INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING
ICPP / 2021 / CHICAGO / USA
acm In-Cooperation
sighpc
AUGUST 9-12, 2021

A Novel Multi-CPU/GPU Collaborative Computing Framework for SGD-based Matrix Factorization

Yizhi Huang[*†], Yinyan Long[†], Yan Liu[*],
Shuibing He[‡], Yang Bai[*], Renfa Li[*]

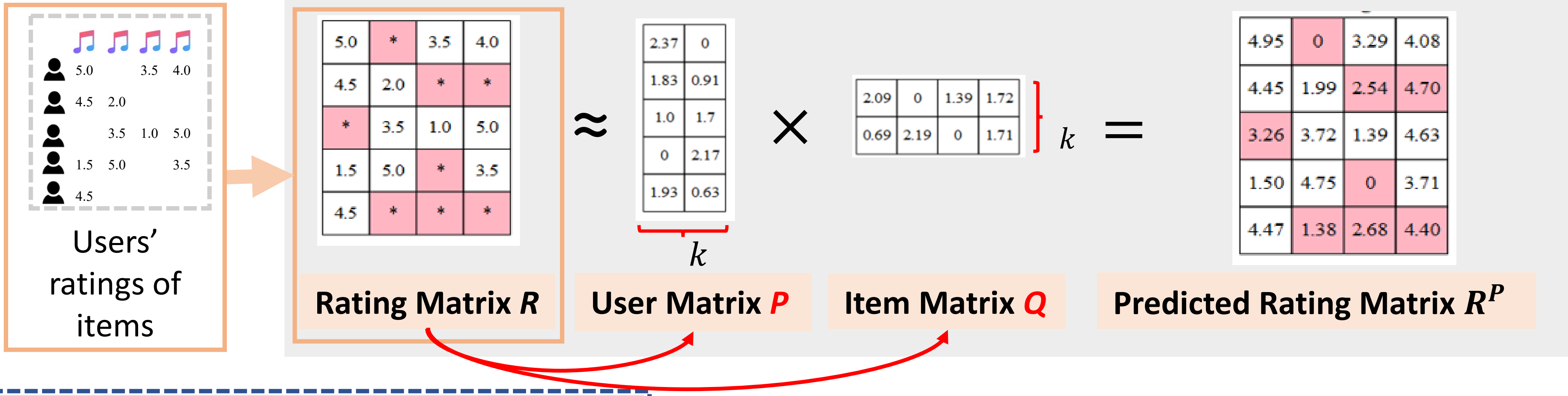
[*] 湖南大學 HUNAN UNIVERSITY [†] 之江实验室 ZHEJIANG LAB [‡] 浙江大學 ZHEJIANG UNIVERSITY

Outline

- Background and Motivation
- Design and Implementation
- Evaluation

Background

- Matrix Factorization: can help recommender systems predicted user's preferences to products.
- SGD-based MF

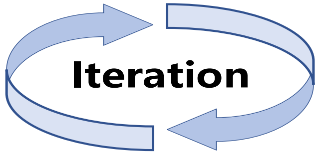


Updating feature matrix *P* ,*Q* by SGD

$$\theta(\mathbf{p}_i,\mathbf{q}_j)=\frac{1}{2}(r_{i,j}-\mathbf{p}_i\bullet\mathbf{q}_j)^2+\lambda_1\|\mathbf{P}\|^2+\lambda_2\|\mathbf{Q}\|^2$$

$$\mathbf{p}_i\leftarrow\mathbf{p}_i-\gamma\frac{\partial\theta(\mathbf{p}_i,\mathbf{q}_j)}{\partial\mathbf{p}_i}$$

$$\mathbf{q}_j\leftarrow\mathbf{q}_j-\gamma\frac{\partial\theta(\mathbf{p}_i,\mathbf{q}_j)}{\partial\mathbf{q}_j}$$

