A Fast Parallel SGD for Matrix Factorization in Shared Memory Systems

Yong Zhuang
Department of Computer Science
National Taiwan University

Joint work with Wei-Sheng Chin, Yu-Chin Juan, and Chih-Jen Lin

Outline

Outline

Outline

Motivation

- Matrix Factorization is an effective method for recommender systems, e.g., Netflix Prize, KDD Cup 2011, etc.
- But training is slow. When we did a HW on training KDD Cup 2011 data, it took too long for us to do experiments

Motivation (Cont'd)

- Distributed MF is possible, but complicated
- Yahoo!Music: 1 million users, 600 thousands items and 252 million ratings: can be stored in memory
- We didn't see many studies on parallel MF on multi-core systems
- Our goal is to develop a parallel MF system in shared memory systems, so at least others students didn't have the same trouble as us

Outline

Matrix Factorization

- For recommender systems: A group of users give ratings to some items

User	Item	Rating
1	5	100
1	10	80
1	13	30
...	...	...
u	v	r
...	...	...

- $(u, v) = r$

Matrix Factorization (Cont'd)

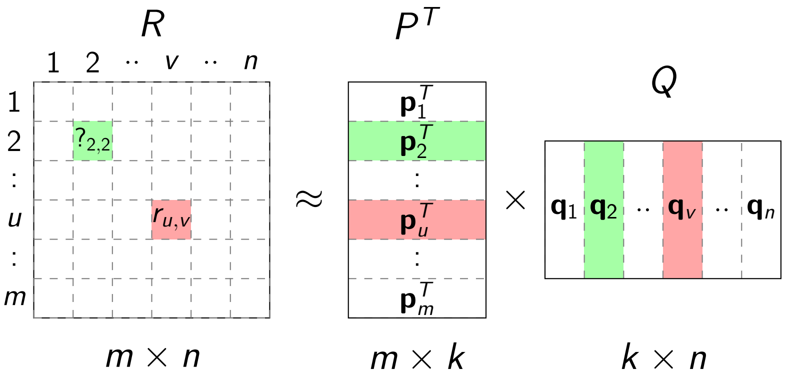
$$R$$

$$\begin{array}{c} \\ 1 \\ 2 \\ : \\ u \\ : \\ m \end{array}\begin{array}{c} \begin{array}{cccccc} 1 & 2 & \cdot\cdot & v & \cdot\cdot & n \end{array} \\ \left[\begin{array}{cccccc} & & & & & \\ & ?_{2,2} & & & & \\ & & & & & \\ & & & r_{u,v} & & \\ & & & & & \\ & & & & & \end{array}\right] \end{array}$$

$$m \times n$$

- m, n : numbers of users and items
- u, v : index for u_{th} user and v_{th} item
- $r_{u,v}$: u_{th} user gives a rating $r_{u,v}$ to v_{th} item

Matrix Factorization (Cont'd)



- k : number of latent dimensions
- $r_{u,v} = \mathbf{p}_u^T \mathbf{q}_v$
- $?_{2,2} = \mathbf{p}_2^T \mathbf{q}_2$

Matrix Factorization (Cont'd)

Objective function:

$$\min_{P,Q} \sum_{(u,v)\in R} \left(r_{u,v} - \mathbf{p}_u^T \mathbf{q}_v\right)^2 + \lambda_P \left\|\mathbf{p}_u\right\|_F^2 + \lambda_Q \left\|\mathbf{q}_v\right\|_F^2 ,$$

SGD: Loops over all ratings in the training set.

Prediction error: $e_{u,v} \equiv r_{u,v} - \mathbf{p}_u^T \mathbf{q}_v$

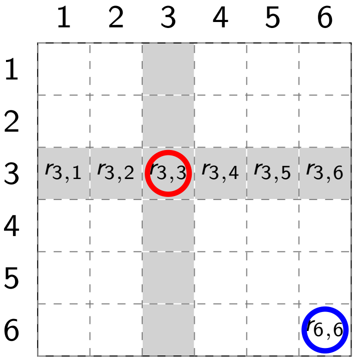
SGD update rule:

$$\begin{aligned}\mathbf{p}_u &\leftarrow \mathbf{p}_u + \gamma\left(e_{u,v}\mathbf{q}_v - \lambda_P \mathbf{p}_u\right), \\ \mathbf{q}_v &\leftarrow \mathbf{q}_v + \gamma\left(e_{u,v}\mathbf{p}_u - \lambda_Q \mathbf{q}_v\right)\end{aligned}$$

Outline

Parallel Matrix Factorization

After $r_{3,3}$ selected, the ratings in gray blocks cannot be updated due to racing condition

