Fontes principais

1. J. Jaja, An introduction to Parallel Algorithms, Addison Wesley, 92
   - ▷ Algoritmos paralelos
2. E. Cáceres, H. Mongeli, S. Song: Algoritmos paralelos usando CGM/PVM/MPI: uma introdução
   `http://www.ime.usp.br/~song/papers/jai01.pdf`

Divisão e Conquista

Divisão e Conquista

A estratégia de divisão e conquista consiste de três passos:

1) Particionamento da entrada em partes iguais

2) Resolver recursivamente o subproblema definido para cada partição da entrada

3) Combinar as soluções de diferentes subproblemas numa solução para o problema global

O sucesso desta estratégia depende de como o terceiro passo possa ser efetuado com eficiência.

Ordenação: Algoritmo Mergesort paralelo

Ordenação: Algoritmo Mergesort paralelo

Idéia:

▷ Para ordenar uma sequência de n números, dividimos essa sequência em 2 metades, ordenamos cada uma das metades e fazemos o merge das 2 metades já ordenadas.

▷ Para ordenar cada uma das metades, aplica-se a mesma idéia: divide, ordena, merge, recursivamente, até termos sequencias de tamanho 1.

Ordenação: Algoritmo Mergesort paralelo

Entrada:

- $\triangleright$ A, B: vetores de $\frac{n}{2}$ elementos ordenados

Saída:

- $\triangleright$ C: vetor de n elementos (de A e B) ordenado

Suposição inicial: Todos os elementos de A e B são distintos

Ordenação: Algoritmo Mergesort paralelo

Estruturas auxiliares:

▷ Posição em A, posição em B, vetores com $\dfrac{n}{2}$ posições.

Posição em $A[i]$ diz em que posição o elemento $B[i]$ deveria ficar, caso fosse inserido em A, de maneira a manter a ordenação.

Posição em $B[i]$ é análogo.

Ordenação: Algoritmo Mergesort paralelo

Algoritmo Merge

para $0 \leq i \leq \frac{n}{2}$ **faça em paralelo**

$posicaoEmA[i] := buscaBinaria(B[i], A, 0, \frac{n}{2} - 1)$

$posicaoEmB[i] := buscaBinaria(A[i], B, 0, \frac{n}{2} - 1)$

$C[posicaoEmA[i] + i] := B[i]$

$C[posicaoEmB[i] + i] := A[i]$

Algoritmo Busca Binária ($num, vetor, i, f$**)**

$inicio := i,\ fim := f$

enquanto $inicio < fim$ **faça**

$meio := [(inicio + fim)/2]$

se $num < vetor[meio]$ **então**

$fim := meio - 1$

senão se $num > vetor[meio]$ **então**

$inicio := meio + 1$

se $num < vetor[meio]$ **então**

devolva $meio$

senão se $num > vetor[meio]$ **então**

devolva $meio + 1$

Ex.: $n = 8$

	0	1	2	3
A	13	14	17	19
B	11	15	16	20

posicaoEmA	0	2	2	4
posicaoEmB	1	1	3	3

Ordenação: Algoritmo Mergesort paralelo

Submodelo e complexidades:

Submodelo: CREW

Complexidades

- ▷ Tempo: $O(\log n)$
- ▷ Processador: $O(n)$

Caso A e B tenham elementos iguais

Usamos 2 rotinas de busca binária.

▷ Busca binária 1 retorna a posição em que $B[i]$ seria inserido em A, de maneira que ele seja inserido após os elementos de A iguais a ele.

▷ Busca binária 2 retorna a posição em que $A[i]$ seria inserido em B, de maneira que ele seja inserido antes do elemento de B iguais a ele.

