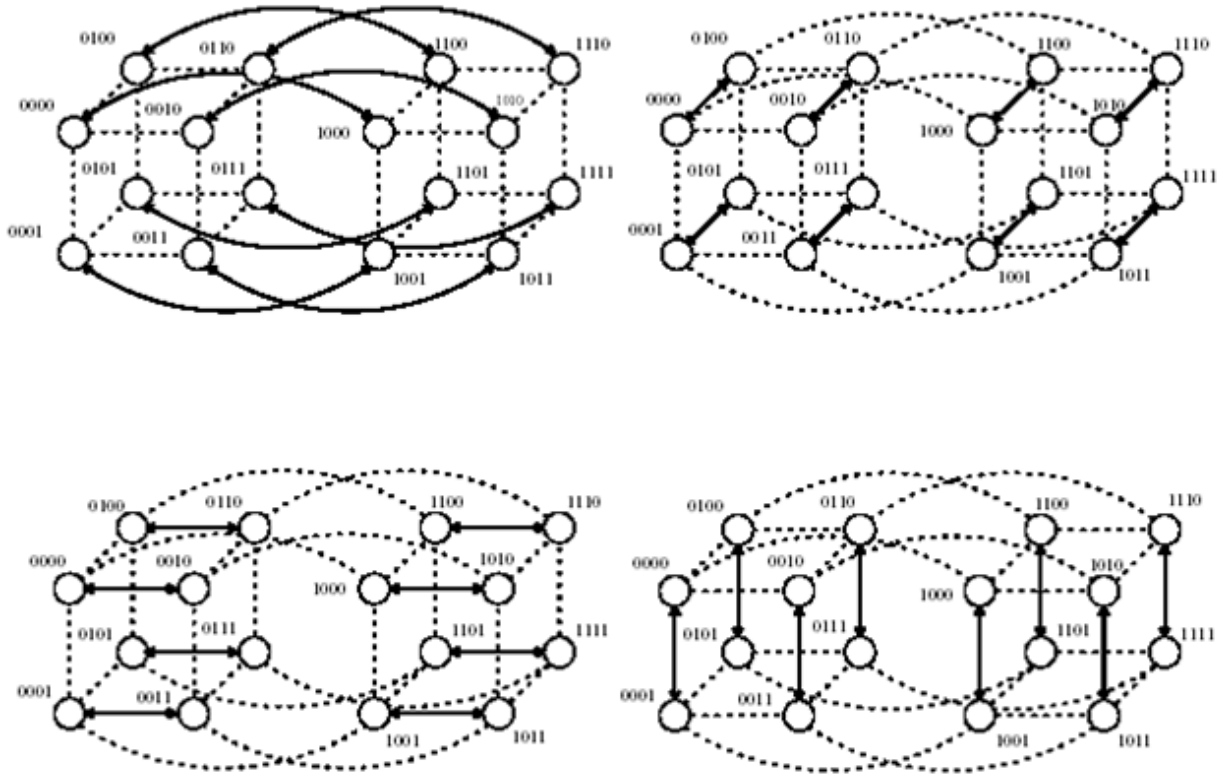


Figura 10 : Comunicarea în timpul ultimei faze a algoritmului lui Batcher

Complexitatea implementării pe o mașină CREW-PRAM

Teorema (4)

Complexitatea timp a algoritmului lui Batcher de sortare paralelă, implementat pe mașină CREW-PRAM cu $O(n)$ unități de procesare este $O(\log^2 n)$. Eficiența algoritmului este $O(\frac{1}{\log n})$.

Demonstrație.

Timpul paralel pentru sortarea unei secvențe bitone de lungime $d < n$ este $O(\log d)$. După $\log n$ faze algoritmul se termină. Eficiența este $\frac{O(n \log n)}{nO(\log^2 n)} = O(\frac{1}{\log n})$. □




Complexitatea implementării pe rețele de sortare

Teorema (5)

Complexitatea timp a algoritmului lui Batcher de sortare paralelă, implementat pe o rețea de sortare cu $O(n)$ intrări si ieșiri, este $O(\log^2 n)$.

Demonstrație.

Rețeaua (R_1, R_2) implementează algoritmul lui Batcher. Numărul de faze este $\log n$. Fazele sunt compuse din $\log d < \log n$ pași □

Complexitatea implementării pe hipercub

Teorema (6)

Complexitatea timp a algoritmului lui Batcher de sortare paralelă, implementat pe un hipercub cu $O(n)$ unități de procesare este $O(\log^2 n)$.