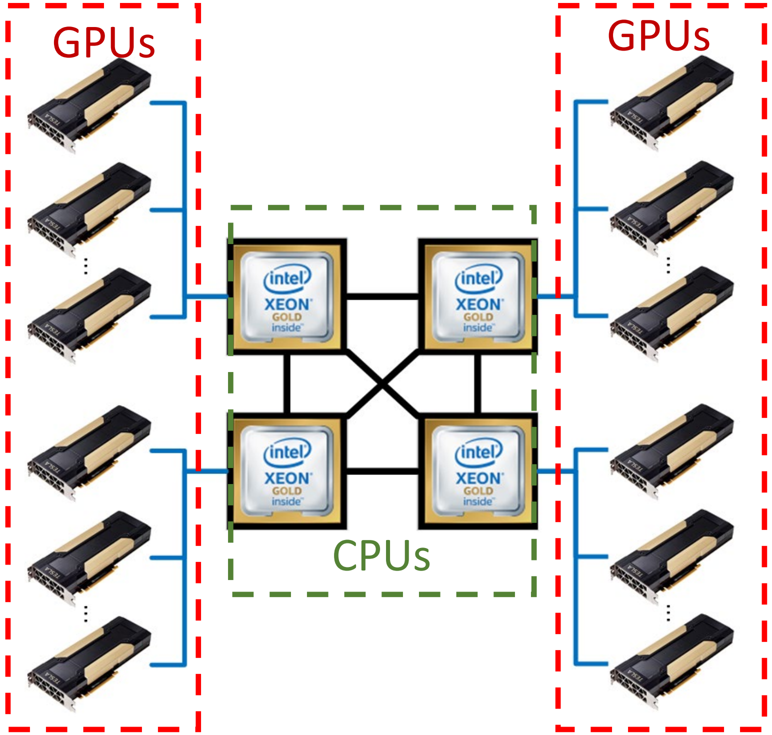


Each score r will be used to update two k-dimensional vectors p, q

Need to accelerate SGD-based MF

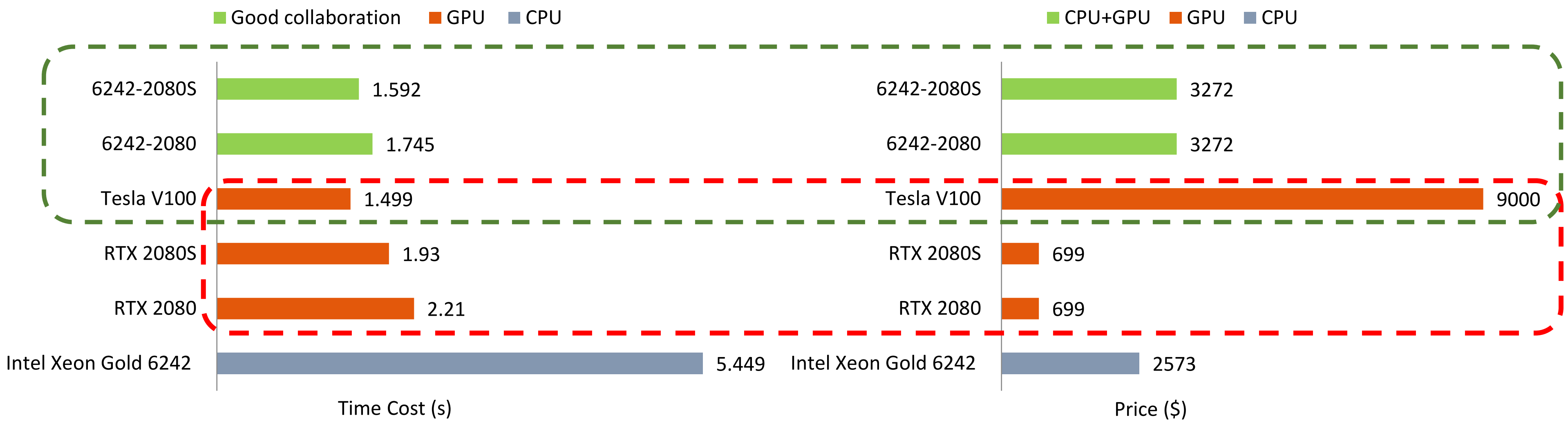
Observation: the Under-utilized CPUs

- Many computing nodes have multi-CPUs/GPUs
- Existing researches more willing to manage the GPUs for computing
- CPUs' computing power is easily overlooked
- Is it possible to cooperate with the CPUs to accelerate SGD-based MF ?



Cooperatively accelerating SDG-based MF?

Observation



- The performance of high-end GPUs does not increase linearly with price

- Cooperative computing of CPU and GPU may bring a good price/performance ratio

Challenges

Unbalanced load leads to short board effect

- How to uniformly manage and transparently use heterogeneous CPUs and GPUs ?
- How to design appropriate data distribution?
- How to optimize communication inter-CPUs/GPUs?

$$R_{m \times n} = P_{m \times k} \times Q_{k \times n}$$

Naïve Communication Cost: $(m + n) \times k \times sizeof(float) \times Iterations / B_{bus}$

Netflix: $m = 480190, n = 17771, k = 128, iterations = 20, cost = 0.4s$

Outline

- Background and Motivation
- Design and Implementation
- Evaluation

Our solution: HCC-MF

Problem 1

How to transparentize heterogeneous CPUs and GPUs

A general framework that unifies the abstraction and workflow

Problem 2

How to distribute data to each heterogeneous CPU/GPU to make the whole system more efficient ?

- A **time cost model** for guiding data Distribution.