Ordenação: Algoritmo Mergesort paralelo

O algoritmo de ordenação utiliza o algoritmo de merge com uma subrotina da forma:

Merge($A, iniA, B, iniB, C, iniC, tamC$**)**

Ordenação: Algoritmo Mergesort paralelo

Merge($A, iniA, B, iniB, C, iniC, tamC$**)**

para $0 \leq i \leq \frac{tamC}{2} - 1$ **faça em paralelo**

$posicaoEmA[iniB + i] := buscaBinaria(B[iniB + i], A, iniA, iniA + \frac{tamC}{2} - 1)$

$posicaoEmB[iniA + i] := buscaBinaria(A[iniA + i], B, iniB, iniB + \frac{tamC}{2} - 1)$

$C[posicaoEmA[iniB + i] + iniB + i] := B[iniB + i]$

$C[posicaoEmB[iniA + i] + iniA + i] := A[iniA + i]$

Ordenação: Algoritmo Mergesort paralelo

Entrada:

- ▷ S: vetor de n elementos a ser ordenado
- ▷ n: potência de 2

Saída:

- ▷ R: vetor de n elementos com os elementos de S ordenado

Estrutura auxiliar:

- ▷ T: vetor de n posições. Usado para fazer a cópia de R.

Algoritmo Mergesort

para $0 \le i \le n-1$ **faça em paralelo**

$R[i] := S[i]$

$\triangleright$ Loop sequencial, subindo na árvore

para $j := (\log n) - 1$ **até** 0 **faça**

para $0 \le i \le n-1$ **faça em paralelo**

$T[i] := R[i]$

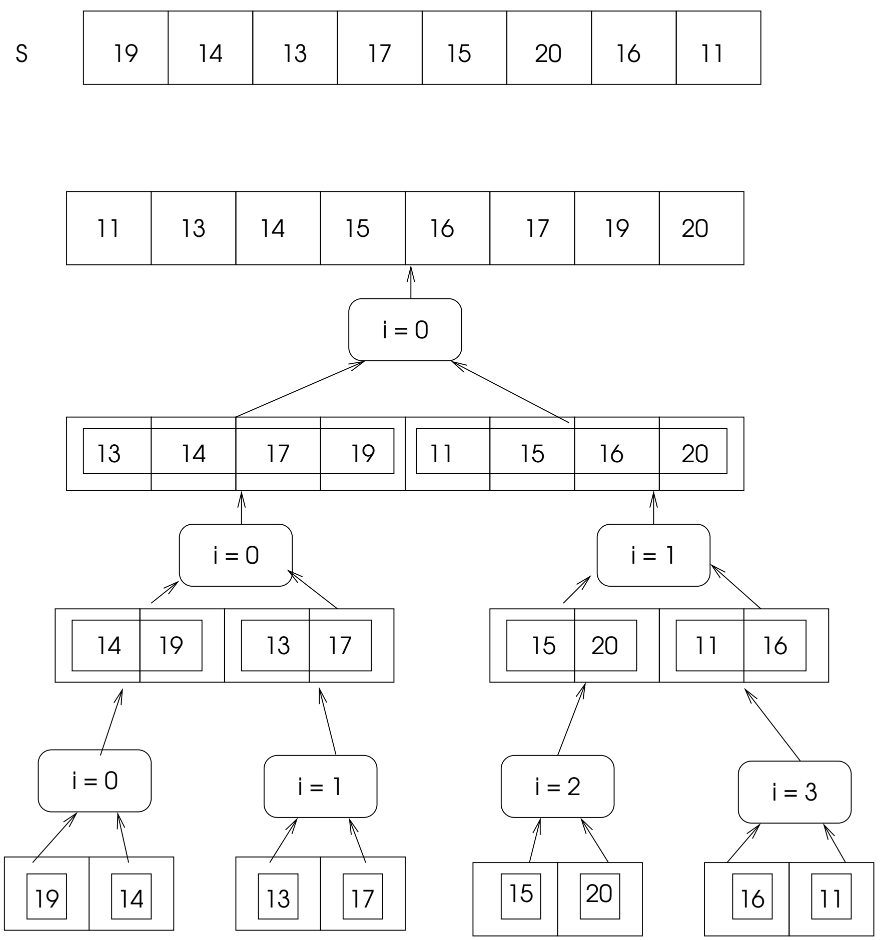
$tam := n/2^j$ $\triangleright$ Tamanho da sequência ordenada

