GPU-Oriented Operations Server for Sparse Matrix

GOOS-SM

長庚大學 林俊淵 助理教授
長庚大學 徐薇舒 研究生
清華大學 林郁翔 研究生
中正大學 劉奕志 研究生

Background

- Array operations are useful in a large number of important scientific codes, such as molecular dynamics, climate modeling, atmosphere, ocean sciences, and etc.
- To calculate the sparse matrix efficiently is a crucial issue (time) in many applications.

Motivation – (1/2)

- A data distribution scheme on the distributed memory multicomputers was the important research topic in the past.
- Data distribution scheme
  - Send Followed Compress (SFC)
  - Compress Followed Send (CFS)
  - Encoding-Decoding (ED)

Motivation – (2/2)

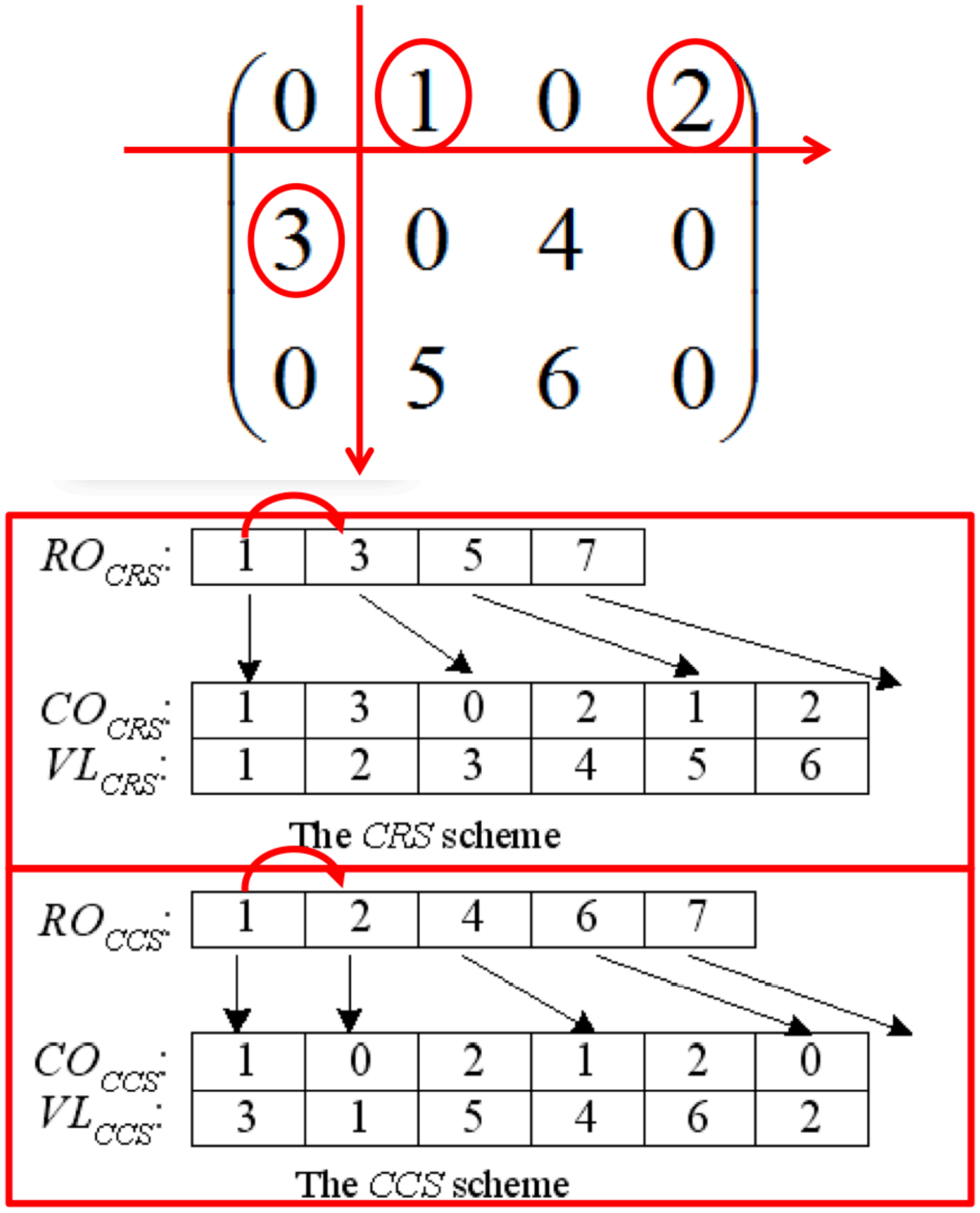
- Graphics Processing Unit (GPU) has become an attractive coprocessor for scientific computing due to its massive processing capability.
- There are several manuscripts about sparse matrix applications on the GPU have been published .

Goal

- Design the strategies for efficiently large amounts of compressing sparse matrices by using three data distribution schemes based on the GPU.
- The compressed sparse matrices in GPU can be queried for other matrix operations executing.

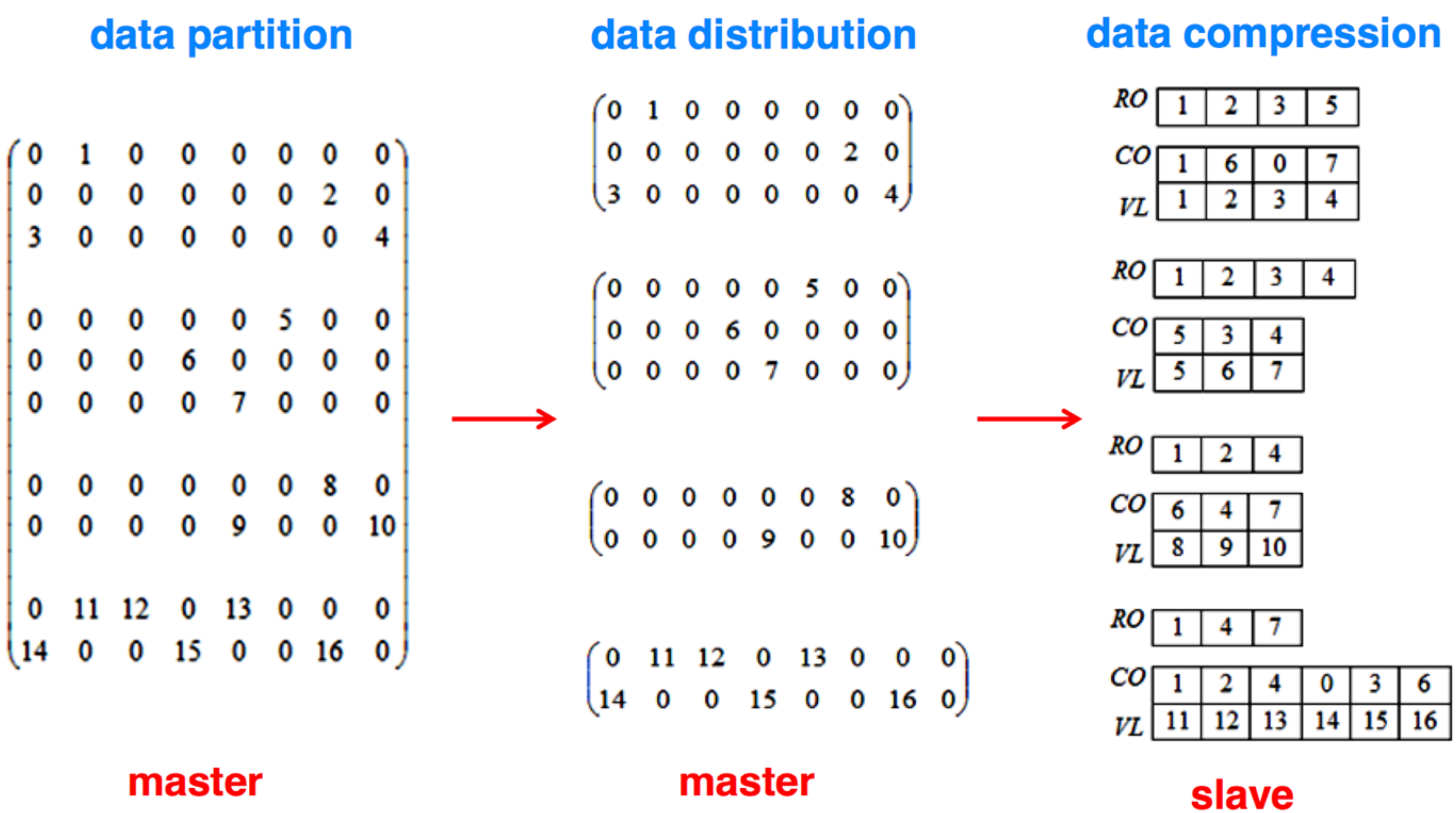
Preliminary Concepts - Data Compressing Storage

- Compressed Row/Column Storage(CRS/CCS)
  - RO stores the information of non-zero array elements of each row (column for CCS).
  - CO stores the column (row for CCS) indices of non-zero array elements of each row (column for CCS).
  - VL stores the values of non-zero array elements of the sparse array.