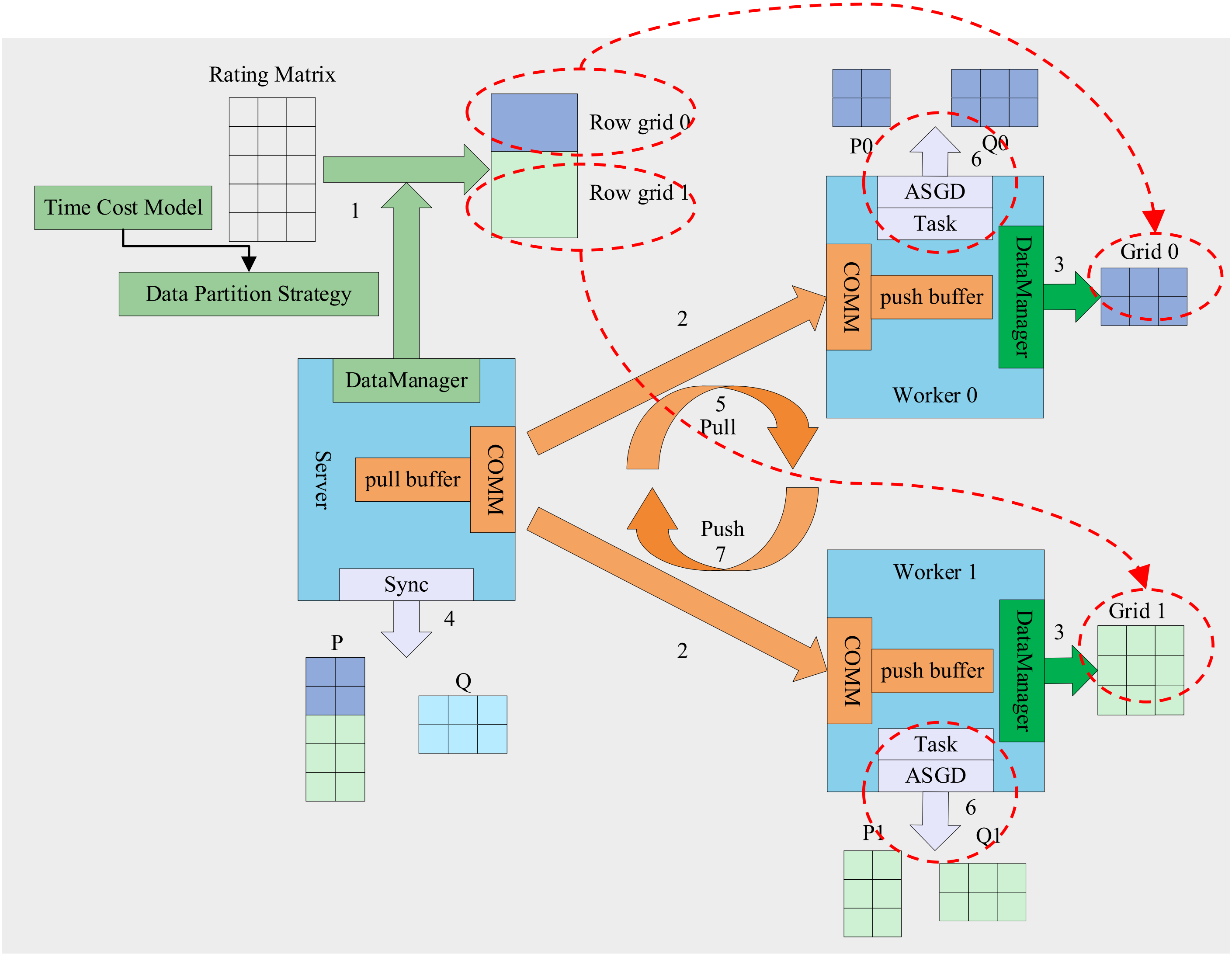
- **Two data partition strategies** to deal with different synchronization overhead conditions

Problem 3

How to optimize communication Inter-CPUs/GPUs ?

Communication optimization strategies that reduce the amount of data transmission and use computation to overlap communication

HCC-MF



- Heterogeneous CPUs/GPUs are abstracted into worker processes
- Use shared memory as a COMM channel between processes
- Server assigns data to workers, workers asynchronously calculate SGD-based MF
- Workers: Pull -> Computing -> Push
- Servers: Synchronization $\sum_{i=1}^{p}(P_i + Q_i)/p$

Our solution: HCC-MF

Problem 1

How to transparentize heterogeneous CPUs and GPUs

A general framework that unifies the abstraction and workflow

Problem 2

How to distribute data to each heterogeneous CPU/GPU to make the whole system more efficient ?

- A **time cost model** for guiding data Distribution.
- **Two data partition strategies** to deal with different synchronization overhead conditions

Problem 3

How to optimize communication Inter-CPUs/GPUs ?

Communication optimization strategies that reduce the amount of data transmission and use computation to overlap communication

Time Cost Model

- $T = max\{T_i\} + T_{sync}$

$P_i \gg B_i$

Omit performance-related components

$$T = max\left\{\frac{x_i nnz\,(16k+4)}{B_i} + \frac{2k(m+n)}{B_{bus\_i}}\right\} + \frac{3tk(m+n)}{B_{server}},$$

computing

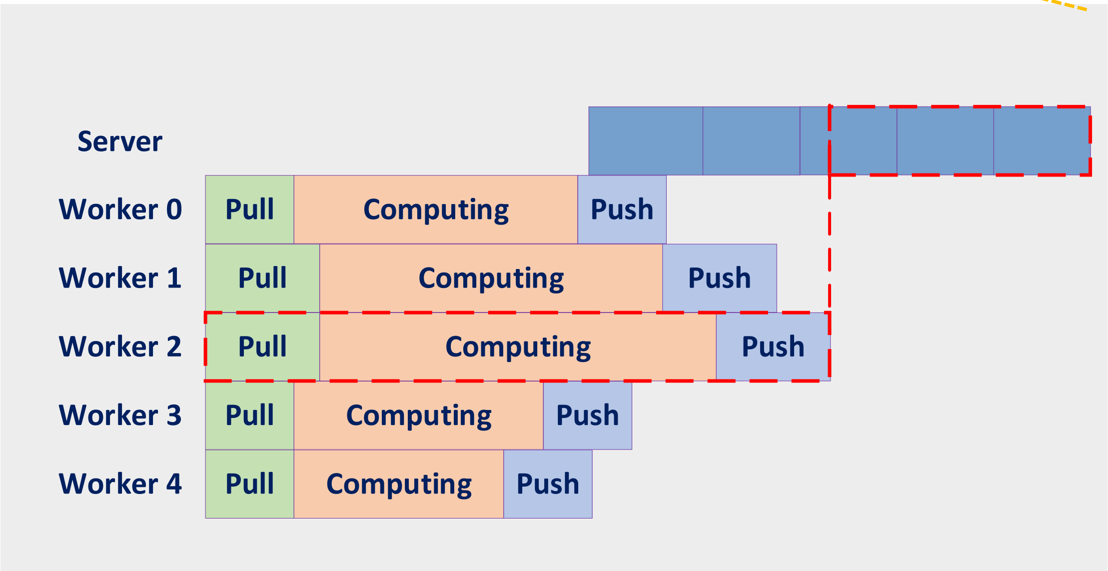
Pull-Push

Sync

Similar

Can sync be ignored ?

