2019

Programmazione Concorrente, Parallela e su Cloud

Algoritmi per memoria condivisa

Carmine Spagnuolo, Ph.D.

Finding the Minimum in $O(logn)$ Time

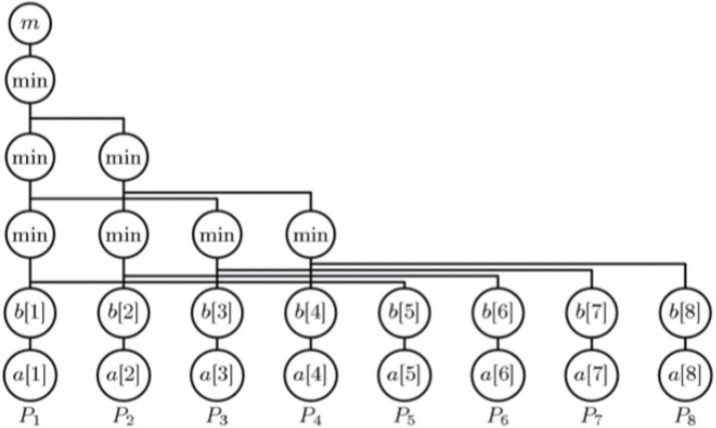
- Algoritmo sequenziale, $T(n) = n - 1 = O(n)$:

```
m := a[1];
for i := 2 to n do
    if a[i] < m then
        m := a[1];
    end if
end for
```

Finding the Minimum in $O(logn)$ Time

- Esecuzione sincrona, in ogni step di computazione i processori eseguono le stesse operazioni su dati differenti.
- Utilizziamo il costrutto **parfor**.
- Identico al costrutto for, ma che permette di eseguire le operazioni del corpo del for su un certo insieme di processori identificati da un indice progressivo $i = 1, 2, \ldots p = n$.
- Le future operazioni logaritmiche sono in base 2.

Finding the Minimum in $O(logn)$ Time – Idea $n = 8 = 2^3$



Finding the Minimum in $O(logn)$ Time

```
parfor P_i, 1 ⩽ i ⩽ n do
    b[i] := a[i];
    k := n;
end parfor
for j := 1 to logn do
    parfor P_i, 1 ⩽ i ⩽ K/2 do
        if b[i] > b[i+k/2] then
            b[i]:=b[i+k/2];
        end if
        k := K/2;
    end parfor
end for
if i=1 then
    m := b[1];
end if
```

Finding the Minimum in $O(logn)$ Time – Metriche

- Time Complexity: $T(p, n)|_{p=n} = T(n) = 1 + logn = O(logn)$
- Speedup: $S(n) = \frac{n-1}{1+logn} = O\left(\frac{n}{logn}\right)$
- Cost: $C(n) = n(1 + logn) = O(nlogn)$
- Efficiency: $E(p, n) = \frac{n-1}{n(1+logn)} = O\left(\frac{1}{logn}\right)$

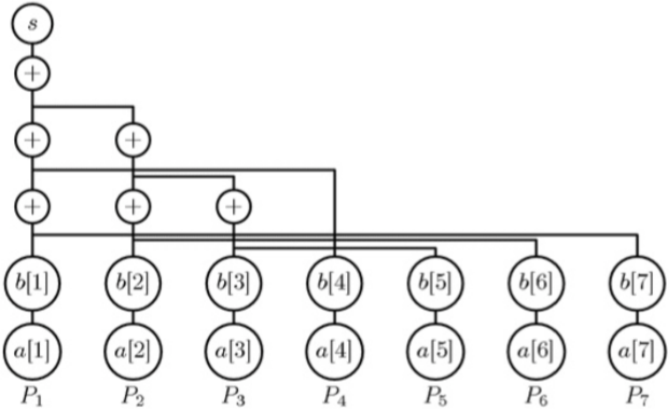
Finding the Minimum in $O(logn)$ Time – Considerazioni (1)

- Lo speedup $O\left(\frac{n}{logn}\right)$, incrementa al crescere di n.
- La versione parallela è molto più veloce della sequenziale.
- Il costo dell'algoritmo $nlogn$ è maggiore della versione sequenziale $n-1$.
- Quindi dato che l'algoritmo non è ottimale in termini di costo, al crescere di n l'efficienza dell'algoritmo tende a 0.

