Processing Order Scheduling in Disk-based Graph Processing

Basic Ideas

Yuantian Ding

Matrix-Vector Multiplication

Programming model

- PageRank: An example of Markov chain **stationary distribution** computation.
- Other Algorithm can be viewed as **iteration of mat-vac multiplication**.

$$A = dM + \frac{1-d}{N} ee^T$$

$$P_{n+1} = AP_n$$

Problem: Matrix is Large

- In many real world problems, the matrix (even if it is sparse) is too large and can not fit into memory.
- Distributed Strategy? Recently, research have shown that single machine disk-based method **outperform** many distributed ones.
- Many disk-based graph processing system have a lots of IO overhead.
- **Our Approach:** By reordering task order, we reduce almost 50% data loading from disk.

Partition Strategy

- Since not all data can fit into memory, we need to divide the whole graph/matrix into **multiple partitions**.

$$P = \begin{pmatrix} P_{11} & P_{12} & \cdots & P_{1n} \\ P_{21} & P_{22} & \cdots & P_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ P_{n1} & P_{n2} & \cdots & P_{nn} \end{pmatrix}$$

Example: 2 Iteration

$$\begin{pmatrix} x_1^2 \\ x_2^2 \\ x_3^2 \\ x_4^2 \end{pmatrix} = \begin{pmatrix} P_{11}x_1^1 + P_{12}x_2^1 + P_{13}x_3^1 + P_{14}x_4^1 \\ P_{21}x_1^1 + P_{22}x_2^1 + P_{23}x_3^1 + P_{24}x_4^1 \\ P_{31}x_1^1 + P_{32}x_2^1 + P_{33}x_3^1 + P_{34}x_4^1 \\ P_{41}x_1^1 + P_{42}x_2^1 + P_{43}x_3^1 + P_{44}x_4^1 \end{pmatrix}$$

$$\mathbf{x}^2 = \mathbf{P}\mathbf{x}^1$$

$$\begin{pmatrix} x_1^1 \\ x_2^1 \\ x_3^1 \\ x_4^1 \end{pmatrix} = \begin{pmatrix} P_{11}x_1^0 + P_{12}x_2^0 + P_{13}x_3^0 + P_{14}x_4^0 \\ P_{21}x_1^0 + P_{22}x_2^0 + P_{23}x_3^0 + P_{24}x_4^0 \\ P_{31}x_1^0 + P_{32}x_2^0 + P_{33}x_3^0 + P_{34}x_4^0 \\ P_{41}x_1^0 + P_{42}x_2^0 + P_{43}x_3^0 + P_{44}x_4^0 \end{pmatrix}$$

$$\mathbf{x}^1 = \mathbf{P}\mathbf{x}^0$$

- First, we focus on the first 2 iteration of the whole graph.

Example: 2 Iteration

$$\begin{pmatrix} x_1^2 \\ x_2^2 \\ x_3^2 \\ x_4^2 \end{pmatrix} = \begin{pmatrix} P_{11}x_1^1 + P_{12}x_2^1 + P_{13}x_3^1 + P_{14}x_4^1 \\ P_{21}x_1^1 + P_{22}x_2^1 + P_{23}x_3^1 + P_{24}x_4^1 \\ P_{31}x_1^1 + P_{32}x_2^1 + P_{33}x_3^1 + P_{34}x_4^1 \\ P_{41}x_1^1 + P_{42}x_2^1 + P_{43}x_3^1 + P_{44}x_4^1 \end{pmatrix}$$

$$\mathbf{x}^2 = \mathbf{P}\mathbf{x}^1$$

$$\begin{pmatrix} x_1^1 \\ x_2^1 \\ x_3^1 \\ x_4^1 \end{pmatrix} = \begin{pmatrix} P_{11}x_1^0 + P_{12}x_2^0 + P_{13}x_3^0 + P_{14}x_4^0 \\ P_{21}x_1^0 + P_{22}x_2^0 + P_{23}x_3^0 + P_{24}x_4^0 \\ P_{31}x_1^0 + P_{32}x_2^0 + P_{33}x_3^0 + P_{34}x_4^0 \\ P_{41}x_1^0 + P_{42}x_2^0 + P_{43}x_3^0 + P_{44}x_4^0 \end{pmatrix}$$

$$\mathbf{x}^1 = \mathbf{P}\mathbf{x}^0$$

- We load P12 P13 P14 in order, and calculate their multiplication with Xi and add them together