Demonstrație.

Sortarea Batcher pe hipercub este congruentă cu sortarea Batcher pe rețele de sortare □

Comentarii

- Algoritmul lui Batcher de sortare paralelă este din clasa timp $O(\log^2 n)$. Numărul de unități de procesare este din clasa $O(n)$. Comparativ cu algoritmii Muller-Preparata și Impar-Par are un cost mai bun.
- Totuși, costul nu este optimal: (Numarul de unități de procesare) x (timpul paralel) $= n\log^2 n = O(n\log^2 n)$. Timpul pentru cel mai rapid algoritm secvențial este $O(n\log n)$.




Sortare rapidă pe hipercub - descriere

- Să ne amintim că un hipercub cu m dimensiuni este format din două hipercuburi cu $m-1$ dimensiuni.
- Numărul unităților de procesare, p, este mai mic decât numărul elementelor secvenței de sortat, n.
- Ideea este de a partiționa secvența de sortat pe subcuburi și apoi de a repeta repeta recursiv această operație.
- Selectarea pivotului este problema cheie.

Sortare rapidă pe hipercub - pseudocod

- *Notații*:
  - $A[0..n-1]$ este un tablou de dimensiune $n = 2^m$.
- *Premise*:
  - Datele de intrare sunt partiționate între unitățile de procesare.
  - Fiecare unitate de proceare(nod al hipercubului) memorează o parte A_i din secventa de sortat, memorată inițial în tabloul $A[0..n-1]$.

SORTARE_ RAPIDA_ PE_HIPERCUB($A_i, m, p_i, pivot$)

```
1   i ← eticheta unitatii de procesare
2   for k ← m − 1 downto 0
3   do /*   */
4       x ← pivot
5       partitioneaza A_i in A_{i_1} si A_{i_2} astfel incat A_{i_1} ≤ x ≤ A_{i_2}
6       if al − k − lea bit este 0
7         then trimite A_{i_2} unitatii de procesare vecina pe dimensiunea k
8              B_{i_1} ← subsecventa primita de la vecinul de pe dimensiunea k
9              A_i ← A_{i_1} ⋃ B_{i_1}
10        else trimite A_{i_1} unitatii de procesare vecina pe dimensiunea k
11             B_{i_2} ← subsecventa primita de la vecinul de pe dimensiunea k
12             A_i ← A_{i_2} ⋃ B_{i_2}
13  Aplica lui A_i sortarea rapida secventiala
```

Exemplu de execuție a algoritmului de sortare rapidă pe hipercub (m=3)

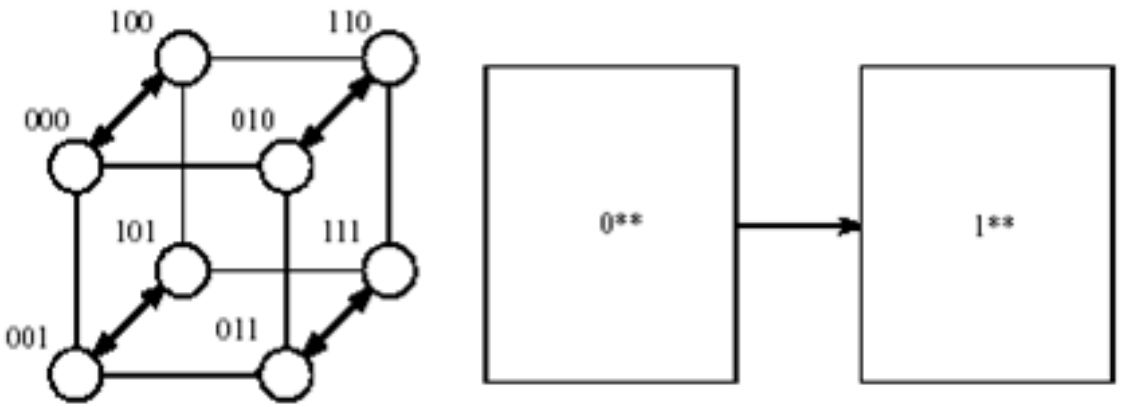


Figura 11 : Partiționarea secvenței de sortat în două blocuri. Se utilizează dimensiunea a treia ($m-1$).