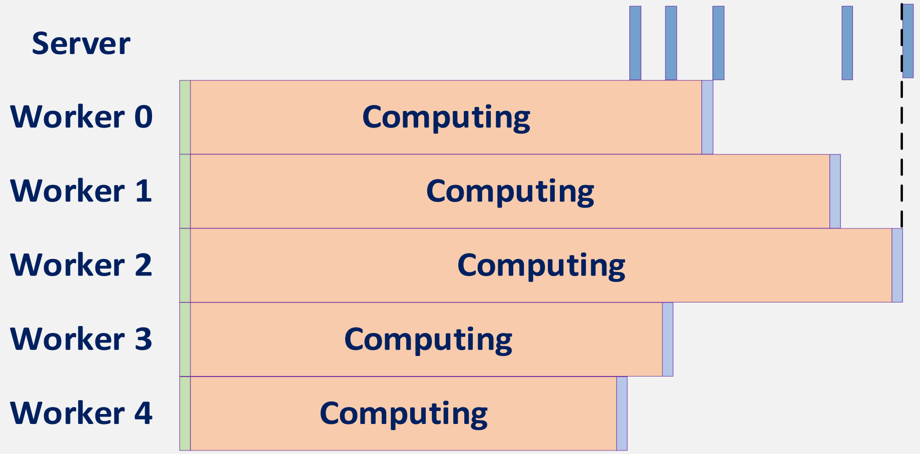
$$T = \begin{cases} max\,\{T_i(x_i)\} & max\,\{T_i\}\,/T_{sync} \geq \lambda, \\ max\,\{T_i(x_i)\} + T_{sync}(\mathrm{x}) & max\,\{T_i\}\,/T_{sync} < \lambda. \end{cases}$$



Worker $\boldsymbol{i}$

- computational complexity : $7kx_i nnz$
- \+
- memory access complexity : $(16k+4)x_i nnz$
- \+
- transmission complexity : $2k(m+n)$

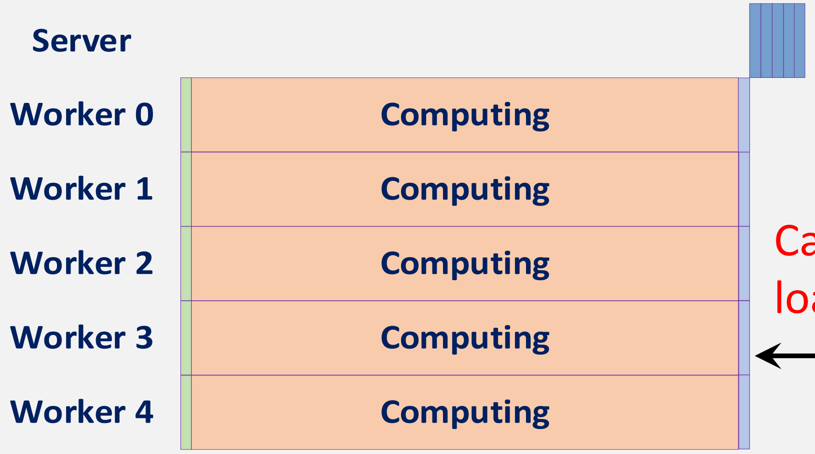
Data partition for load balance



$$\theta(\mathrm{x}) = min\{T\} = \min\left\{max\left\{\boxed{\frac{x_i nnz(16k+4)}{B_i}} + \boxed{\frac{2k(m+n)}{B_{bus\_i}}}\right\}\right\}$$

$$\theta(\mathrm{x}) = min\{max\{\boxed{A}x + \boxed{B}\}\}$$

Assuming B_i is a constant function of x_i



$a_1x_1 + b_1 = a_2x_2 + b_2 = \cdots = a_nx_n + b_n, \theta$ is the minimum

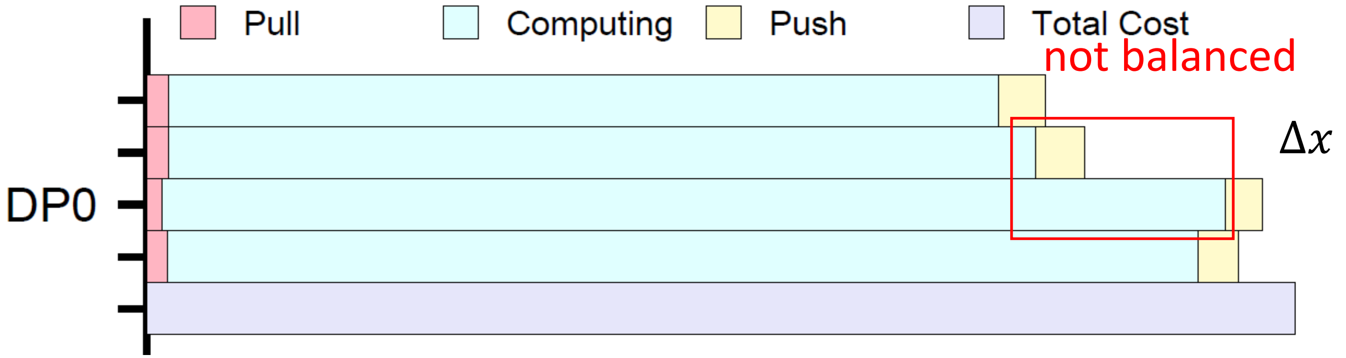
$$b_1 \approx b_2 \approx \cdots \approx b_n$$

Can DP0 really guarantee load balance?