Example: 2 Iteration

$$\begin{pmatrix} x_1^2 \\ x_2^2 \\ x_3^2 \\ x_4^2 \end{pmatrix} = \begin{pmatrix} P_{11}x_1^1 + P_{12}x_2^1 + P_{13}x_3^1 + P_{14}x_4^1 \\ P_{21}x_1^1 + P_{22}x_2^1 + P_{23}x_3^1 + P_{24}x_4^1 \\ P_{31}x_1^1 + P_{32}x_2^1 + P_{33}x_3^1 + P_{34}x_4^1 \\ P_{41}x_1^1 + P_{42}x_2^1 + P_{43}x_3^1 + P_{44}x_4^1 \end{pmatrix}$$

$$\mathbf{x^2} = \mathbf{Px^1}$$

$$\begin{pmatrix} x_1^1 \\ x_2^1 \\ x_3^1 \\ x_4^1 \end{pmatrix} = \begin{pmatrix} P_{11}x_1^0 + P_{12}x_2^0 + P_{13}x_3^0 + P_{14}x_4^0 \\ P_{21}x_1^0 + P_{22}x_2^0 + P_{23}x_3^0 + P_{24}x_4^0 \\ P_{31}x_1^0 + P_{32}x_2^0 + P_{33}x_3^0 + P_{34}x_4^0 \\ P_{41}x_1^0 + P_{42}x_2^0 + P_{43}x_3^0 + P_{44}x_4^0 \end{pmatrix}$$

$$\mathbf{x^1} = \mathbf{Px^0}$$

- We load P11, and do the same thing.

Example: 2 Iteration

$$\begin{pmatrix} x_1^2 \\ x_2^2 \\ x_3^2 \\ x_4^2 \end{pmatrix} = \begin{pmatrix} P_{11}x_1^1 + P_{12}x_2^1 + P_{13}x_3^1 + P_{14}x_4^1 \\ P_{21}x_1^1 + P_{22}x_2^1 + P_{23}x_3^1 + P_{24}x_4^1 \\ P_{31}x_1^1 + P_{32}x_2^1 + P_{33}x_3^1 + P_{34}x_4^1 \\ P_{41}x_1^1 + P_{42}x_2^1 + P_{43}x_3^1 + P_{44}x_4^1 \end{pmatrix}$$

$$\mathbf{x^2} = \mathbf{Px^1}$$

$$\begin{pmatrix} x_1^1 \\ x_2^1 \\ x_3^1 \\ x_4^1 \end{pmatrix} = \begin{pmatrix} P_{11}x_1^0 + P_{12}x_2^0 + P_{13}x_3^0 + P_{14}x_4^0 \\ P_{21}x_1^0 + P_{22}x_2^0 + P_{23}x_3^0 + P_{24}x_4^0 \\ P_{31}x_1^0 + P_{32}x_2^0 + P_{33}x_3^0 + P_{34}x_4^0 \\ P_{41}x_1^0 + P_{42}x_2^0 + P_{43}x_3^0 + P_{44}x_4^0 \end{pmatrix}$$

$$\mathbf{x^1} = \mathbf{Px^0}$$

- We load P11, and do the same thing.
- And we do not discard P11 immediately, we found we can get the value of x11, and calculate another in-memory mat-vec multiplication.

