コンピュータ科学特別講義Ⅳ

Parallel Algorithm Design（#7）

Masato Edahiro

June 29, 2018

Please download handouts before class from
http://www.pdsl.jp/class/utyo2018/

Graduate School of Information Science, Nagoya Univ.
Parallel & Distributed Systems Lab.

PDSL

Contents of This Class

- Our Target
  - Understand Systems and Algorithms on “Multi-Core” processors
- Schedule (Tentative)
  - #1 April 6 (=Today) What’s “Multi-Core”?
  - #2 April 13 : Parallel Programming Languages (Ex. 1)
  - April 20, 27, May 4, 11, 18: NO CLASS
  - #3 May 25 : Parallel Algorithm Design
  - #4 June 1 (Fri) : Laws on Multi-Core
  - #5 June 8 : Examples of Parallel Algorithms (1) (Ex. 2)
  - June 15: NO CLASS
  - #6 June 22 : Examples of Parallel Algorithms (2)
  - #7 June 29 : Examples of Parallel Algorithms (3)
  - #8 July 6 : Examples of Parallel Algorithms (4)
  - #9 July 13 : Examples of Parallel Algorithms (5) (Ex. 3)
  - (July 20)
  - If you want to graduate in August, ask Edahiro asap.

Graduate School of Information Science, Nagoya Univ.
Parallel & Distributed Systems Lab.
PDSL

Today's Topic

- Parallel Prefix

  $X = \{x_1, x_2, \ldots, x_n\}$を集合$Y$に含まれる要素とする。また結合律を満たし、$Y$に関して閉じている二項演算$\otimes$を考える。（ここでは交換律は条件としない）

  Parallel Prefixとは次のn 個のprefixを計算することである
  $x_1, x_1 \otimes x_2, x_1 \otimes x_2 \otimes x_3, \ldots, x_1 \otimes x_2 \otimes \cdots \otimes x_n$

- 結合律: $(x_i \otimes x_j) \otimes x_k = x_i \otimes (x_j \otimes x_k)$
- 交換律: $x_i \otimes x_j = x_j \otimes x_i$
- 閉じる: $x_i \otimes x_j$ もまたYの要素
- $\otimes$の例: 加算、積算、最大値、最小値、AND, OR, XORなど
- Lower bound = $\Omega(n)$