$\triangleright$ a ser obtida neste nível

para $0 \le i \le 2^j - 1$ **faça em paralelo**

$Merge(T, i*tam, T, i*tam + \frac{tam}{2}, R, i*tam, tam)$

Ex.: $n = 8$


S
19 14 13 17 15 20 16 11
11 13 14 15 16 17 19 20
i = 0
13 14 17 19 11 15 16 20
i = 0
i = 1
14 19 13 17 15 20 11 16
i = 0
i = 1
i = 2
i = 3
19 14 13 17 15 20 16 11

Ordenação: Algoritmo Mergesort paralelo

Submodelo: CREW (leitura concorrente em T e tam)

Complexidades:

- ▷ Tempo: $O(\log^2 n)$ ($\log n$ passos do sort e $\log n$ passos do merge)
- ▷ Processadores: $O(n)$

Ordenação: Algoritmo Mergesort paralelo

No nível j da árvore, usamos

- ▷ 2^j processadores, cada um fazendo um merge.
- ▷ Para cada merge, cada processador usa $\frac{tam}{2}$ processadores.
- ▷ Logo, no nível j usamos $2^j \cdot \frac{tam}{2}$ processadores

$$2^j \cdot \frac{tam}{2} = 2^j \cdot \frac{\frac{n}{2^j}}{2} = \frac{n}{2}$$

O problema da envoltória convexa

O problema da envoltória convexa

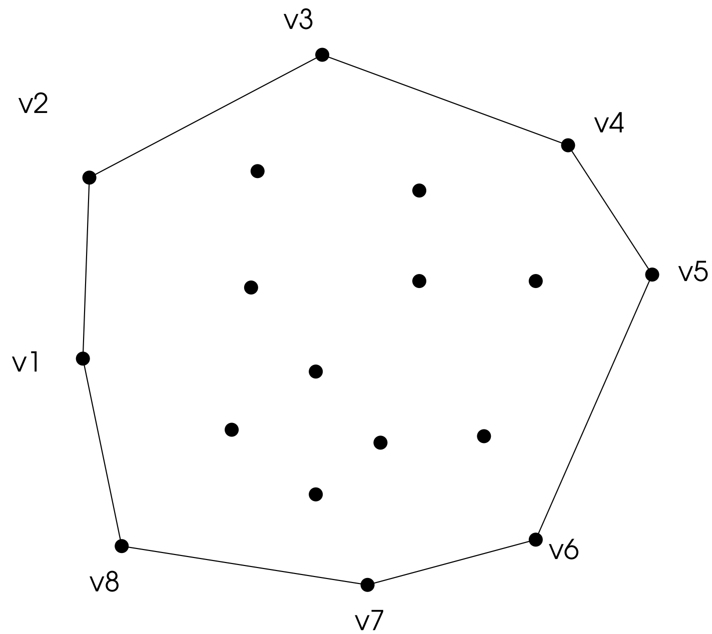
Dado um conjunto $S = \{p_1, p_2, \cdots, p_n\}$ de n pontos no plano, cada um representado pelas suas coordenadas (x, y), a **envoltória convexa planar** de S é o menor polígono convexo contendo todos os n pontos de S.

O problema da envoltória convexa

O problema da envoltória convexa é o de determinar a lista ordenada (sentido horário) $CH(S)$ de pontos de S definindo a fronteira da envoltória convexa de S.