Example: 2 Iteration

$$\begin{pmatrix} x_1^2 \\ x_2^2 \\ x_3^2 \\ x_4^2 \end{pmatrix} = \begin{pmatrix} P_{11}x_1^1 + P_{12}x_2^1 + P_{13}x_3^1 + P_{14}x_4^1 \\ P_{21}x_1^1 + P_{22}x_2^1 + P_{23}x_3^1 + P_{24}x_4^1 \\ P_{31}x_1^1 + P_{32}x_2^1 + P_{33}x_3^1 + P_{34}x_4^1 \\ P_{41}x_1^1 + P_{42}x_2^1 + P_{43}x_3^1 + P_{44}x_4^1 \end{pmatrix}$$

$$\mathbf{x^2 = Px^1}$$

$$\begin{pmatrix} x_1^1 \\ x_2^1 \\ x_3^1 \\ x_4^1 \end{pmatrix} = \begin{pmatrix} P_{11}x_1^0 + P_{12}x_2^0 + P_{13}x_3^0 + P_{14}x_4^0 \\ P_{21}x_1^0 + P_{22}x_2^0 + P_{23}x_3^0 + P_{24}x_4^0 \\ P_{31}x_1^0 + P_{32}x_2^0 + P_{33}x_3^0 + P_{34}x_4^0 \\ P_{41}x_1^0 + P_{42}x_2^0 + P_{43}x_3^0 + P_{44}x_4^0 \end{pmatrix}$$

$$\mathbf{x^1 = Px^0}$$

- As we have the value of x11, when we load **P21, P31, P41**, we can do this mat-vec multiplication in both these **two iteration at same time.**

Example: 2 Iteration

$$\begin{pmatrix} x_1^2 \\ x_2^2 \\ x_3^2 \\ x_4^2 \end{pmatrix} = \begin{pmatrix} P_{11}x_1^1 + P_{12}x_2^1 + P_{13}x_3^1 + P_{14}x_4^1 \\ P_{21}x_1^1 + P_{22}x_2^1 + P_{23}x_3^1 + P_{24}x_4^1 \\ P_{31}x_1^1 + P_{32}x_2^1 + P_{33}x_3^1 + P_{34}x_4^1 \\ P_{41}x_1^1 + P_{42}x_2^1 + P_{43}x_3^1 + P_{44}x_4^1 \end{pmatrix}$$

$$\mathbf{x}^2 = \mathbf{P}\mathbf{x}^1$$

$$\begin{pmatrix} x_1^1 \\ x_2^1 \\ x_3^1 \\ x_4^1 \end{pmatrix} = \begin{pmatrix} P_{11}x_1^0 + P_{12}x_2^0 + P_{13}x_3^0 + P_{14}x_4^0 \\ P_{21}x_1^0 + P_{22}x_2^0 + P_{23}x_3^0 + P_{24}x_4^0 \\ P_{31}x_1^0 + P_{32}x_2^0 + P_{33}x_3^0 + P_{34}x_4^0 \\ P_{41}x_1^0 + P_{42}x_2^0 + P_{43}x_3^0 + P_{44}x_4^0 \end{pmatrix}$$

$$\mathbf{x}^1 = \mathbf{P}\mathbf{x}^0$$

- We can do this recursively in the inner matrix (P22 ~ P44).

Example: 2 Iteration

$$\begin{pmatrix} x_1^2 \\ x_2^2 \\ x_3^2 \\ x_4^2 \end{pmatrix} = \begin{pmatrix} P_{11}x_1^1 + P_{12}x_2^1 + P_{13}x_3^1 + P_{14}x_4^1 \\ P_{21}x_1^1 + P_{22}x_2^1 + P_{23}x_3^1 + P_{24}x_4^1 \\ P_{31}x_1^1 + P_{32}x_2^1 + P_{33}x_3^1 + P_{34}x_4^1 \\ P_{41}x_1^1 + P_{42}x_2^1 + P_{43}x_3^1 + P_{44}x_4^1 \end{pmatrix}$$