Finding the Minimum in $O(logn)$ Time – Considerazioni (2)

- Problema $p == n$, il numero di processori è una funzione della dimensione dell'input n.
- $p(n) = n -> $ **processor complexity of the algorithm**.
- Inoltre la dimensione dell'input n deve essere una potenza di 2.
- Questo può essere risolto!
- Come?
- Supponiamo di voler trovare la somma di n elementi in un array a di interi.

Finding the Sum in $O(logn)$ Time – $n = 7, p = 7$



Finding the Sum in $O(logn)$ Time – $n \neq 2^r, r > 0$

parfor $P_i, 1 \leqslant i \leqslant n$ **do**
    b[i] := a[1];
    k := n;
end parfor
for $j := 1$ to $\lceil logn \rceil$ **do**
    parfor $P_i, 1 \leqslant i \leqslant \lceil\rceil K/2$ **do**
        b[i]:= b[i] + b[k+1-i];
        k := $\lceil K/2 \rceil$;
    end parfor
end for
if i=1 **then**
    $s := b[1]$;
end if

Finding the Sum in $O(logn)$ Time – Considerazioni

- Efficienza tende a 0 al crescere di n.
- Ciò è dovuto al numero di processori p che deve essere uguale ad n.
- Soluzione dividere l'array a in p sotto-array, ognuno di dimensione $\lceil n/p \rceil$.

Finding the Sum in $O(logn)$ Time – $n \neq 2^r, r > 0$

parfor $P_i, 1 \leqslant i \leqslant p$ **do**
    g $= \lceil \frac{n}{p} \rceil (i-1) + 1$;
    b[i] := a[g];
    for $j = 1$ to $\lceil \frac{n}{p} \rceil - 1$ **do**
        if $g + j \leqslant n$ **then**
            b[i] = b[i] + a[g+j];
        end if
    end for
end parfor
Sum up $b[1], b[2], \ldots, b[p]$ in O($logp$) time.

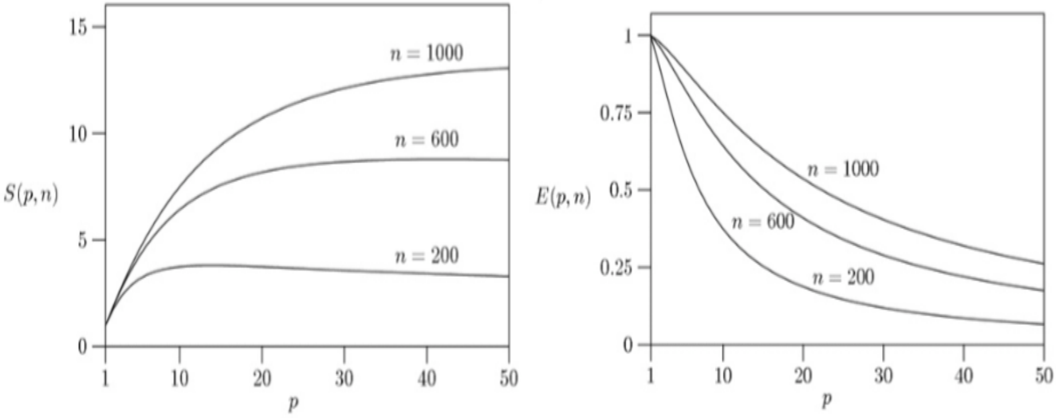
Finding the Sum in $O(logn)$ Time – Metriche

- Time Complexity: $T(p,n) = O(n/p + logp)$
- Speedup: $S(n) \cong O\left(\frac{n}{n/p+logp}\right)$
- Cost: $C(n) \cong O(n + plogp)$
- Efficiency: $E(p,n) \cong O\left(\frac{n}{n+plogp}\right)$

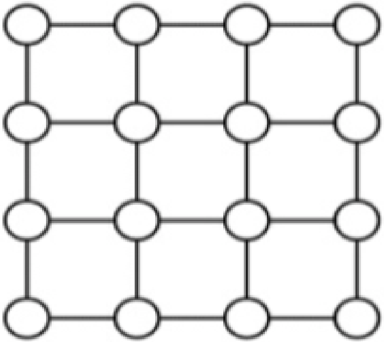
Finding the Sum in $O(logn)$ Time – Considerazioni (1) $p = n/logn$

- Time Complexity: $T(p, n) = O(logn)$
- Speedup: $S(n) \cong O\left(\frac{n}{logn}\right)$
- Cost: $C(n) \cong O(n)$
- Efficiency: $E(p, n) \cong \Theta(1)$

Finding the Sum in $O(logn)$ Time – Considerazioni (2) $p = n/logn$



Sorting in $O(log n)$ Time



- Idea counting sort algorithm.
- n^2 processori in una rete virtuale $n \times n$ a mesh.