$$\mathrm{DP_0}:\ x_i = \frac{1}{\sum_{j=1}^{p} \frac{a_i}{a_j}} = \frac{1}{\sum_{j=1}^{p} \frac{T_{i\_e}}{T_{j\_e}}}$$

Data partition for load balance

- The assumption of B_i is not true
- The Runtime performance may not be ignored



Differential → $if\ \Delta x\ is\ small, \Delta T\ can\ be\ regarded\ as\ linear$

Few iterations

DP0 → Algorithm 1 → DP1

Algorithm 1 Compensation algorithm

Input: Old data partition $\{x_{b\_1}, x_{b\_2}, ..., x_{b\_p}\}$; The computing time $\{t_1, t_2, ..., t_p\}$;

Output: New data partition $\{x_1, x_2, ..., x_p\}$

$T_{avg\_cpu} \leftarrow \frac{1}{c}\sum_{i=1}^{c} T_{i\_cpu}, T_{avg\_gpu} \leftarrow \frac{1}{g}\sum_{i=1}^{g} T_{i\_gpu}$
while $\frac{|T_{avg\_cpu} - T_{avg\_gpu}|}{min(T_{avg\_cpu}, T_{avg\_gpu})} > 0.1$ **do**
  $T_{avg\_cpu} > T_{avg\_gpu}\ ?\ l \leftarrow 1\ :\ l \leftarrow -1$
  $\Delta T \leftarrow \frac{l(T_{avg\_cpu} - T_{avg\_gpu})}{c+g}$
  for $i = 1 \rightarrow c$ **do**
    $x_{i\_cpu} \leftarrow \frac{x_{b\_i\_cpu}(t_{i\_cpu} - lg\Delta T)}{t_{i\_cpu}}$
  end for
  for $j = 1 \rightarrow g$ **do**
    $x_{j\_gpu} \leftarrow \frac{x_{b\_j\_gpu}(t_{j\_gpu} + lc\Delta T)}{t_{j\_gpu}}$
  end for
  $\{x_{b\_1}, x_{b\_2}, ..., x_{b\_p}\} \leftarrow \{x_{cpu}\} \cup \{x_{gpu}\}$
  $\{t_1, t_2, ..., t_p\} \leftarrow sgd\_update(\{x_{b\_1}, x_{b\_2}, ..., x_{b\_p}\})$
  $T_{avg\_cpu} \leftarrow \frac{1}{c}\sum_{i=1}^{c} T_{i\_cpu}, T_{avg\_gpu} \leftarrow \frac{1}{g}\sum_{i=1}^{g} T_{i\_gpu}$
end while
$\{x_1, x_2, ..., x_p\} \leftarrow \{x_{b\_1}, x_{b\_2}, ..., x_{b\_p}\}$
return $\{x_1, x_2, ..., x_p\}$

Data partition: hiding synchronization

$$T = \boxed{\max\left\{\frac{x_i nnz(16k+4)}{B_i} + \frac{2k(m+n)}{B_{bus_i}}\right\}} + \boxed{\frac{3tk(m+n)}{B_{server}}}$$

t is a nonlinear function of x

Difficult to solve the objective function

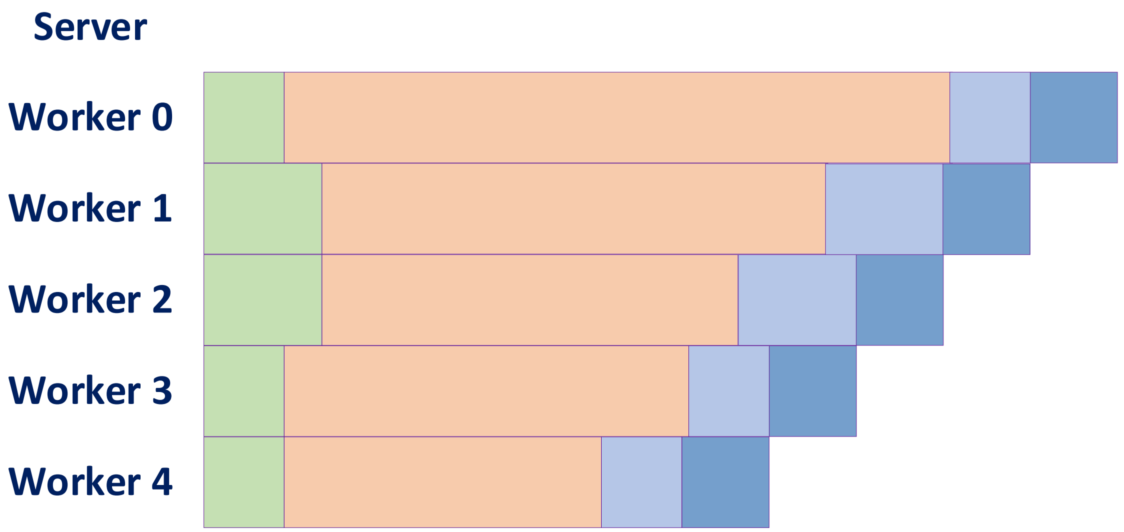
Use DP1 to balance the computational overhead of each worker

Use calculation to hide synchronization overheadc

$$\boxed{T_1 = T_2 = \cdots = T_n}$$

DP1-->DP2

$$\boxed{T_{(i \pm n)} = T_i \pm nT_{i\_sync}}$$



Our solution: HCC-MF

Problem 1

How to transparentize heterogeneous CPUs and GPUs

A general framework that unifies the abstraction and workflow

Problem 2

How to distribute data to each heterogeneous CPU/GPU to make the whole system more efficient ?

- A **time cost model** for guiding data Distribution.
- **Two data partition strategies** to deal with different synchronization overhead conditions

Problem 3

How to optimize communication Inter-CPUs/GPUs ?