$$\mathbf{x^2} = \mathbf{Px^1}$$

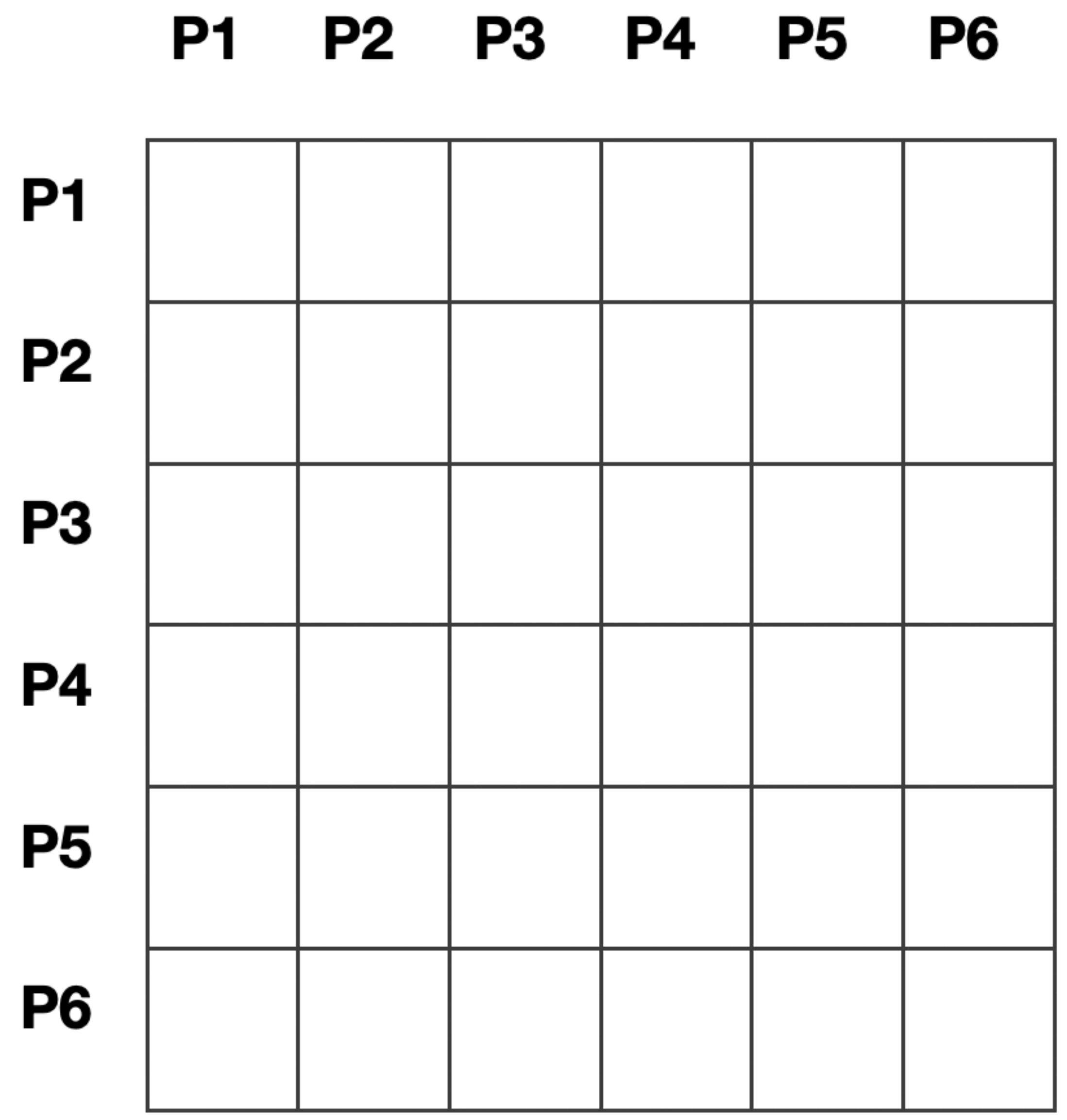
$$\begin{pmatrix} x_1^1 \\ x_2^1 \\ x_3^1 \\ x_4^1 \end{pmatrix} = \begin{pmatrix} P_{11}x_1^0 + P_{12}x_2^0 + P_{13}x_3^0 + P_{14}x_4^0 \\ P_{21}x_1^0 + P_{22}x_2^0 + P_{23}x_3^0 + P_{24}x_4^0 \\ P_{31}x_1^0 + P_{32}x_2^0 + P_{33}x_3^0 + P_{34}x_4^0 \\ P_{41}x_1^0 + P_{42}x_2^0 + P_{43}x_3^0 + P_{44}x_4^0 \end{pmatrix}$$