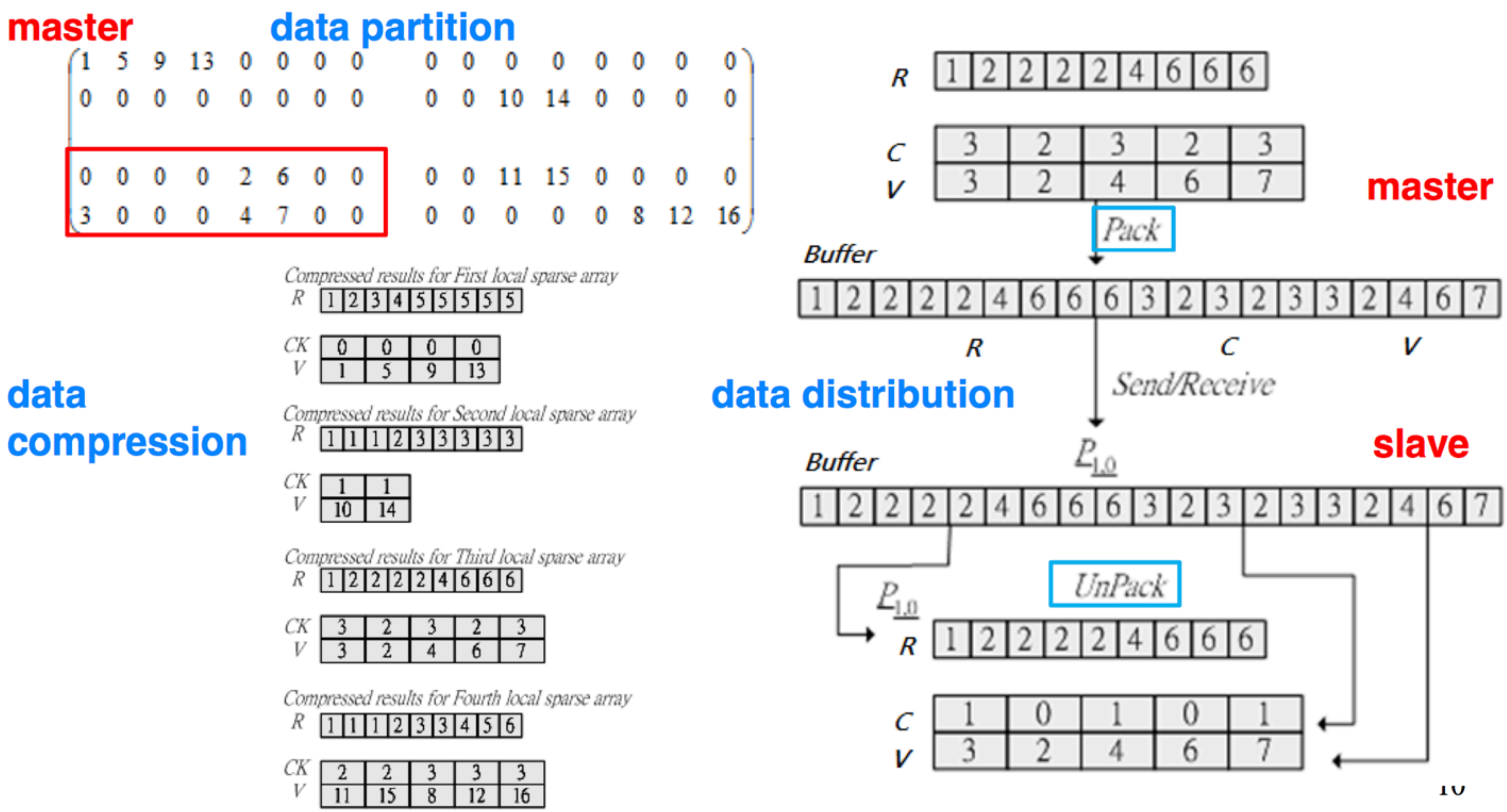
Preliminary Concepts - Data Compressing Storage

- Send Followed Compress (SFC)



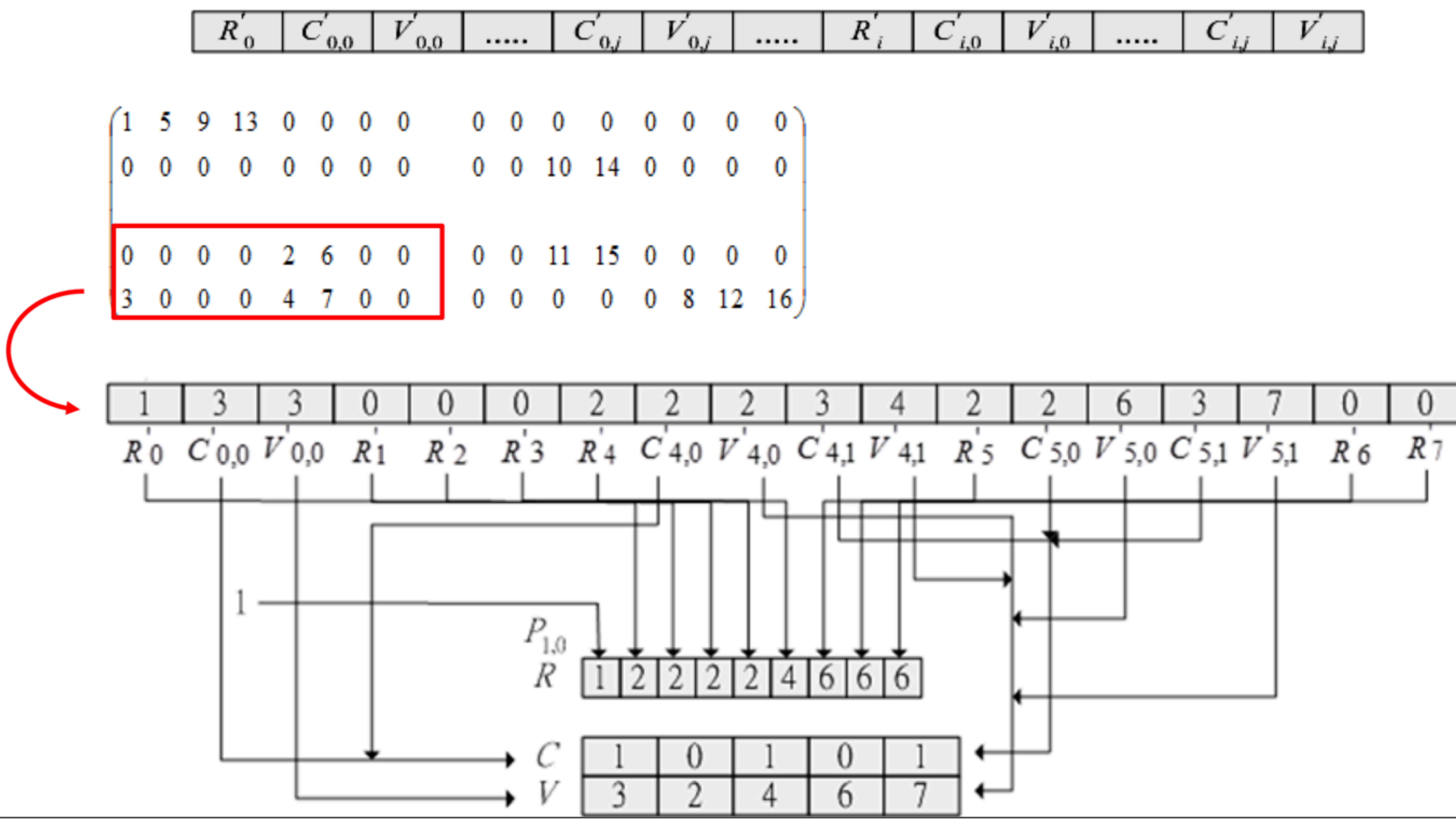
Preliminary Concepts - Data Compressing Storage

- Compress Followed Send (CFS)



Preliminary Concepts - Data Compressing Storage

- Encoding-Decoding(ED)



Sparse Matrix and CRS, CCS

Sparse Martrix

0	0	1	0	2	0	0	0	0
0	0	0	0	0	0	0	3	0
0	0	4	0	0	0	0	0	0
0	0	5	0	0	0	0	0	0
0	0	0	0	0	0	6	0	0
0	0	0	0	0	0	0	0	0
0	0	0	7	0	0	0	0	0
0	0	0	0	0	8	0	0	0
0	9	0	0	0	0	0	0	0