Communication optimization strategies that reduce the amount of data transmission and use computation to overlap communication

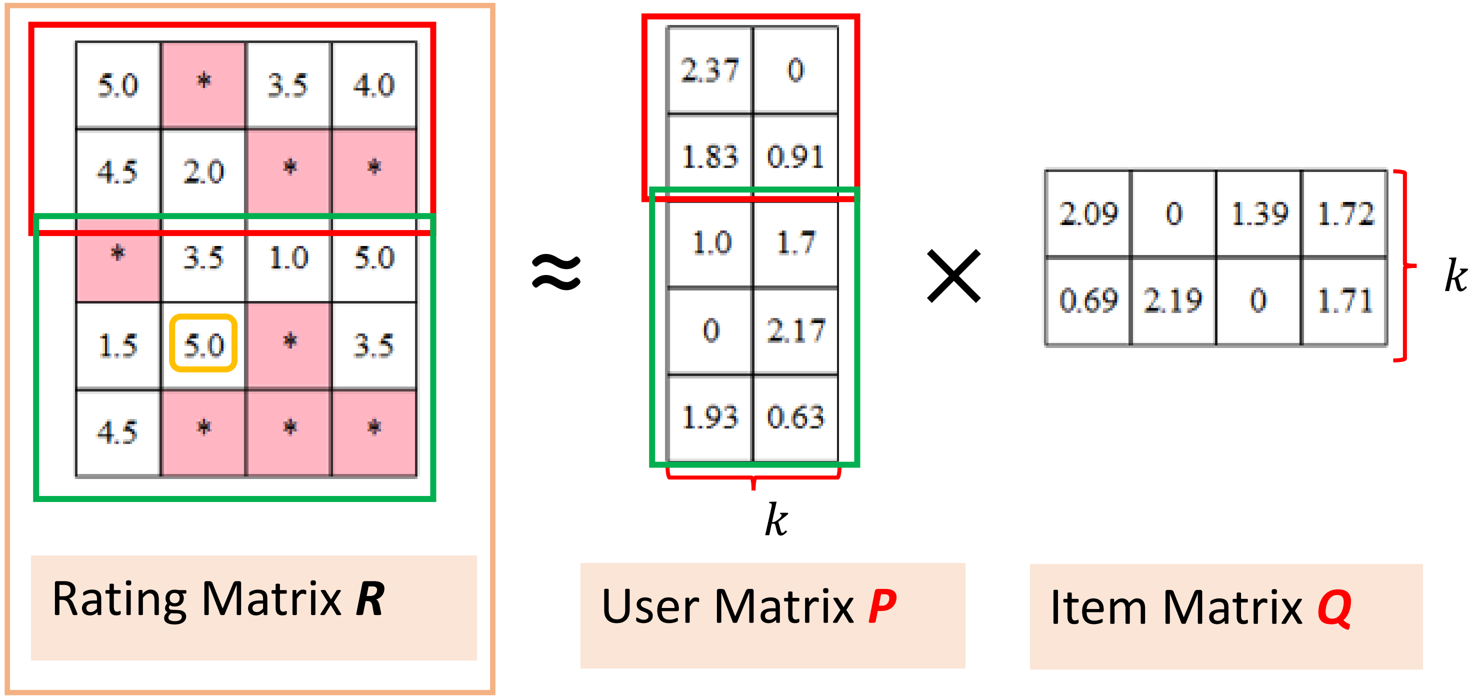
Reduce data transmission

Rows(columns) are independent of each other

Transmitting Q matrix only

The data range of the rating matrix is limited

Transmitting FP16 Data