$$\mathbf{x^1} = \mathbf{Px^0}$$

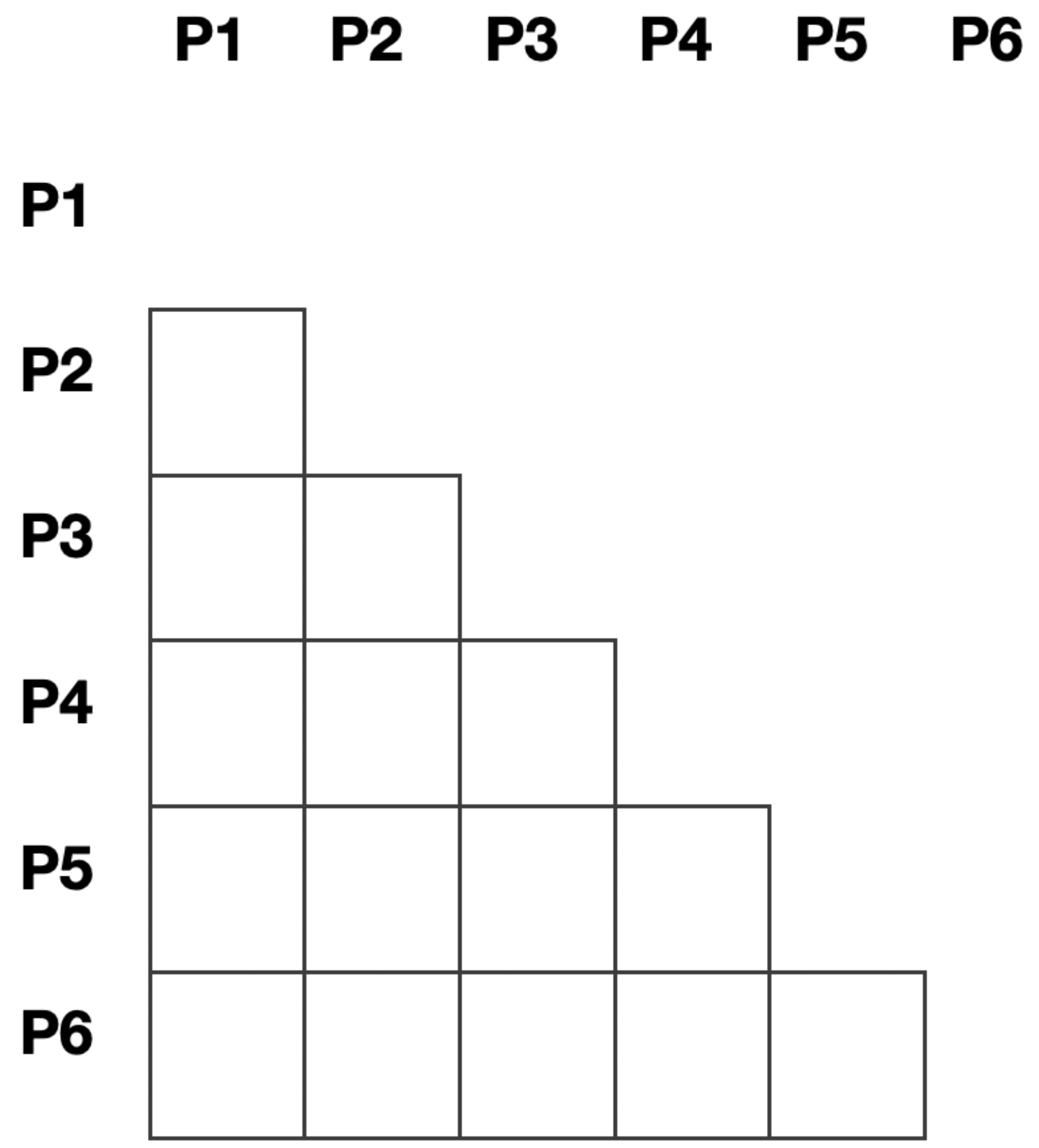
- We can do this recursively in the inner matrix (P22 ~ P44).