O problema da envoltória convexa

Considere o conjunto S de pontos abaixo. O Fecho convexo de S é dado por $CH(S) = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$



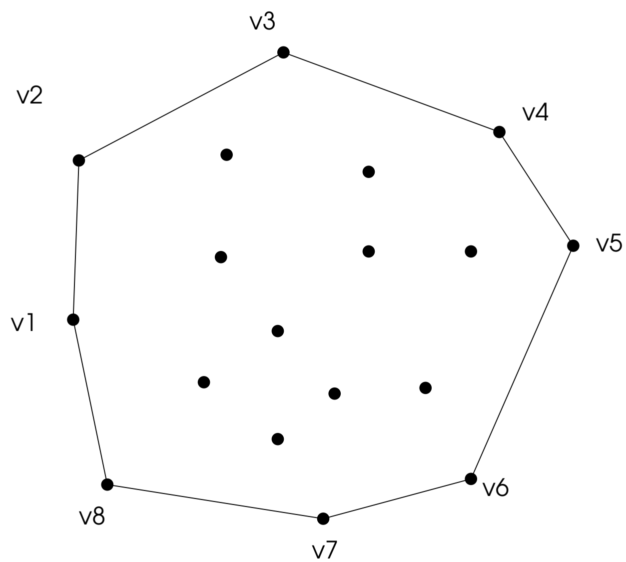
Algoritmo paralelo para o problema da envoltória convexa

Algoritmo paralelo para o problema da envoltória convexa

Sejam p e q pontos de S com a menor e a maior coordenada x, respectivamente. Claramente p e q pertencem a $CH(S)$ e particionam $CH(S)$ em uma envoltória superior $UH(S)$ consistindo de todos os pontos de p e q de $CH(S)$ (sentido horário) e uma envoltória inferior $LH(S)$ definida de modo análogo de p a q.

Algoritmo paralelo para o problema da envoltória convexa

Considere o conjunto S de pontos abaixo.



- $\triangleright$ $UH(S) = \{v_1, v_2, v_3, v_4, v_5\}$
- $\triangleright$ $LH(S) = \{v_5, v_6, v_7, v_8, v_1\}$

Algoritmo paralelo para o problema da envoltória convexa

▷ Vamos mostrar como computar $UH(S)$. A computação de $LH(S)$ é feita de modo análogo.
▷ A ordenação pode ser feito em uma EREW PRAM em tempo $O(\log n)$ com n processadores
▷ Assumimos por simplicidade que dados dois pontos quaisquer de S , eles não possuem a mesma coordenada x ou y e que n é potência de 2.

Algoritmo paralelo para o problema da envoltória convexa

Iniciamos com a ordenação dos pontos p_i pelas suas coordenadas x.

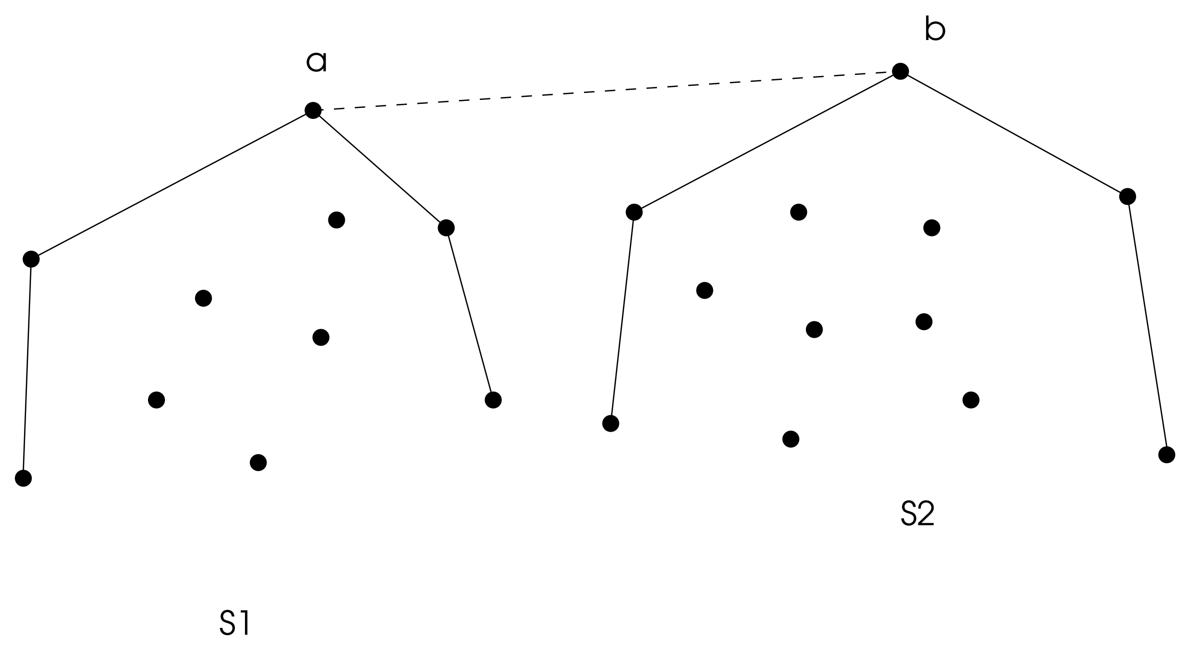
Seja $x(p_1) < x(p_2) < \cdots < x(p_n)$, onde $x(p_i)$ é a coordenada x de p_i