CRS

RO	1	3	4	5	6	7	7	8	9	10
CO	2	4	7	2	2	6	3	5	1	
VL	1	2	3	4	5	6	7	8	9	

CCS

RO	1	1	2	5	6	7	8	9	10	10
CO	8	0	2	3	6	0	7	4	1	
VL	9	1	4	5	7	2	8	6	3	

Data Dispatch

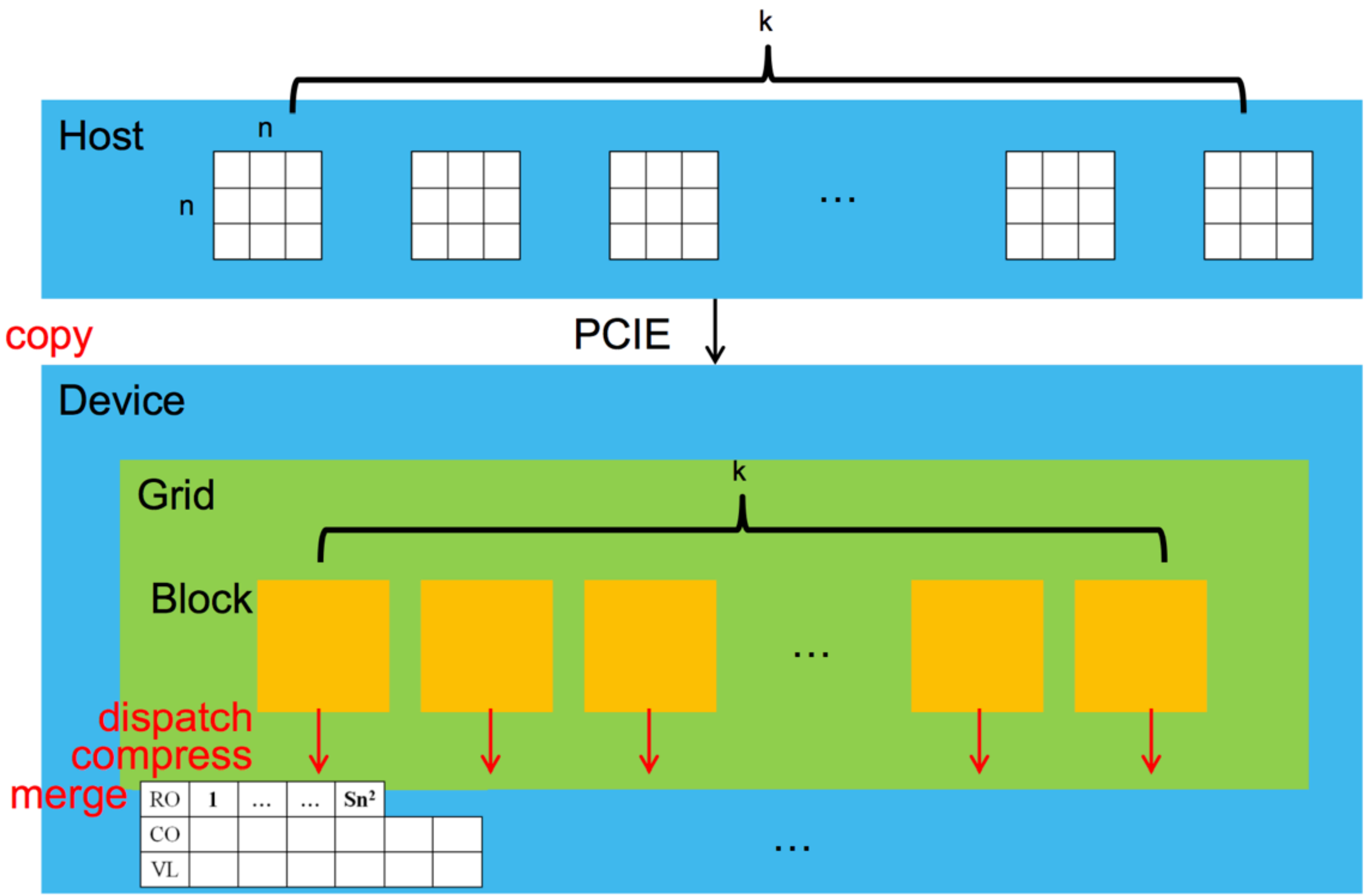
Row Dispatch

	0	0	1	0	2	0	0	0	0
T0	0	0	0	0	0	0	0	3	0
	0	0	4	0	0	0	0	0	0
	0	0	5	0	0	0	0	0	0
T1	0	0	0	0	0	0	6	0	0
	0	0	0	0	0	0	0	0	0
	0	0	0	7	0	0	0	0	0
T2	0	0	0	0	0	8	0	0	0
	0	9	0	0	0	0	0	0	0

Col Dispatch

	T0			T1			T2	
0	0	1	0	2	0	0	0	0
0	0	0	0	0	0	0	3	0
0	0	4	0	0	0	0	0	0
0	0	5	0	0	0	0	0	0
0	0	0	0	0	0	6	0	0
0	0	0	0	0	0	0	0	0
0	0	0	7	0	0	0	0	0
0	0	0	0	0	8	0	0	0
0	9	0	0	0	0	0	0	0

Send Followed Compress (SFC)



Parallelism scheme

- *Inter-task parallelization*
  - Each task is assigned to exactly one thread and dimBlock tasks are performed in parallel by different threads in a thread block.
- *Intra-task parallelization*
  - Each task is assigned to one thread block and all dimBlock threads in the thread block cooperate to perform the task in parallel.

Row dispatch, CRS compress

dispatch

ROW

0	0	1	0	2	0	0	0	0
0	0	0	0	0	0	0	3	0
0	0	4	0	0	0	0	0	0
0	0	5	0	0	0	0	0	0
0	0	0	0	0	0	6	0	0
0	0	0	0	0	0	0	0	0
0	0	0	7	0	0	0	0	0
0	0	0	0	0	8	0	0	0
0	9	0	0	0	0	0	0	0