Sorting in $O(log n)$ Time

- **Input**: $A[0, 1, \ldots, n]$
- **Output**: $B[0, 1, \ldots, n]$
- **Memoria di supporto**: $C[0, 1, \ldots, n]$
- **Algo**:
  1. Calcolo il minimo e il massimo di A, $k = max - min - 1$.
  2. Per ogni elemento $A[i], 0 \leqslant i \leqslant n-1$, incrementa $C[A[i] - min]$ di 1, ad esempio se il valore 7 appare 8 volte nell'array A, allora $C[7] = 8$.
  3. Aggiorna lo stato di C, $C[i] = C[i] + C[i-1], 1 \leqslant i \leqslant k$.
  4. Ordina A in B secondo i valori di C. $C[A[i]], 1 \leqslant i \leqslant k$, da n a 0 e decrementando di 1 il valore di posizione in $C[A[i]]$.

Sorting in $O(logn)$ Time

- Maggiori Dettagli: https://spagnuolocarmine.github.io/news/counting-sort/

Sorting in $O(logn)$ Time - Idea

1. L'$i-$esima riga della matrice w, computa la posizione dell'$i-$esimo elemento nell'array a.
2. Per ogni elemento utilizziamo la seguente regola di confronto:

$$w[i,j] = \begin{cases} 1, a[i] > a[j] || (a[i] = a[j] \& i > j) \\ 0, altrimenti \end{cases}$$

3. Contiamo il numero di 1 per ogni ogni riga i, tale somma corrisponde alla posizione dell'elemento $a[i]$ nel vettore ordinato b.

Sorting in $O(logn)$ Time

parfor $P_{i,j}, 1 \leqslant i,j \leqslant n$ **do**
    if $(a[i] > a[j]) or ((a[i]) = a[j]) and (i > j))$ **then**
        w[i,j] = 1;
    else
        w[i,j] = 0;
    end if
end parfor
k = n;
for r **do** = 1 to $\lceil logn \rceil$ {count the ones in rows of array w}
    parfor $P_{i,j}, 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant \lfloor k/2 \rfloor$ **do**
        w[i,j] = w[i,j]+w[i,k+1-j];
    end parfor
    k=$k/2$
end for
parfor $P_{i,j}, 1 \leqslant i \leqslant n, j = 1$ **do**
    b[w[i,1]+1] = a[i];
end parfor

Sorting in $O(logn)$ Time - Esempio

- $w[i,j] = \begin{cases} 1, a[i] > a[j] || (a[i] = a[j] \& i > j) \\ 0, altrimenti \end{cases}$
- $A = [3, 5, 4, 7]$
- $P = \begin{bmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & P_{34} \\ P_{41} & P_{42} & P_{43} & P_{44} \end{bmatrix}$
- $w = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix}$

Sorting in $O(logn)$ Time - Esempio

- $\lceil log(4) \rceil = 2$
- passo 1 $w = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 2 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 \end{bmatrix}$
- passo 2 $w = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 2 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}$

- $b[w[1,1]+1] = a[1] = 3$
- $b[w[2,1]+1] = a[2] = 4$
- $b[w[3,1]+1] = a[3] = 5$
- $b[w[4,1]+1] = a[4] = 7$

Sorting in $O(logn)$ Time – Metriche

- Time Complexity: $T(p,n)|_{p=n^2} = O(logn)$
- Speedup: $S(n) = \frac{nlogn}{logn} = O(n)$ alto
- Cost: $C(n) = O(n^2 logn)$
- Efficiency: $E(p,n) = \frac{nlogn}{n^2 logn} = O(1/n)$ basso utilizzo dei processori

References I

- Czech, Zbigniew J. (2017a). *Introduction to Parallel Computing*. Finding the Minimum and Sum of Elements in O(logn) Time, Section 3.5.1. Cambridge University Press.
- — (2017b). *Introduction to Parallel Computing*. Finding Minimum in O(1) Time, Section 3.5.4. Cambridge University Press.
- — (2017c). *Introduction to Parallel Computing*. Sorting in O(logn), Section 3.5.4. Cambridge University Press.