Algoritmo paralelo para o problema da envoltória convexa

Seja $S_1 = (p_1, p_2, \cdots, p_{\frac{n}{2}})$ e $S_2 = (p_{\frac{n}{2}+1}, p_{\frac{n}{2}+2}, \cdots, p_n)$.

Vamos supor que $UH(S_1)$ e $UH(S_2)$ é a tangente comum tal que $UH(S_1)$ e $UH(S_2)$ estão abaixo dela.

Algoritmo paralelo para o problema da envoltória convexa



O segmento de linha (a, b) é a tangente comum superior da envoltória de S_1 e S_2

Algoritmo paralelo para o problema da envoltória convexa

A computação da tangente comum superior entre $UH(S_1)$ e $UH(S_2)$ pode ser feita em tempo sequencial $O(\log n)$, usando o método de busca binária. Isso pode ser feito de forma mais eficiente.

Algoritmo paralelo para o problema da envoltória convexa

Sejam $UH(S_1) = (q_1, \cdots, q_s)$ e $UH(S_2) = (q'_1, \cdots, q'_t)$ as envoltórias superiores de S_1 e S_2, respectivamente, dados na ordem da esquerda para a direita. Observe que a tangente comum superior tenha sido deteminada e seja dado por (q_i, q'_j).

Algoritmo paralelo para o problema da envoltória convexa

Então, $UH(S)$ é o vetor consistindo das primeiras i entradas de $UH(S_1)$ e as últimas $t-j+1$ entradas de $UH(S_2)$; isto é, $UH(S) = (q_1, \cdots, q_i, q'_j, \cdots, q'_t)$. Se s e t são dados, uma vez que i e j são conhecidos, $UH(S)$ e seu tamanho pode ser determinado em tempo paralelo $O(1)$ com n processadores.

Algoritmo paralelo para o problema da envoltória convexa

Entrada:

$\triangleright$ Um conjunto S de n pontos no plano, dos quais não existam dois pontos que tenham as mesmas coordenadas x ou y, tal que $x(p_1) < x(p_1) < \cdots < x(p_n)$, onde n é uma potência de 2.

Saída:

$\triangleright$ Envoltória convexa superior de S

Algoritmo paralelo para o problema da envoltória convexa

Algoritmo Envoltória superior simples

1 Se $n \leq 4$, então use um método de força bruta para determinar $UH(S)$ e finalize

2 Sejam $S_1 = (p_1, p_2, \cdots, p_{\frac{n}{2}})$ e $S_2 = (p_{\frac{n}{2}+1}, p_{\frac{n}{2}+2}, \cdots, p_n)$. Recursivamente, compute $UH(S_1)$ e $UH(S_2)$ em paralelo.

3 Encontre a tangente comum superior entre $UH(S_1)$ e $UH(S_2)$ e deduza a envoltória convexa superior de S.

Algoritmo paralelo para o problema da envoltória convexa

Submodelo CREW

- ▷ Tempo: $O(\log^2 n)$
- ▷ Processadores: $O(n \log n)$

Fim