compress

T0

RO	1	3	4	5
CO	2	4	7	2
VL	1	2	3	4

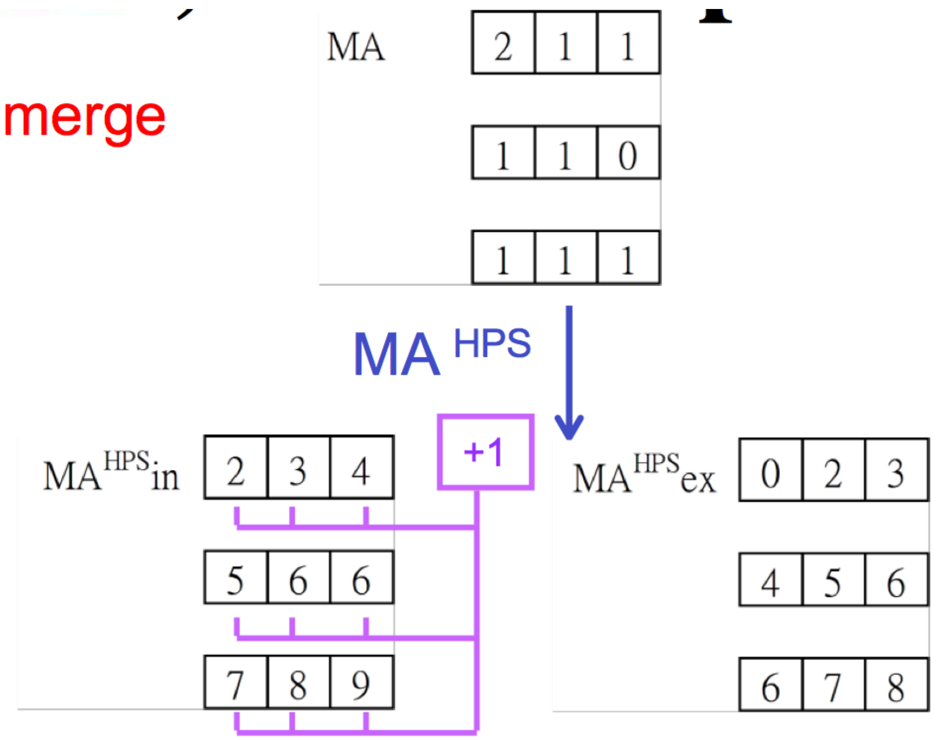
T1

RO	1	2	3	3
CO	2	6		
VL	5	6		

T2

RO	1	2	3	4
CO	1	4	5	
VL	7	8	9	

merge



change into RO

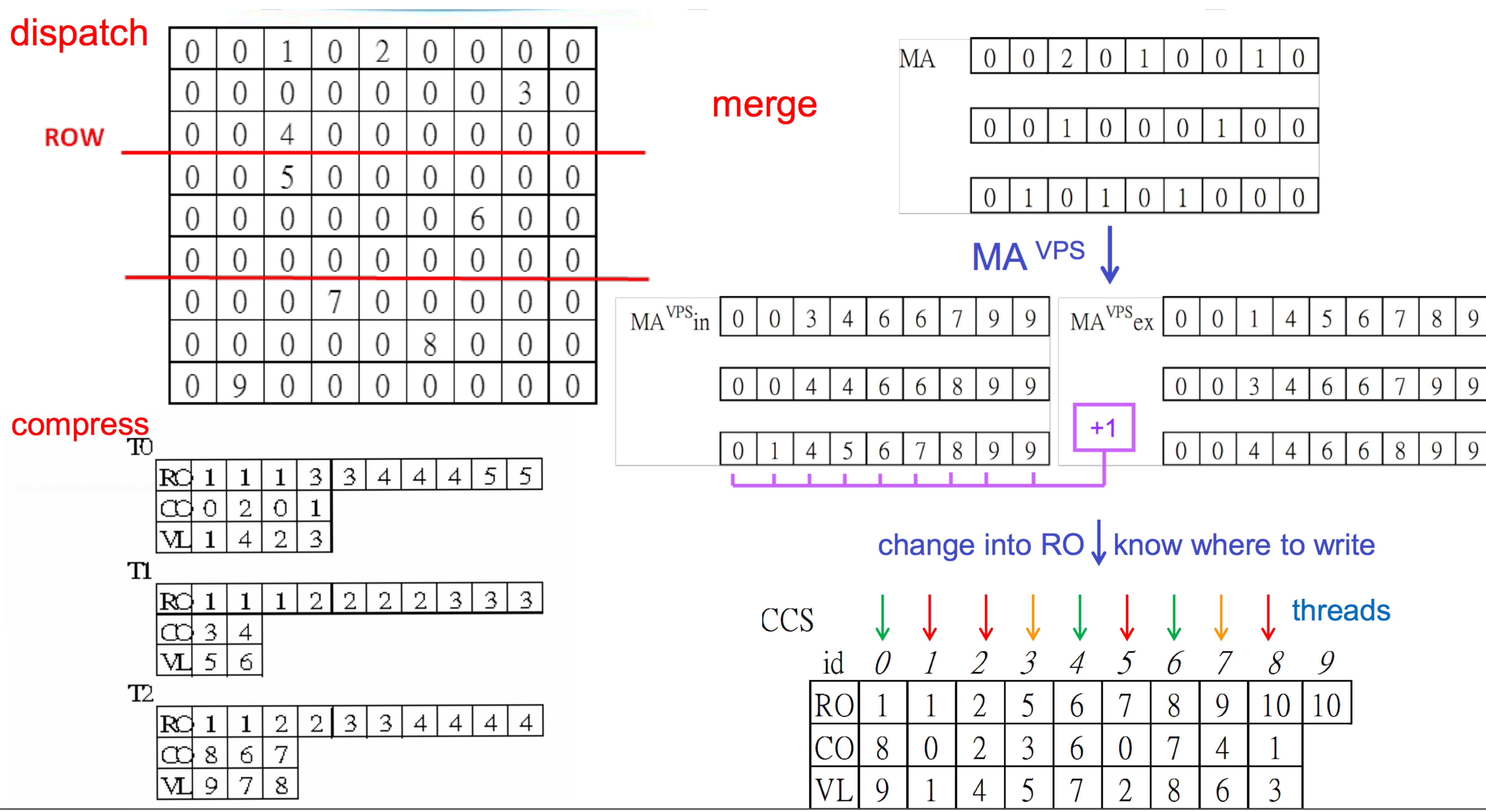
know where to write

CRS

threads

id	0	1	2	3	4	5	6	7	8	9
RO	1	3	4	5	6	7	7	8	9	10
CO	2	4	7	2	2	6	3	5	1	
VL	1	2	3	4	5	6	7	8	9	

Row dispatch, CCS compress

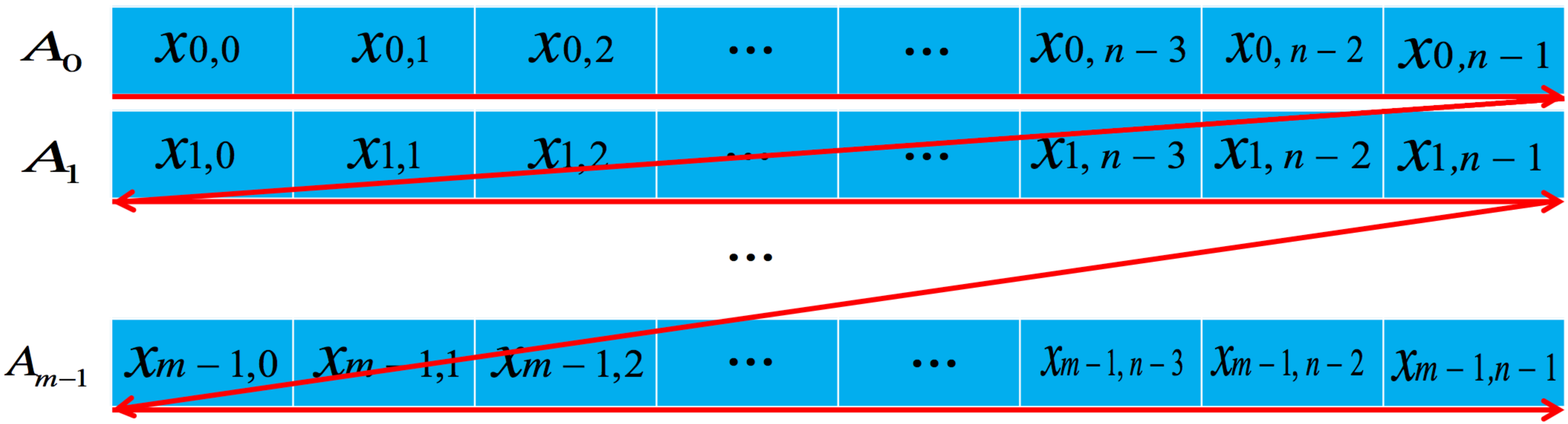


Tradition Prefix Sum (TPS)

- Definition:
  - A series with n elements， $A = [x_0, x_1, \ldots, x_{n-1}]$
  - Inclusive： $A_{in}^{TPS} = [x_0, (x_0 \oplus x_1), \ldots, (x_0 \oplus x_1 \oplus \ldots \oplus x_{n-1})]$
  - Exclusive： $A_{ex}^{TPS} = [I, x_0, (x_0 \oplus x_1), \ldots, (x_0 \oplus x_1 \oplus \ldots \oplus x_{n-2})]$
- Method: Work-Efficient Parallel Scan
- Example:

A	6	3	1	0	2	4	5	7
A^{TPS}_{in}	6	9	10	10	12	16	21	28
A^{TPS}_{ex}	0	6	9	10	10	12	16	21

Horizontal Prefix Sum (HPS)



Example for MA^{HPS}_{in}

Goal

A_0	6	3	1	0	2	4	5	7
A_1	1	7	4	6	2	5	3	0
A_2	3	0	5	4	6	1	7	2

Initial

A_0	6	3	1	0	2	4	5	7
A_1	1	7	4	6	2	5	3	0
A_2	3	0	5	4	6	1	7	2

$A_i{}^{TPS}{}_{in}$

$A_{0\,in}^{TPS}$	6	9	10	10	12	16	21	28
$A_{1\,in}^{TPS}$	1	8	12	18	20	25	28	28
$A_{2\,in}^{TPS}$	3	3	8	12	18	19	26	28

$XMA^{TPS}{}_{ex}$

XMA	XMA^{TPS}_{ex}
28	0
28	28
28	56

Update

add

$A_{0\,in}^{TPS}$	6	9	10	10	12	16	21	28
$A_{1\,in}^{TPS}$	1	8	12	18	20	25	28	28
$A_{2\,in}^{TPS}$	3	3	8	12	18	19	26	28