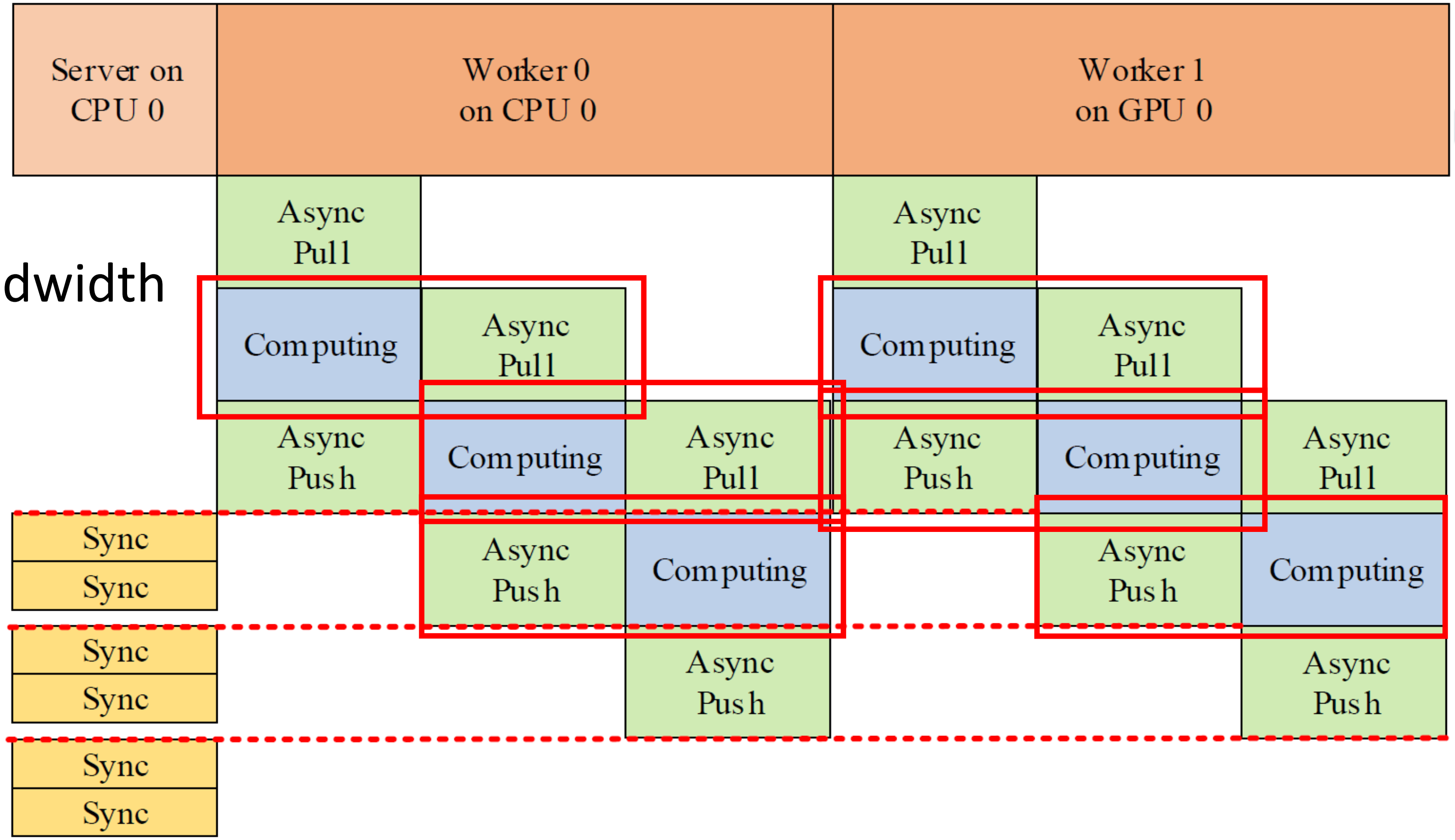
Overleap communication

Multiple Asynchronous computing-transmission streams in worker

GPU: copy engine

CPU: multithreads and free bandwidth

SoC: copy engine in iGPU



Outline

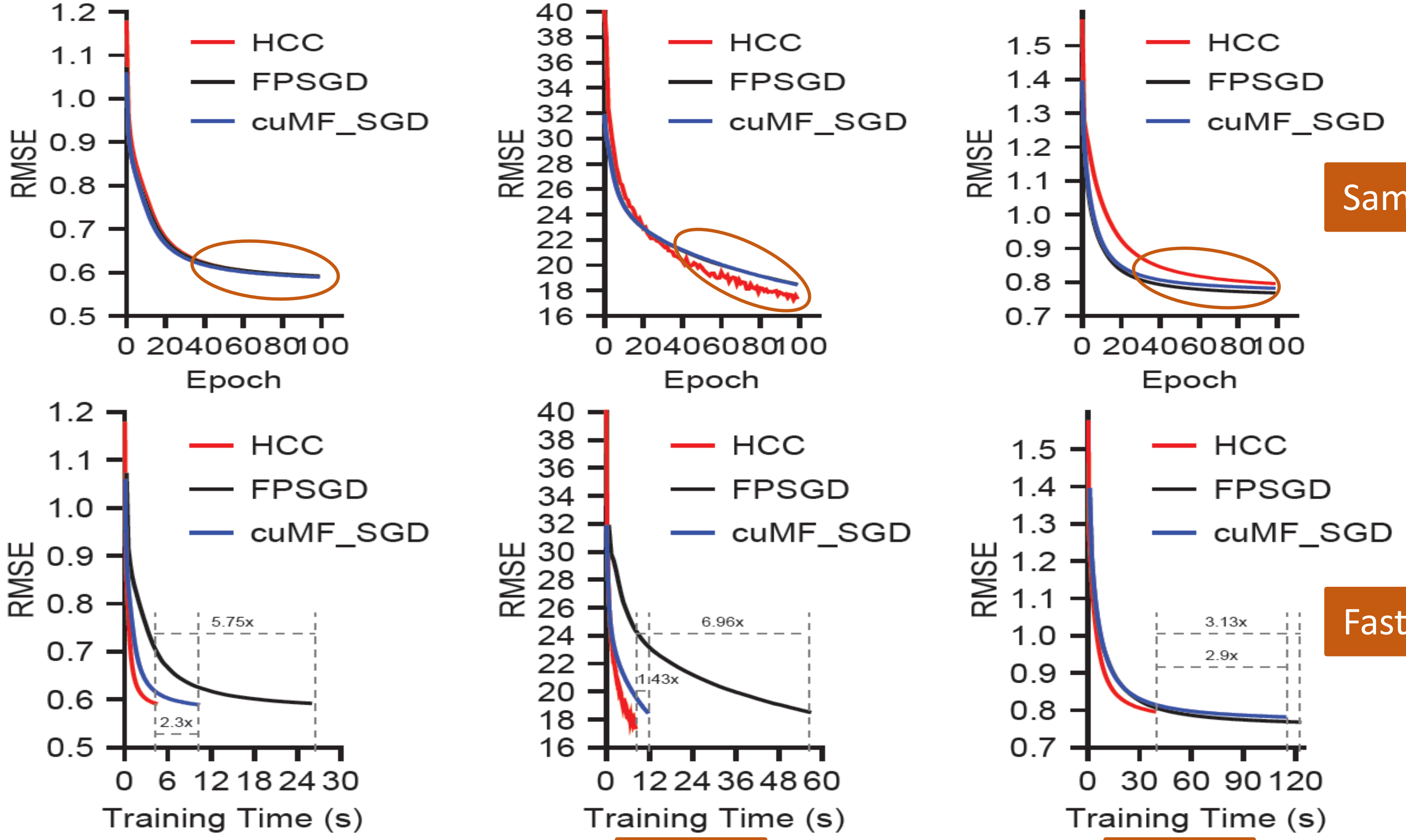
- Background and Motivation
- Design and Implementation
- Evaluation