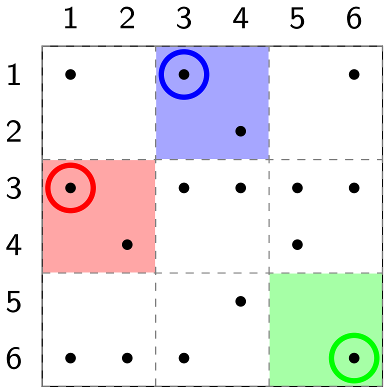


- $r_{3,1} = \mathbf{p_3}^T \mathbf{q_1}$
- $r_{3,2} = \mathbf{p_3}^T \mathbf{q_2}$
- ..
- $r_{3,6} = \mathbf{p_3}^T \mathbf{q_6}$

- $r_{3,3} = \mathbf{p_3}^T \mathbf{q_3}$
  $r_{6,6} = \mathbf{p_6}^T \mathbf{q_6}$

Parallel Matrix Factorization (Cont'd)

We can split the matrix to blocks.
Then use threads to update the blocks where ratings in different blocks don't share **p** or **q**

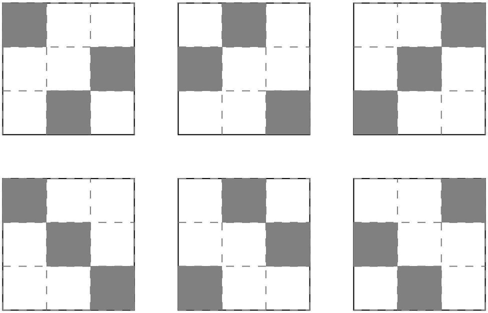


Outline

Outline

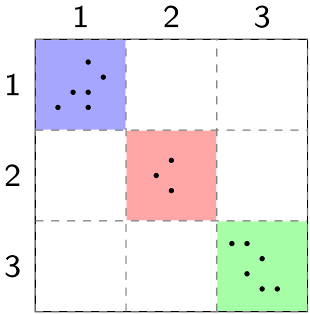
Locking Problem

- DSGD [Gemulla et al., 2011]
- Distributed system: Communication cost is an issue
  T nodes, the matrix $\rightarrow T \times T$ blocks to reduce communication cost



Locking Problem (Cont'd)

- We apply it in shared memory system



- Shared memory: Idle time will be an issuse

- Block 1: 20s
- Block 2: 10s
- Block 3: 20s

We have 3 threads

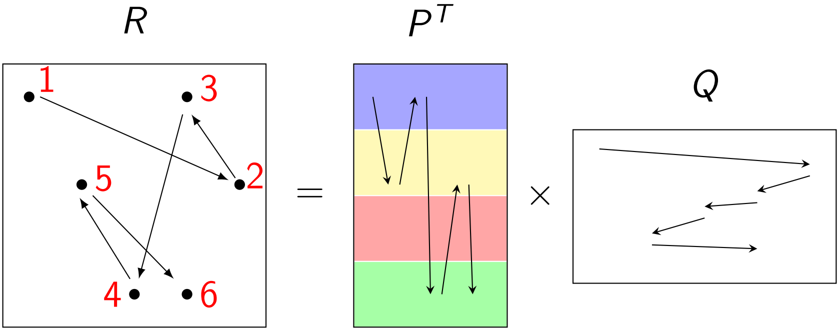
Thread	0→10	10→20
1	Busy	Busy
2	Busy	**Idle**
3	Busy	Busy

10s wasted!!

Outline

Memory Discontinuity

- HogWild [Niu et al., 2011]: assume the proability of racing condition is really low
- All R, P, and P are memory discontinuous



Outline

Our approach

- We proposes a fast parallel SGD for Matrix Factorization in shared memory systems (FPSGD)
- It applies two strategies to speed up Matrix Factorization
  - Lock-Free Scheduling
  - Partial Random Method

Outline

Lock-Free Scheduling

We split the matrix to enough blocks.
E.g., for 2 threads, we split the matrix to 4×4 blocks

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

0 is the updated counter recording the number of updated times for each block

Lock-Free Scheduling (Cont'd)

Firstly, T_1 selects a block randomly

T_1 0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Lock-Free Scheduling (Cont'd)

For T_2, it selects a block neither green nor gray randomly

T_1 0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	T_2 0

Lock-Free Scheduling (Cont'd)

After T_1 finishes, the counter for the corresponding block is added by one

1	0	0	0
0	0	0	0
0	0	0	0
0	0	0	T_2 0

Lock-Free Scheduling (Cont'd)

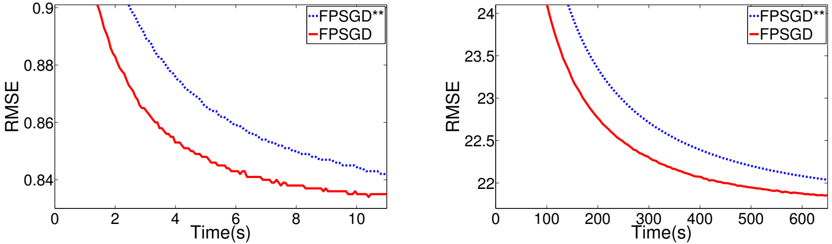
T_1 can select available blocks to update