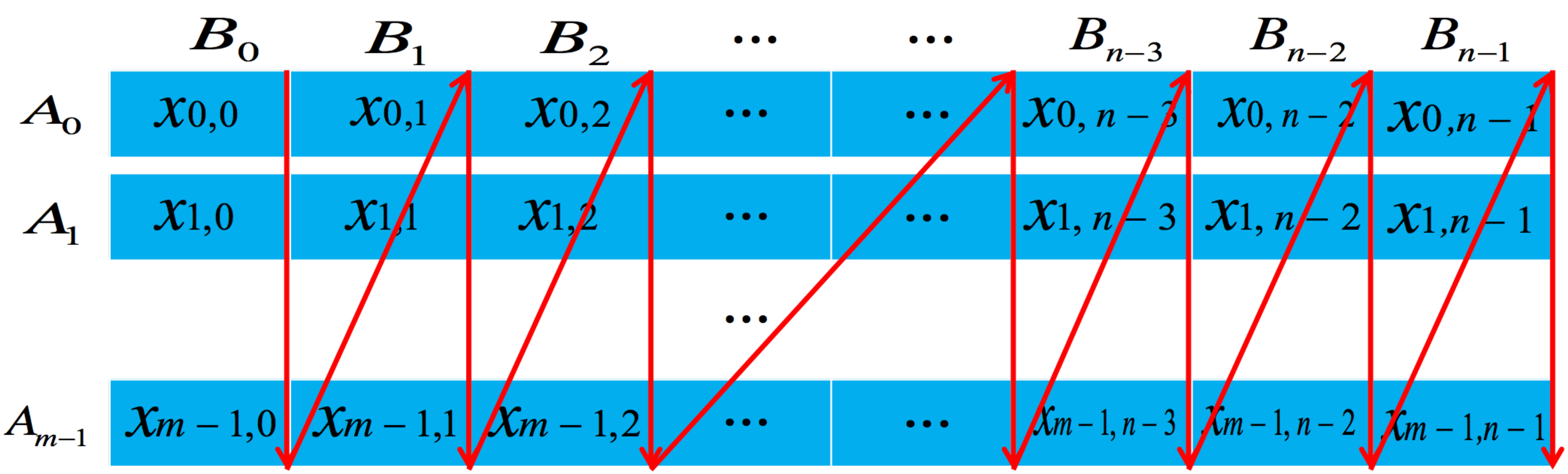
XMA^{TPS}_{ex}
0
28
56

Result

MA^{HPS}_{in}	6	9	10	10	12	16	21	28
	29	36	40	46	48	53	56	56
	59	59	64	68	74	75	82	84

Vertical Prefix Sum (VPS)

- There are m series, each with n elements, and we need to combine them by prefix sum with vertical direction.



Example for MA^{VPS}_{in}

Initial

	B_0	B_1	B_2	B_3	B_4	B_5	B_6	B_7
A_0	6	3	1	0	2	4	5	7
A_1	1	7	4	6	2	5	3	0
A_2	3	0	5	4	6	1	7	2

$B_i{}^{TPS}{}_{in}$

B_{0in}^{TPS}	B_{1in}^{TPS}	B_{2in}^{TPS}	B_{3in}^{TPS}	B_{4in}^{TPS}	B_{5in}^{TPS}	B_{6in}^{TPS}	B_{7in}^{TPS}
6	3	1	0	2	4	5	7
7	10	5	6	4	9	8	7
10	10	10	10	10	10	15	9

Goal

	B_0	B_1	B_2	B_3	B_4	B_5	B_6	B_7
A_0	6	3	1	0	2	4	5	7
A_1	1	7	4	6	2	5	3	0
A_2	3	0	5	4	6	1	7	2

$XMB^{TPS}{}_{ex}$

XMB	10	10	10	10	10	10	15	9
XMB^{TPS}_{ex}	0	10	20	30	40	50	60	75

Update

	B_{0in}^{TPS}	B_{1in}^{TPS}	B_{2in}^{TPS}	B_{3in}^{TPS}	B_{4in}^{TPS}	B_{5in}^{TPS}	B_{6in}^{TPS}	B_{7in}^{TPS}
	6	3	1	0	2	4	5	7
	7	10	5	6	4	9	8	7
	10	10	10	10	10	10	15	9
add								
XMB^{TPS}_{ex}	0	10	20	30	40	50	60	75

Result

MA^{VPS}_{in}	6	13	21	30	42	54	65	82
	7	20	25	36	44	59	68	82
	10	20	30	40	50	60	75	84

SFC without prefix sum

dispatch

0	0	1	0	2	0	0	0	0
0	0	0	0	0	0	0	3	0
0	0	4	0	0	0	0	0	0
0	0	5	0	0	0	0	0	0
0	0	0	0	0	0	6	0	0
0	0	0	0	0	0	0	0	0
0	0	0	7	0	0	0	0	0
0	0	0	0	0	8	0	0	0
0	9	0	0	0	0	0	0	0

compress

TO

RO	0	1	3	
CO	1	2	2	2
VL	9	1	4	5

T1

RO	1	1	1
CO	3	4	5
VL	7	2	8

T2

RO	1	1	0
CO	6	7	
VL	6	3	

decoding

CCS

threads

RO	0	1	3	1	1	1	1	1	0

CO	8	0	2	3	6	0	7	4	1
VL	9	1	4	5	7	2	8	6	3

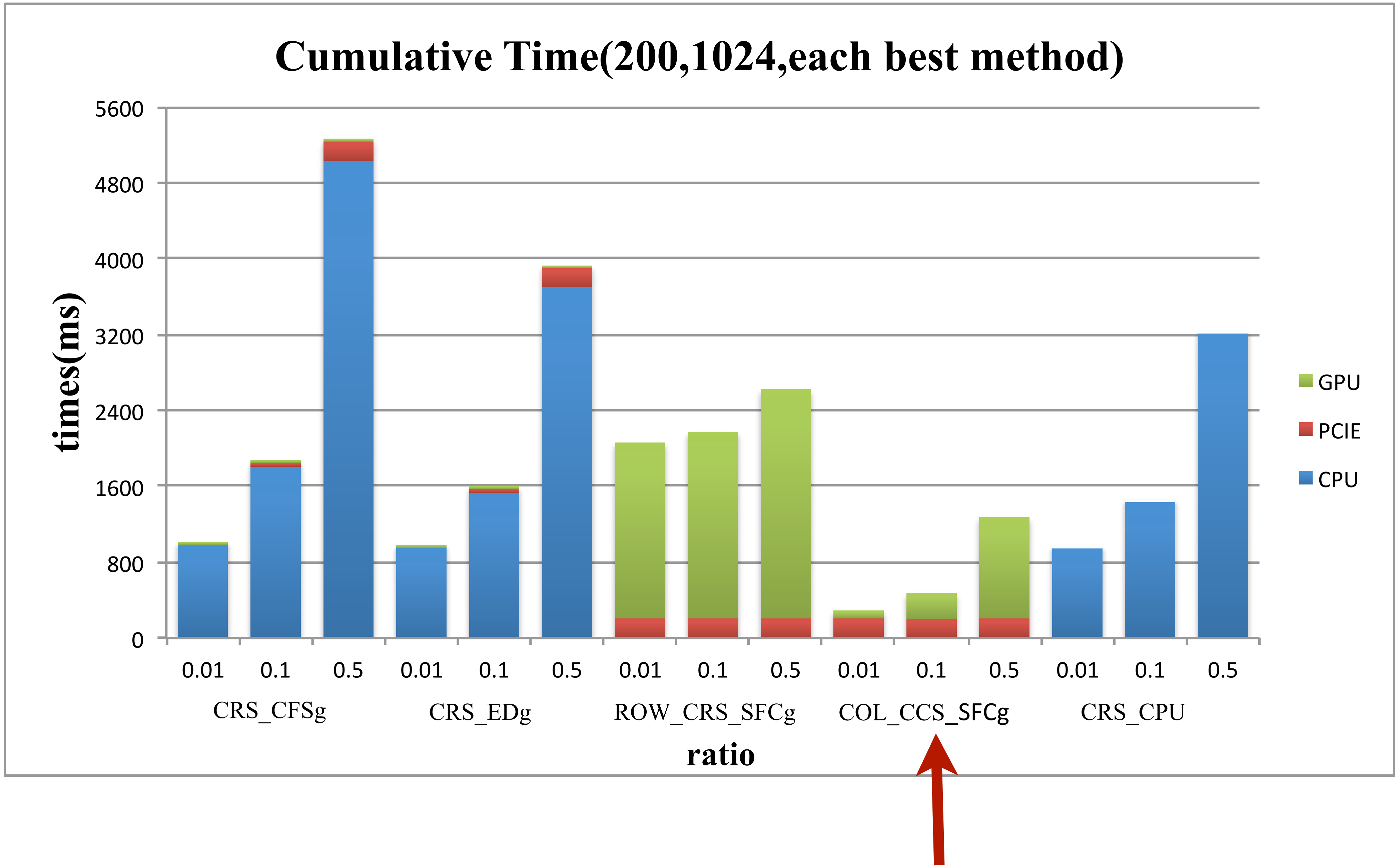
Multi-GPUs

- Type 1
  - First come first serve. If out of memory of single GPU, then split half of input data to another GPU.
- Type 2
  - Load balancing with global memory size.