例

加算

x	4	3	6	2	1	5
p	4	7	13	15	16	21

最小値

x	4	3	6	2	1	5
p	4	3	3	2	1	1

逐次アルゴリズム

$p_1 = x_1$

For i=1 to n-1 do

$p_{i+1} = p_i \otimes x_{i+1}$

End For

出力：$\{p_1, p_2, \ldots, p_n\}$

コスト=プロセッサ数×時間複雑度

$=\Omega(n)$

並列アルゴリズム

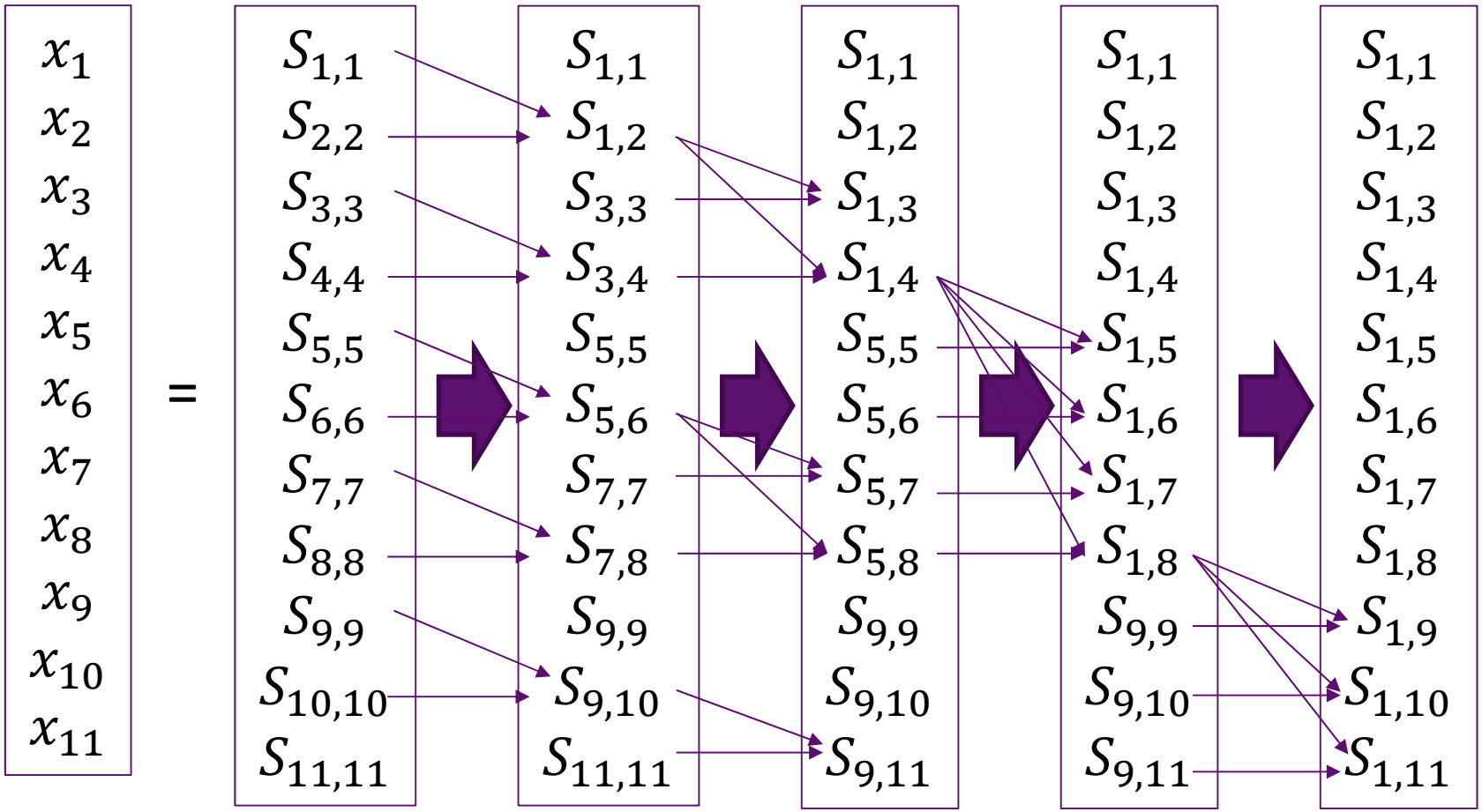
- セグメント $s_{i,j} = \{x_i, x_{i+1}, \ldots, x_j\}$
- セグメント $s_{i,j}$に対する最後のprefix $S_{i,j}$:

$$S_{i,j} = x_i \otimes x_{i+1} \otimes \cdots \otimes x_j$$

- Parallel Prefix $p_k = S_{1,k}\ (k = 1, \ldots, n)$

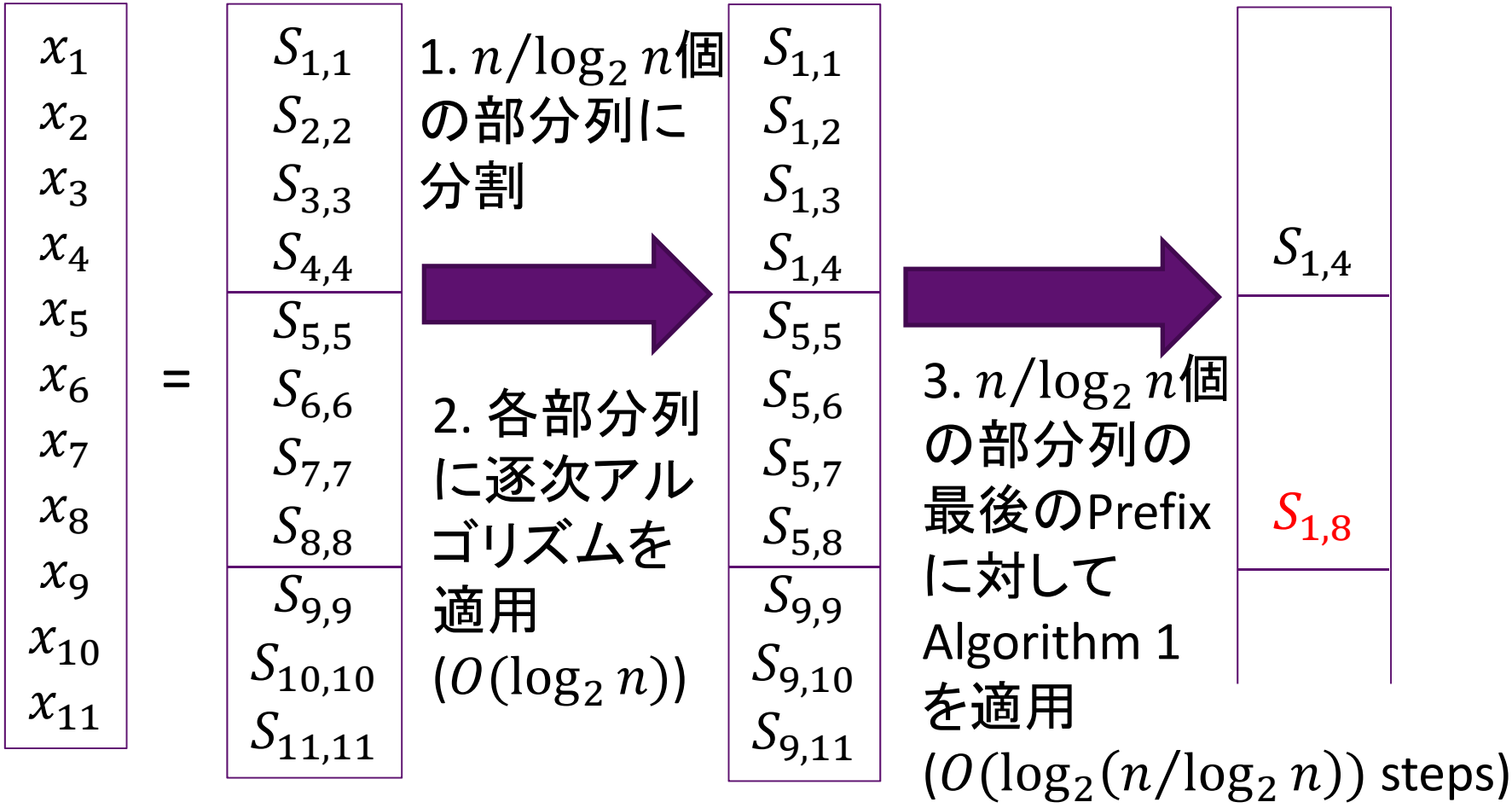
Algorithm 1

- $\lceil \log_2 n \rceil$ steps, n processors (Cost=$O(n \log_2 n)$)



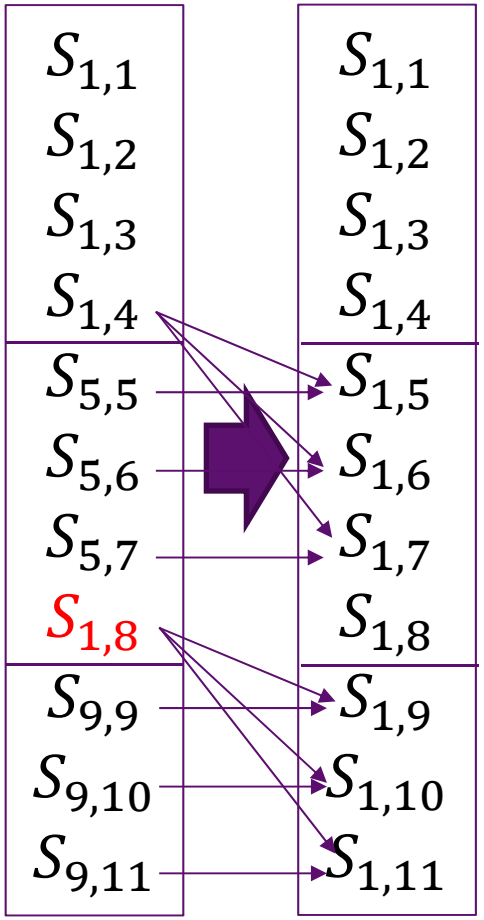
Algorithm 2 (1)