Exemplu de execuție a algoritmului de sortare rapidă pe hipercub - continuare

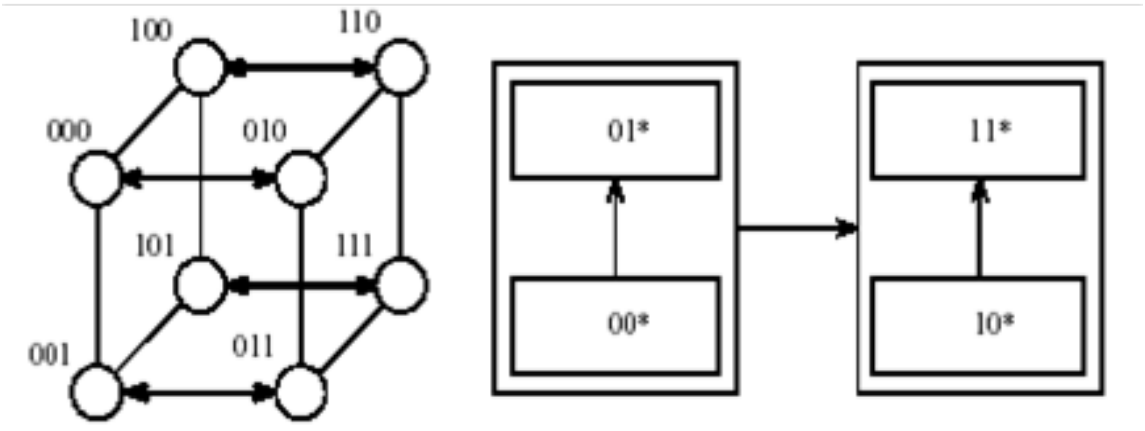


Figura 12 : Partiționarea fiecărui bloc în două sub-blocuri. Se utilizează dimensiunea a doua ($m-2=1$).

Exemplu de execuție a algoritmului de sortare rapidă pe hipercub - continuare

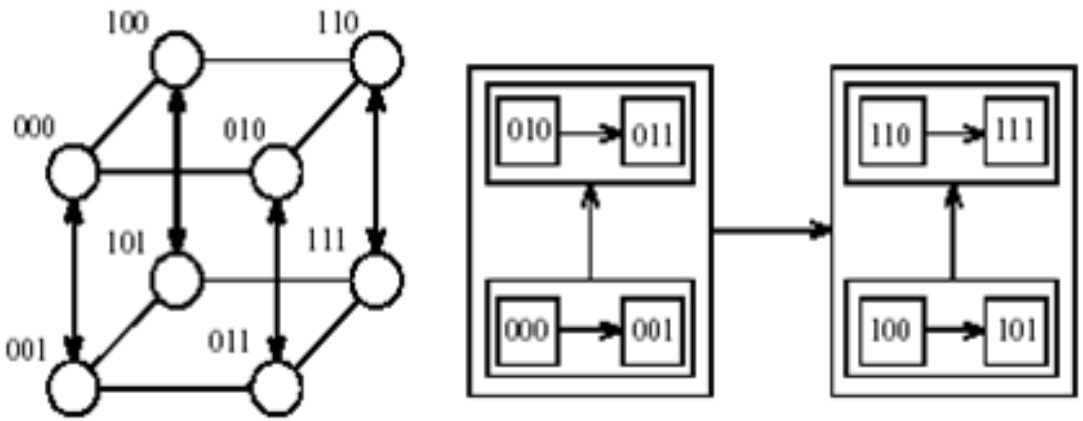


Figura 13 : Partiționarea sub-blocurilor. Se utilizează prima dimensiune ($m-3=0$).

Complexitatea

Teorema (7)

Dacă pivotul este ales astfel încât să partiționeze secvența în două subsecvențe de dimensiuni aproximativ egale, atunci

$$T_p = O(\frac{n}{p}\log\frac{n}{p}) + O(\frac{n}{p}\log p) + O(\log^2 p)$$

.




Comentarii

- Selectarea unui pivot care să partiționeze secvența în două subsecvențe de dimensiuni aproximativ egale este dificilă.




Comentarii bibliografice

- Capitolul sortare are la bază cartea
  V. Kumar, A. Grama A. Gupta & G Karypis, Introduction to Parallel Computing: Design and Analysis of Algorithms, Addison Wesley, 2003
  și ediția mai veche
  V. Kumar, A. Grama A. Gupta & G Karypis, Introduction to Parallel Computing: Design and Analysis of Algorithms, Benjamin-Cummings, 1994