Previous Test

Test Platform	
OS	Ubuntu 10.04
CPU	Intel E5506 @ 2.13GHz-2core
Memory	12 GB RAM
GPU	Tesla C2050 @ 1.15 GHz-448 core
CUDA Version	2.0

Test Data	
Amount	50、100、150、200
Ratio	0.01、0.1、0.5
Size	128x128、256x256、512x512、1024x1024



SFC is better strategy in GPU

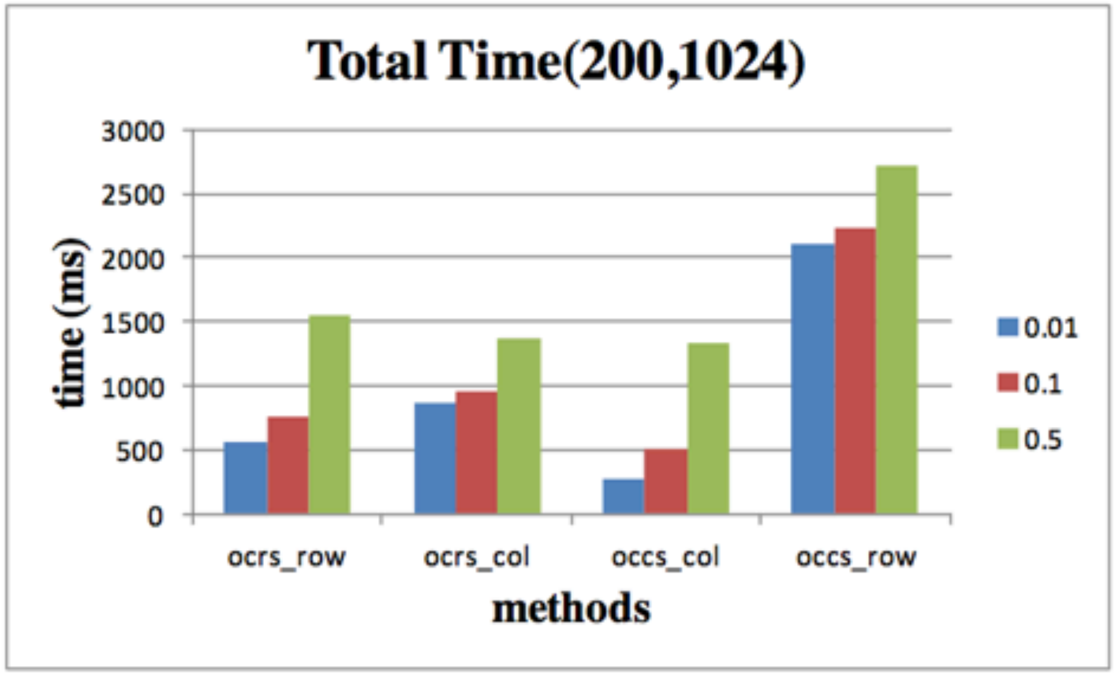
Now in Formosa 5

- 39 computing nodes. Each computing node consists of:
    - CPU - Intel Xeon x5670 six cores 2.93GHz x2
    - Main Memory - 96GB
    - Hard Disk - 120GB SSD
    - Network - 4x QDR (40Gb) InfiniBand HCA
    - GPU - NVIDIA TESLA M2070 x3

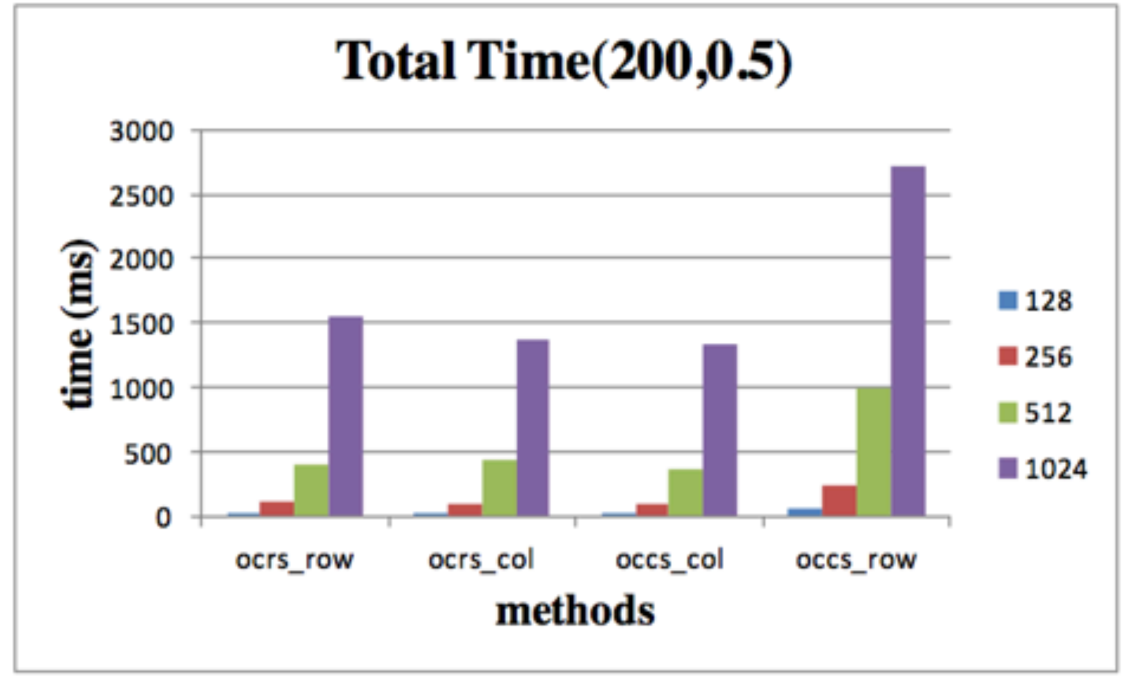
Test Data

Test Data	
Amount	50、100、150、200、400、600、800、1000、1200、2000、3000、4000、5000、6000
Ratio	0.01、0.1、0.5
Size	128x128、256x256、512x512、1024x1024

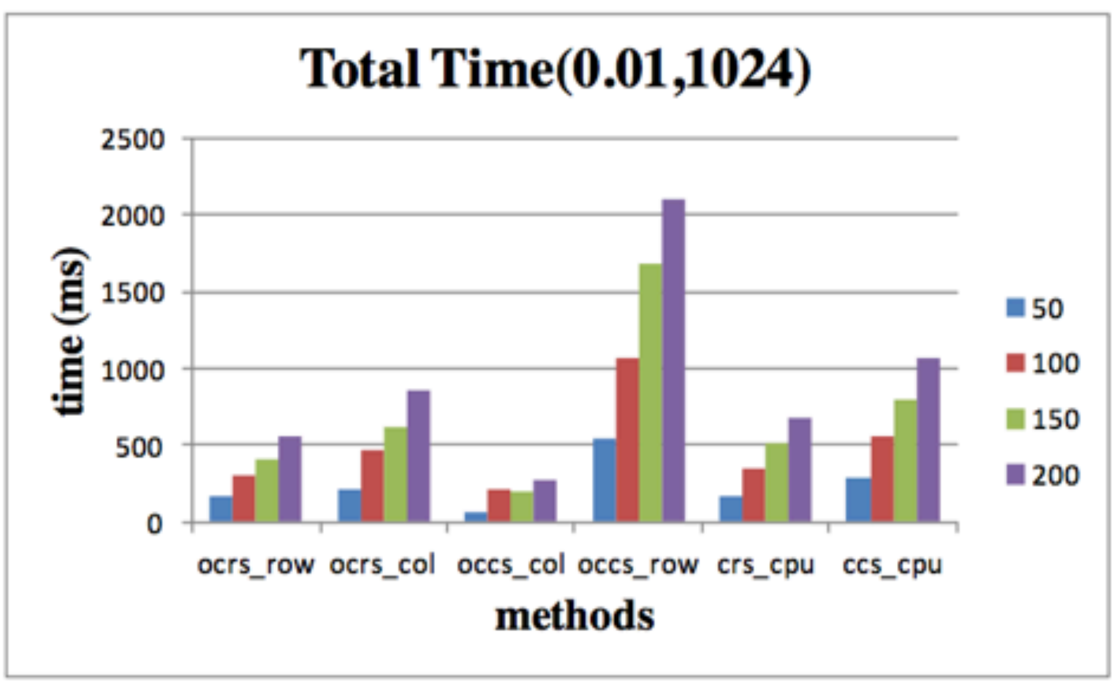
The experimental results of SFC.