Lumos Task Ordering: Consider 2 Iteration

Result



Iteration t



Iteration t+1

- That means in every two iteration, we only need to load 3/4 of original grid.
- This is found in Lumos(ATC'20), however, this is not the **BEST** case.

Example: 3 Iteration

$$\begin{pmatrix} x_1^3 \\ x_2^3 \\ x_3^3 \\ x_4^3 \end{pmatrix} = \begin{pmatrix} P_{11}x_1^2 + P_{12}x_2^2 + P_{13}x_3^2 + P_{14}x_4^2 \\ P_{21}x_1^2 + P_{22}x_2^2 + P_{23}x_3^2 + P_{24}x_4^2 \\ P_{31}x_1^2 + P_{32}x_2^2 + P_{33}x_3^2 + P_{34}x_4^2 \\ P_{41}x_1^2 + P_{42}x_2^2 + P_{43}x_3^2 + P_{44}x_4^2 \end{pmatrix}$$

$$\mathbf{x^3} = \mathbf{Px^2}$$

$$\begin{pmatrix} x_1^2 \\ x_2^2 \\ x_3^2 \\ x_4^2 \end{pmatrix} = \begin{pmatrix} P_{11}x_1^1 + P_{12}x_2^1 + P_{13}x_3^1 + P_{14}x_4^1 \\ P_{21}x_1^1 + P_{22}x_2^1 + P_{23}x_3^1 + P_{24}x_4^1 \\ P_{31}x_1^1 + P_{32}x_2^1 + P_{33}x_3^1 + P_{34}x_4^1 \\ P_{41}x_1^1 + P_{42}x_2^1 + P_{43}x_3^1 + P_{44}x_4^1 \end{pmatrix}$$

$$\mathbf{x^2} = \mathbf{Px^1}$$

- We investigate the third iteration

Example: 3 Iteration

$$\begin{pmatrix} x_1^3 \\ x_2^3 \\ x_3^3 \\ x_4^3 \end{pmatrix} = \begin{pmatrix} P_{11}x_1^2 + P_{12}x_2^2 + P_{13}x_3^2 + P_{14}x_4^2 \\ P_{21}x_1^2 + P_{22}x_2^2 + P_{23}x_3^2 + P_{24}x_4^2 \\ P_{31}x_1^2 + P_{32}x_2^2 + P_{33}x_3^2 + P_{34}x_4^2 \\ P_{41}x_1^2 + P_{42}x_2^2 + P_{43}x_3^2 + P_{44}x_4^2 \end{pmatrix}$$

$$\mathbf{x^3} = \mathbf{P}\mathbf{x^2}$$

$$\begin{pmatrix} x_1^2 \\ x_2^2 \\ x_3^2 \\ x_4^2 \end{pmatrix} = \begin{pmatrix} P_{11}x_1^1 + P_{12}x_2^1 + P_{13}x_3^1 + P_{14}x_4^1 \\ P_{21}x_1^1 + P_{22}x_2^1 + P_{23}x_3^1 + P_{24}x_4^1 \\ P_{31}x_1^1 + P_{32}x_2^1 + P_{33}x_3^1 + P_{34}x_4^1 \\ P_{41}x_1^1 + P_{42}x_2^1 + P_{43}x_3^1 + P_{44}x_4^1 \end{pmatrix}$$

$$\mathbf{x^2} = \mathbf{P}\mathbf{x^1}$$

- We investigate the third iteration, after previous example, we found we have already known the result of x24
- When we load P14 P24 P34, we can calculate different multiplication in these **TWO** iteration