Evaluation Setup

Item	Content
Hardware	2 Intel(R) Xeon(R) Gold 6242, Nvidia RTX 2080S, Nvidia Rtx 2080
DataSet	Netflix, Yahoo Music R1, R2, R1*, Movielens-20m
Baseline	FPSGD and cuMF_SGD we implemented

- We do not change the core idea of the baseline algorithm in our implementation
- We optimized the code to make the baseline execute faster
- We use baseline as the kernel running on the worker

Overall performance



Same convergence rate

Faster training speed

Netflix

R1

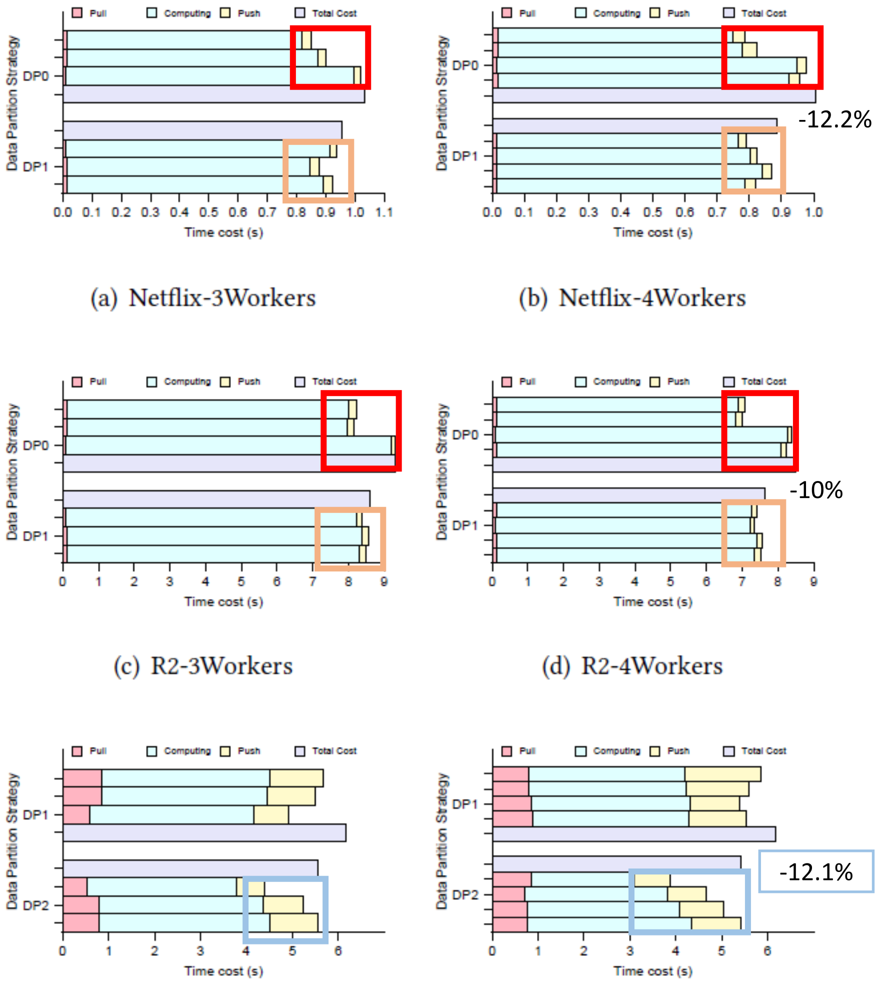
R2

Data partition evaluation

DP0 can only guarantee load balancing on similar processors

DP1 can guarantee load balance on all processors
- Netflix-4workers: -12.2%
- R2-4workers: -10%

DP2 can hide synchronization overhead
- R1*-4workers: -12.1%



(a) Netflix-3Workers (b) Netflix-4Workers

(c) R2-3Workers (d) R2-4Workers

(e) R1*-3Workers (f) R1*-4Workers

Communication optimization

		Netflix		R1_NEW		R2	
	Optimization	Cost time (s)	Speedup	Cost time (s)	Speedup	Cost time (s)	Speedup
COMM	P&Q	3.289744	1x	19.569929	1x	7.0763885	1x
	Q	0.180084684	18.3x	6.729931	2.9x	0.9467911	7.5x
	half-Q	0.056680425	58x	2.04014235	9.6x	0.31296455	22.6x
COMM-P	P&Q	21.8169325	1x	140.821585	1x	51.00871	1x
	Q	1.461305316	14.9x	50.57931	2.8x	7.190965	7.1x
	half-Q	0.53061025	41.1x	24.5123435	5.7x	4.039398	12.6x

Without any communication optimization, the communication overhead will offset the benefits brought by parallelism

Q can achieve better optimization results, but the effectiveness depends on the shape of the rating matrix

The transmission performance of half-q is more than twice that of Q

Conclusion

HCC-MF: A heterogeneous multi-CPU/GPU collaborative computing framework for SGD-based matrix factorization

- Unified workflow and transparent heterogeneous CPUs/GPUs usage
- Data distribution algorithm for different synchronization conditions
- Optimal inter-CPUs/GPUs communication

Limitation (Under study):

- Communication overhead can be further optimized
- Server bottleneck

Thank you

Yizhi Huang
huangyizhi @hnu.edu.cn