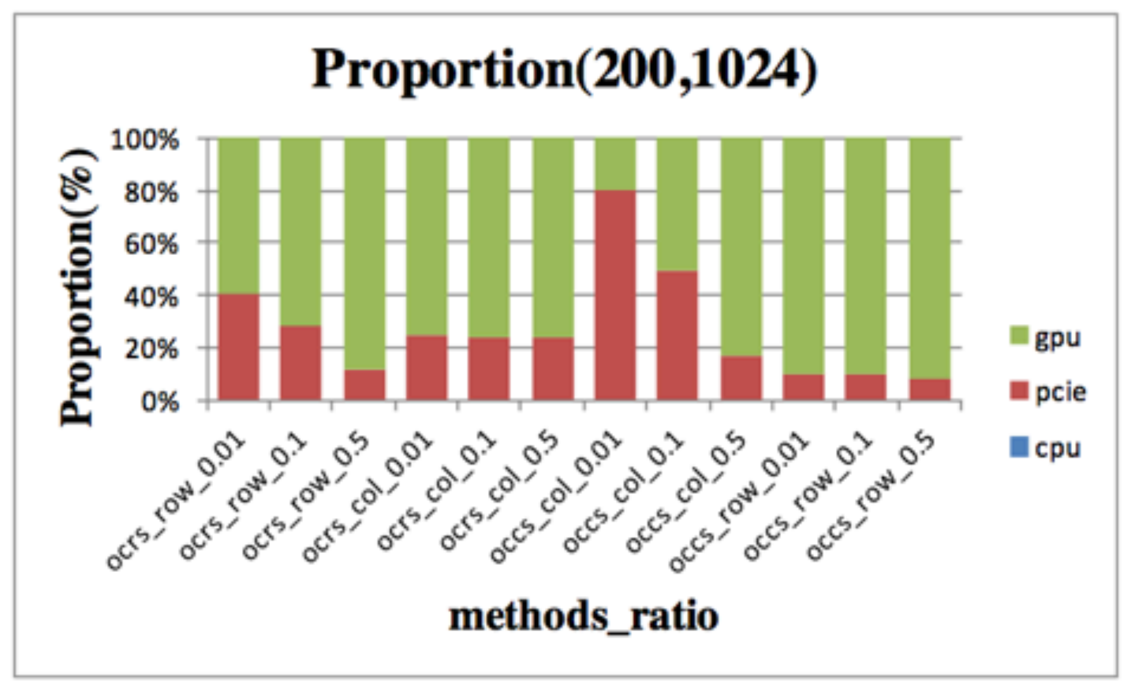
(a) The total time of the different ration based on amounts 200, 1024×1024.



(b) The total time of the different size based on amounts 200, ration 0.5.

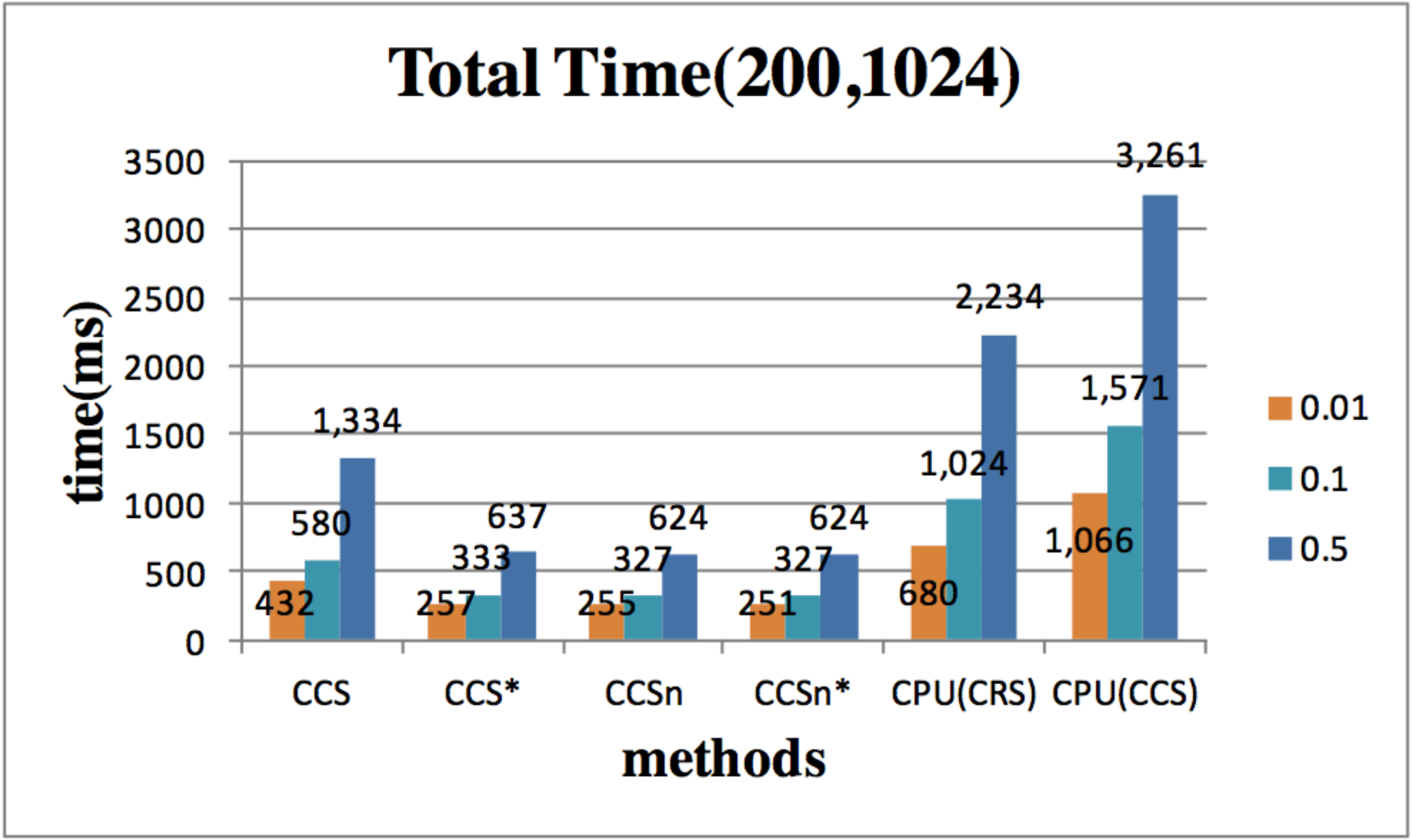


(c) The total time of the different amounts based on ration 0.01, 1024×1024.

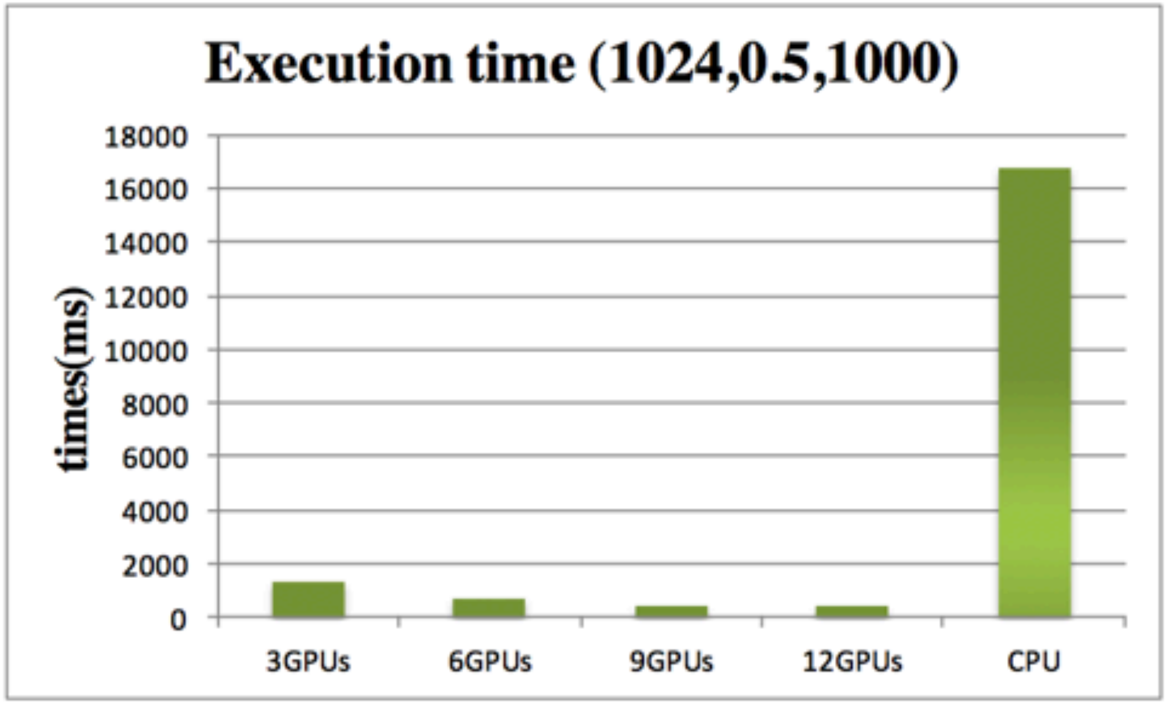


(d) The proportion of the GPU computing, memory copy, and GPU computing phase.