Example: 3 Iteration

$$\begin{pmatrix} x_1^3 \\ x_2^3 \\ x_3^3 \\ x_4^3 \end{pmatrix} = \begin{pmatrix} P_{11}x_1^2 + P_{12}x_2^2 + P_{13}x_3^2 + P_{14}x_4^2 \\ P_{21}x_1^2 + P_{22}x_2^2 + P_{23}x_3^2 + P_{24}x_4^2 \\ P_{31}x_1^2 + P_{32}x_2^2 + P_{33}x_3^2 + P_{34}x_4^2 \\ P_{41}x_1^2 + P_{42}x_2^2 + P_{43}x_3^2 + P_{44}x_4^2 \end{pmatrix}$$

$$\mathbf{x}^3 = \mathbf{P}\mathbf{x}^2$$

$$\begin{pmatrix} x_1^2 \\ x_2^2 \\ x_3^2 \\ x_4^2 \end{pmatrix} = \begin{pmatrix} P_{11}x_1^1 + P_{12}x_2^1 + P_{13}x_3^1 + P_{14}x_4^1 \\ P_{21}x_1^1 + P_{22}x_2^1 + P_{23}x_3^1 + P_{24}x_4^1 \\ P_{31}x_1^1 + P_{32}x_2^1 + P_{33}x_3^1 + P_{34}x_4^1 \\ P_{41}x_1^1 + P_{42}x_2^1 + P_{43}x_3^1 + P_{44}x_4^1 \end{pmatrix}$$

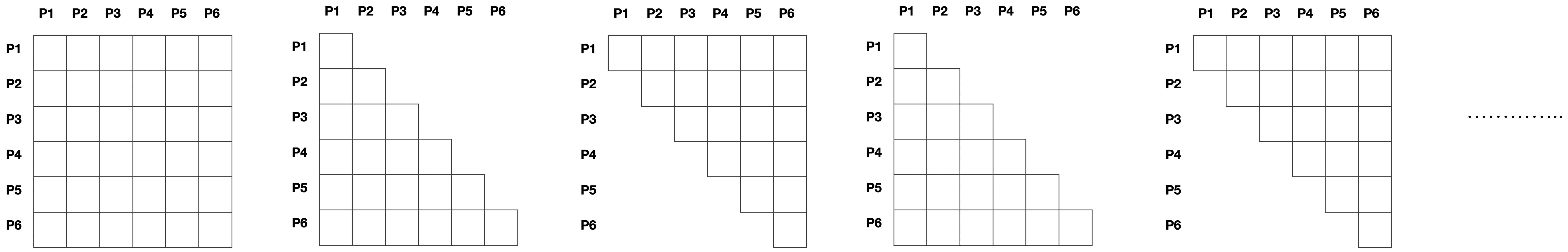
$$\mathbf{x}^2 = \mathbf{P}\mathbf{x}^1$$

- After we processing iteration 2, we found we complete half of computation in Iteration 3.
- We can do the same thing to iteration 4.

Task Ordering

Final Result

- We further found that we can reduce 50% IO overhead.



Compress CSR Graph Further

- As IO overhead is still larger than computing time, we try to divide IO time into read time and compress time.
- To reduce IO cost, we use Variable byte code(VB code) to compress CSR graph. This technique always used in a posting of search engine.
    - Sort the vertex id in CSR adjacent list.
    - Storage the difference of two vertices id, instead of their id.
    - Save this id-difference in VBCode.

Vertex ID	824	829	215406
difference	824	5	214577
encode	00000110 10111000	10000101	00001101 00001100 10110001

Thanks