Rule: select one that is least updated

1	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0 T_2

Lock-Free Scheduling (Cont'd)

FPSGD: applying Lock-Free Scheduling
FPSGD**: applying DSGD-like Scheduling



(a) MovieLens 10M (b) Yahoo!Music

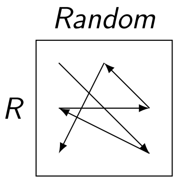
- MovieLens 10M: 18.71s → 9.72s (RMSE: 0.835)
- Yahoo!Music: 728.23s → 462.55s (RMSE: 21.985)

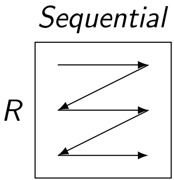
Outline

Partial Random Method

- For SGD, there are two types of update order

Update order	Advantages	Disadvantages
Random	Faster and stable	Memory discontinuity
Sequential	Memory continuity	Non-stable

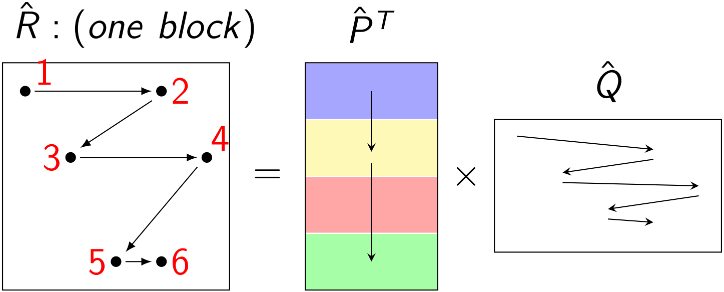




- Lock-free scheduling: the property of randomness
- How to make it more friendly to cache?

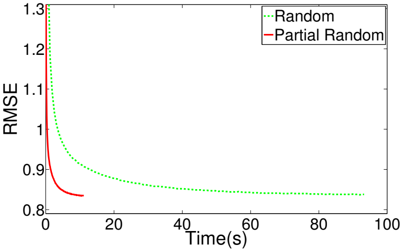
Partial Random Method (Cont'd)

Access both $\hat{R}$ and $\hat{P}$ memory continuously

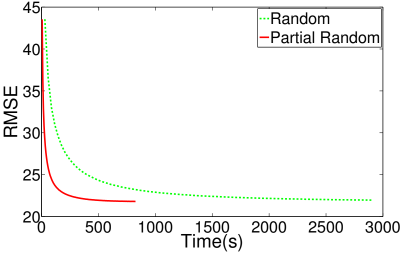


- Partial: FPSGD is sequential in **each block**
- Random: FPSGD is random when **selecting block**

Partial Random Method (Cont'd)



(a) MovieLens 10M



(b) Yahoo!Music

- The performance of Partial Random Method is better than that of Random Method

Outline

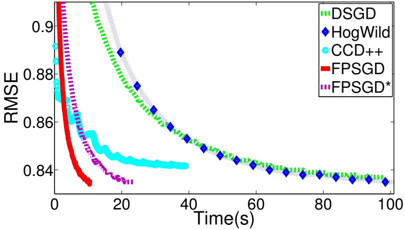
Experiments

Some of the state-of-art methods

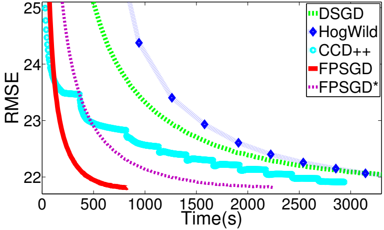
- CCD++ [Yu et al., 2012]: Coordinate descent method (**Best paper award in ICDM 2012**)
- DSGD [Gemulla et al., 2011]: Stochastic gradient descent
- HogWild [Niu et al., 2011]: Stochastic gradient descent

Experiments (Cont'd)

- We compare FPSGD with DSGD, HogWild, CCD++, and FPSGD*
- FPSGD: with fast SSE instructions
- FPSGD*: without SSE instructions



(a) MovieLens 10M



(b) Yahoo!Music

Experiments (Cont'd)

Method	Category	Problem
DSGD	Stochastic	Locking problem
HogWild	Stochastic	Memory discontinuity

- Stochastic methods may get a better solution for its randomness
- Deterministic methods (e.g., CCD++) avoid tuning learning rate which is the advantage over Stochastic methods

Outline

Conclusion

- We point out some computational bottlenecks in existing parallel SGD methods
- We propose FPSGD to address these issues and confirm its effectiveness by experiments
- 1 SGD iteration:
  1 thread: around 30s $\rightarrow$ 8 threads: around 4s for Yahoo!Music with 252 million ratings

Conclusion (Cont'd)

- We develop the package LIBMF available at http://www.csie.ntu.edu.tw/~cjlin/libmf
- Updated paper and instructions of LIBMF is at http://www.csie.ntu.edu.tw/~cjlin/papers/libmf.pdf
- Your comments to our work are very welcome