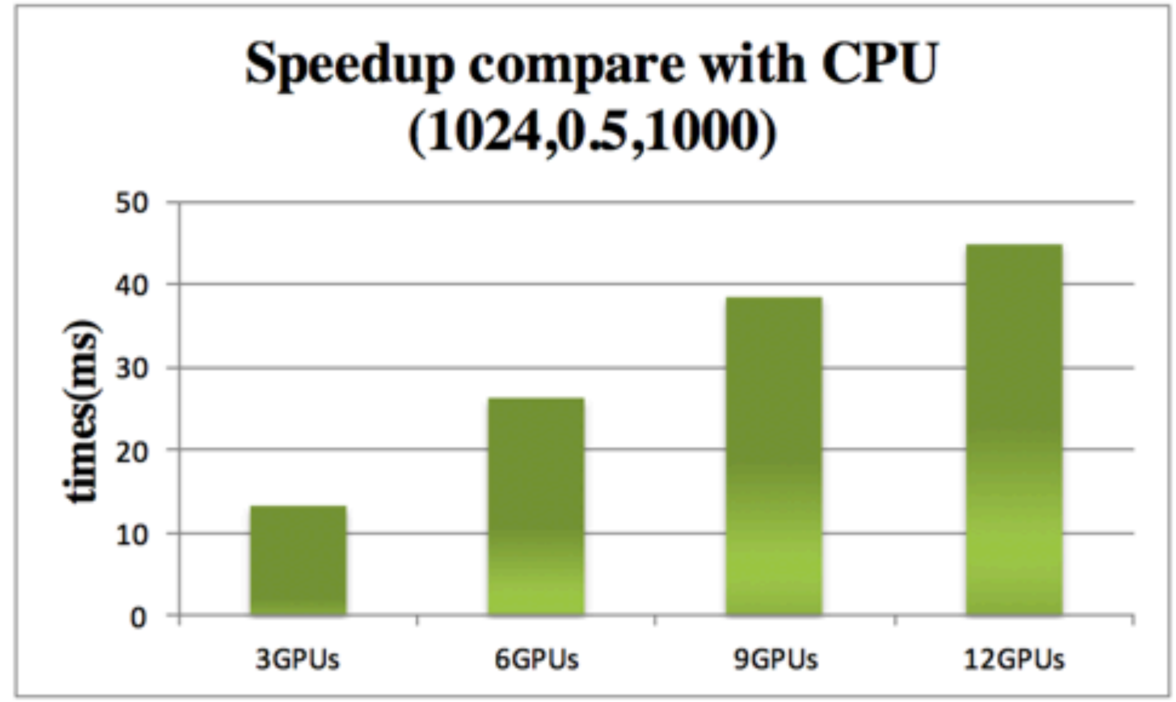
The execute time of different methods with ratio 0.01, 0.1, 0.5.



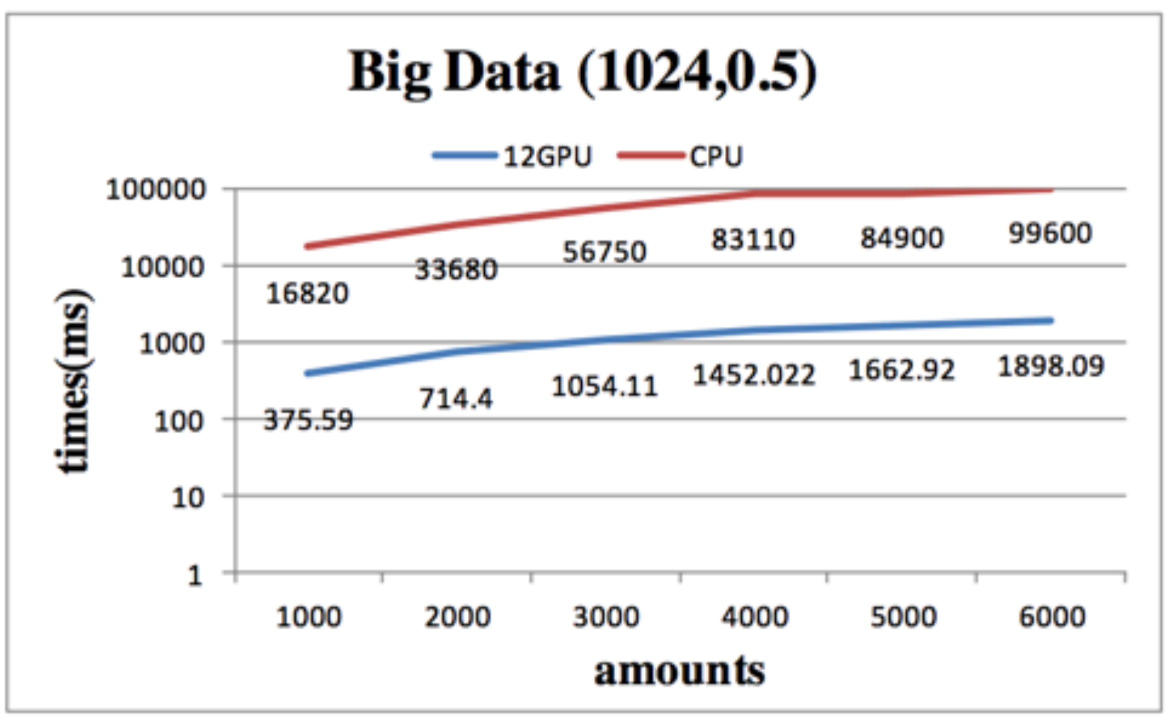
Testing multi-GPUs.



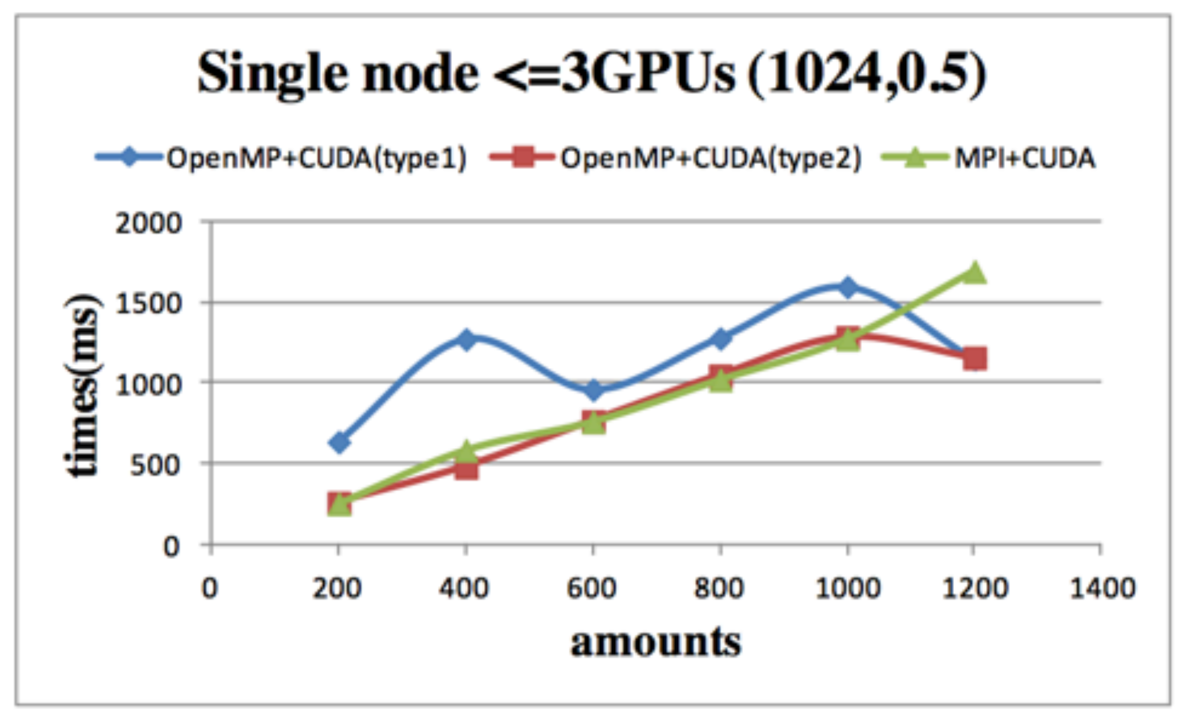
(a) The execute time of various number of GPUs.



(b) The speedups compared with CPU version.



(c) The execute time of different amounts data with 12 GPUs and CPU.



(d) The execute time of different amounts data with different multi-GPUs computing

Conclusion

- Optimal techniques using CUDA
  - Intra-task parallelization
  - Work-Efficient Parallel Scan
  - Avoiding Bank Conflicts
  - Coalescing
  - Cache Configurable
- Using 12 GPUs, speedup can up to 55x
  - 6000, 1024*1024 sparse matrix, total 23.4GB