- $O(\log_2 n)$ time, $n/\log_2 n$ processors (Cost=$O(n)$)



Algorithm 2 (2)

- $O(\log_2 n)$ time, $n/\log_2 n$ processors (Cost=$O(n)$)



4. 各部分列で
逐次アルゴリズムを
適用 $(O(\log_2 n))$

Maximum Sum Subsequence

- 要素列$X = \langle x_1, x_2, \ldots, x_n \rangle$が与えられているとき、$x_u + x_{u+1} + \cdots + x_v$が最も大きくなるような連続した部分列$\langle x_u, x_{u+1}, \ldots, x_v \rangle$に対応した添字$u$と$v$ $(u \leq v)$を求める

逐次アルゴリズム

```
Global_Max←x0
u←0   // Start Index of Global Max Subsequence
v←0   // End Index of Global Max Subsequence
Current_Max←x0
q←0   // Index of Current Subsequence
For i=0 to n-1 do
  If Current_Max ≧ 0 Then
    Current_Max←Current_Max+xi
  Else
    Current_Max←xi
    q←i   // Reset Index of Current Subsequence
  End Else
  If Current_Max>Global_Max Then
    Global_Max←Current_Max
    u←q
    v←i
  End If
End For
```

例

i	x_i	Global_Max	u	v	Current_Max	q
0	5	5	0	0	5	0
1	3	8	0	1	8 (Max!)	0
2	-2	8	0	1	6	0
3	4	10	0	3	10 (Max!)	0
4	-6	10	0	3	4	0
5	-5	10	0	3	-1 (Reset!)	0
6	1	10	0	3	1	6
7	10	11	6	7	11 (Max!)	6
8	-2	11	6	7	9	6

並列アルゴリズム(1)

1. Parallel Prefix Sum $p_i = x_1 + x_2 + \cdots + x_i$ (for i = 1, … , n)を計算
2. Parallel Postfix Maximum $m_i = \max(p_i, p_{i+1}, \ldots, p_n)$ (for i = 1, … , n)を計算し、対応する添字a_iを求める。すなわち、

$$p_{a_i} = \max(p_i, p_{i+1}, \ldots, p_n)$$

   - 交換律を満たす演算を用いて$X = \{x_1, x_2, \ldots, x_n\}$に対するParallel postfix 計算は$\{x_n, x_{n-1}, \ldots, x_1\}$に対するParallel prefix 計算と同じになる

並列アルゴリズム(2)

3. $x_u + x_{u+1} + \cdots + x_v = p_v - p_u + x_u$であるため、$u$ からはじまるmaximum sum subsequenceは、$\max(p_u, p_{u+1}, \ldots, p_n) - p_u + x_u = m_u - p_u + x_u$となる。
   それゆえ、$m_{u^*} - p_{u^*} + x_{u^*} \geq m_u - p_u + x_u$ $(1 \leq \forall u \leq n)$となる$u^*$を求めることにより、$u^*$ から a_{u^*}がMaximum Sum Subsequence となる
   - すべてのuに対する“$m_u - p_u + x_u$”の計算は容易に並列化可能。u^* はreductionにより求まる

Algorithm 2 を用いてp_iを計算せよ

i	x_i					
0	5					
1	3					
2	-2					
3	4					
4	-6					
5	-5					
6	1					
7	10					
8	-2					

Maximum Sum Subsequenceを求めよ
(Algorithm 2を使う必要はない)

i	x_i	p_i	m_i	a_i	$m_i-p_i+x_i$

Array Packing

- データの配列が与えられており、その一部がマークされている。このとき、マークされているデータがマークされていないデータより前になるように並び替える

- 逐次アルゴリズム
  - マークされている要素に0を、されていない要素に1を付け、$O(n)$ の整数ソートアルゴリズムを使う
- Parallel Prefix Sumを用いた並列アルゴリズム

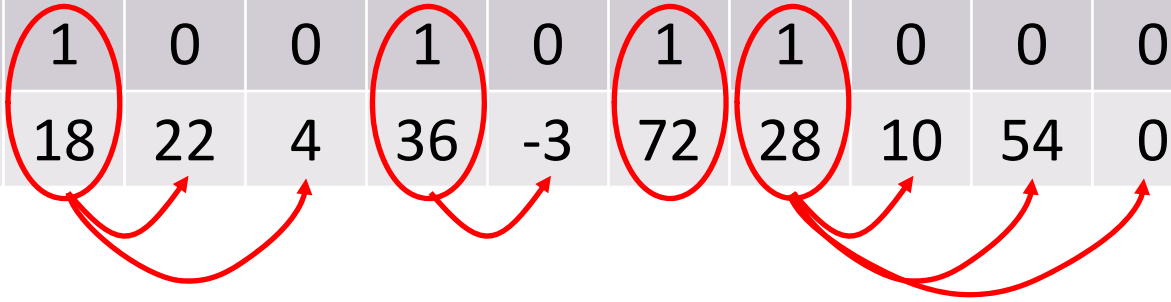
Array Packing問題に対する並列アルゴリズム

1. マークされている要素に1を、されていない要素に0を付け、Parallel Prefix Sumを用いると、出力配列においてマークされている要素の場所が計算できる
2. マークされていない要素に1を、されている要素に0を付け、Parallel Postfix Sumを用いると、出力配列においてマークされている要素の後ろから数えた場所が計算できる
3. 各要素を出力配列に並列に転送する

Segmented Broadcasting

- セグメント化された集団にデータをブロードキャストする

Item Index x_i	0	1	2	3	4	5	6	7	8	9
Leader	1	0	0	1	0	1	1	0	0	0
Data (x_i)	18	22	4	36	-3	72	28	10	54	0



Item Index x_i	0	1	2	3	4	5	6	7	8	9
Leader	1	0	0	1	0	1	1	0	0	0
Data	18	22	4	36	-3	72	28	10	54	0
Leader Index	0	0	0	3	3	5	6	6	6	6
Leader Data	18	18	18	36	36	72	28	28	28	28

Segmented Broadcastingに対する並列アルゴリズム

- 各要素x_iに対し、「リーダー」であれば(i, x_i)を付加。それ以外は$(-1, x_i)$を付加する

- Prefix演算$\otimes$を以下のように定める

$$(i,a) \otimes (j,b) = \begin{cases} (i,a)\ if\ i > j; \\ (j,b)\ otherwise \end{cases}$$

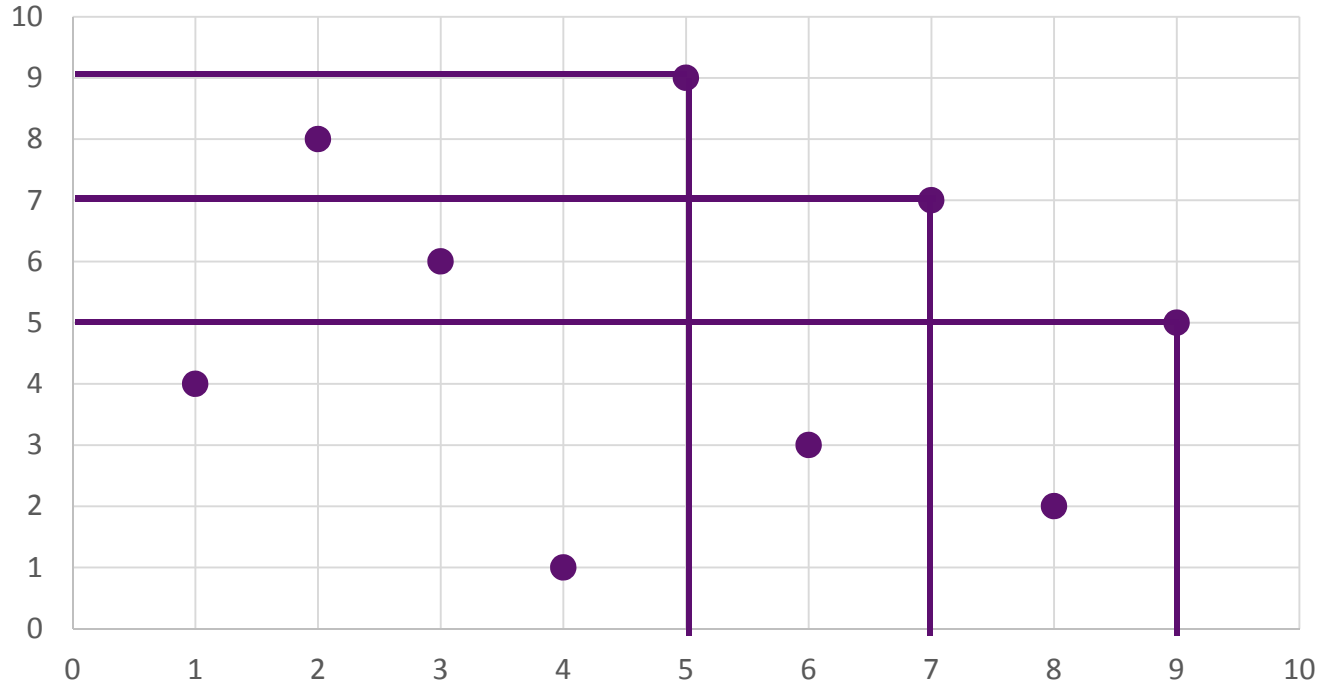
- これにより、Segmented Broadcastingは Parallel Prefix により解ける

Algorithm 2を用いてSegmented Broadcastingを求めよ

Index	Leader	Data					
0	1	18					
1	0	22					
2	0	4					
3	1	36					
4	0	-3					
5	1	72					
6	1	28					
7	0	10					
8	0	54					
9	0	0					

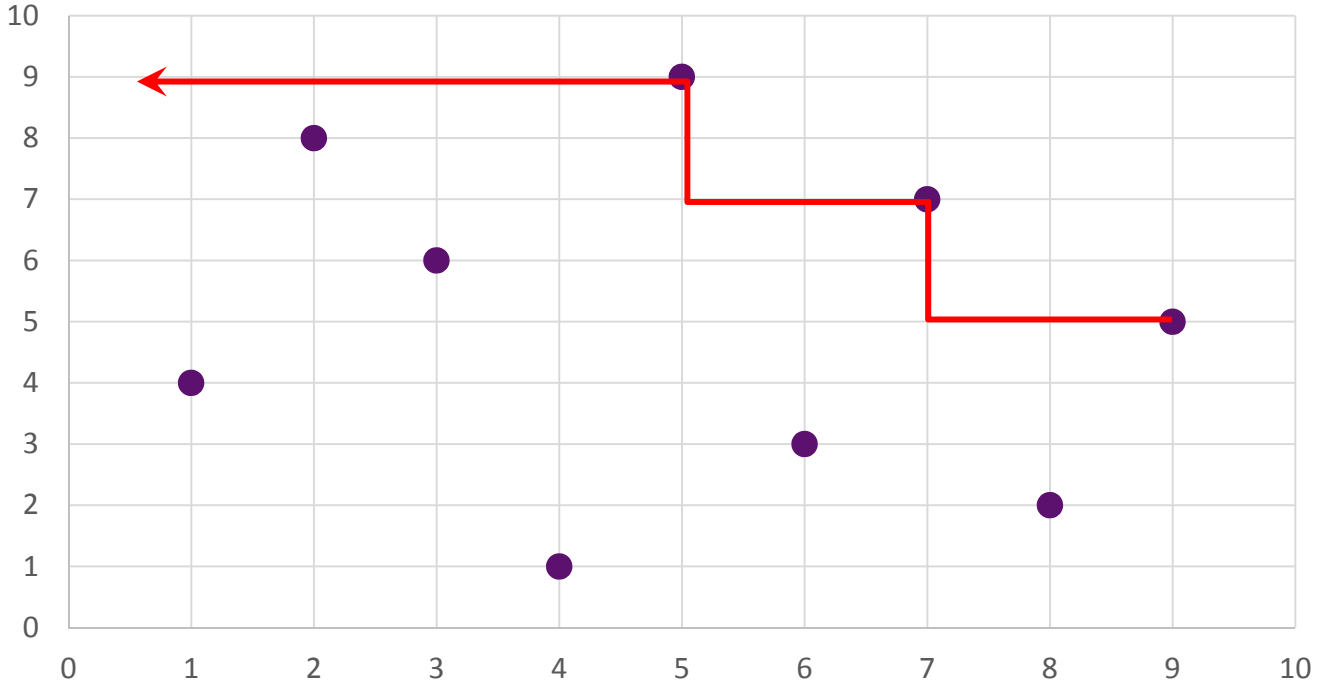
Point Domination Query

- ２点(x_1, y_1)、(x_2, y_2)に対し、$x_1 > x_2$ かつ $y_1 > y_2$ である場合、(x_1, y_1) は(x_2, y_2)を支配するという
- 与えられた点集合に対し、他の点に支配されない点の集合を求める



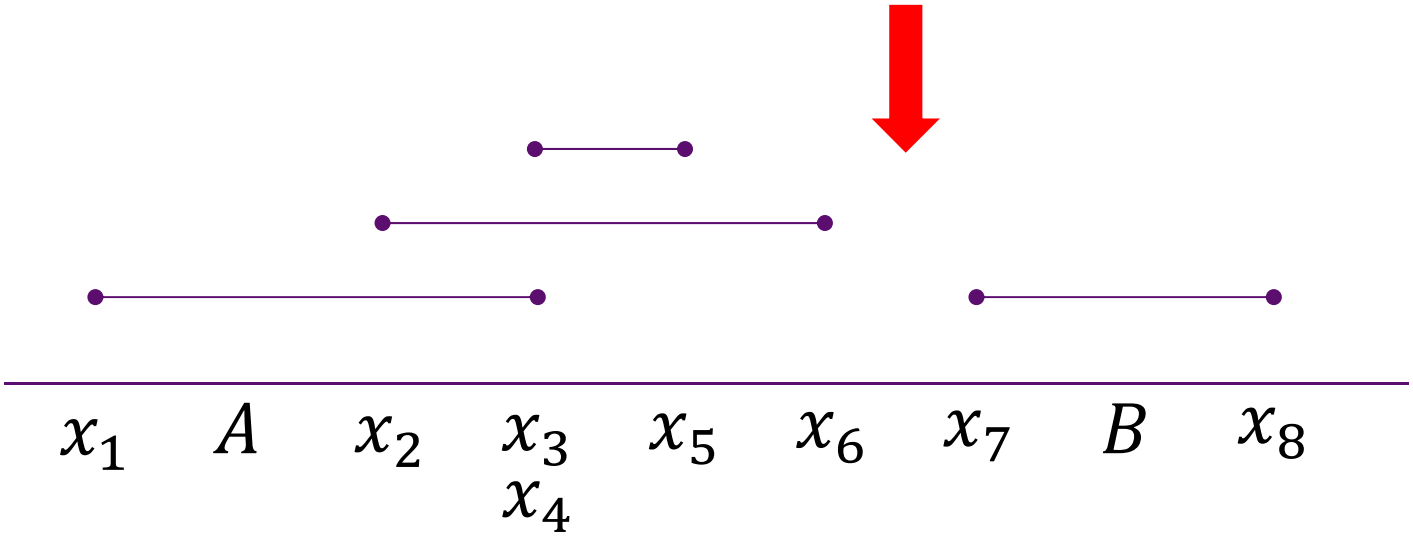
Point Domination Queryに対する並列アルゴリズム

- yに対する最大値をprefix演算とし、Parallel Postfix Computationを用いる



Overlapping Line Segments

1. Coverage Query
   - x軸上に線分集合と区間(A,B)が与えられる。これに対し、区間が線分に完全に含まれるか調べる
   - x座標は昇順に与えられるものとする



Coverage Queryに対する並列アルゴリズム

- 左端点を1、右端点を-1とするデータを考える
  - 2つ以上の端点が同じx座標を持つときには、左端点の方を先に処理する
- parallel prefix sumを使う

point	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
record	1	1	1	-1	-1	-1	1	-1
prefix	1	2	3	2	1	0	1	0

A B

Overlapping Line Segments

2. Maximum Overlapping Point
   - x軸上に線分集合が与えられる。もっとも多くの線分に含まれる点を求める
   - ２つ以上の端点が同じx座標を持つときには、左端点の方を先に処理する
     1. x_iを被覆する線分数c_i を求める
     2. c_iを最大化するi を求める