

稀疏线性代数子程序的性能优化

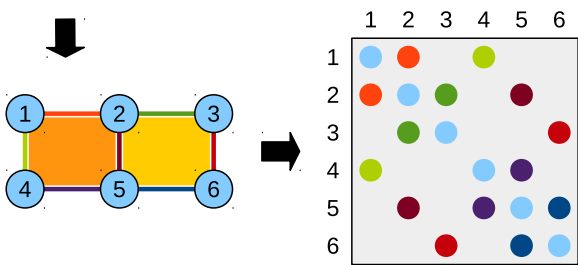
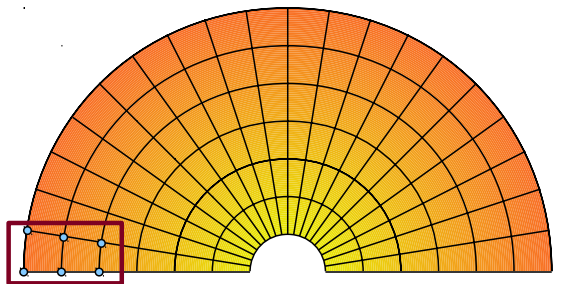
Weifeng Liu (刘伟峰)¹, Brian Vinter², Hao Wang³, Kaixi Hou³, Wu-chun Feng³, Ang Li⁴, Jonathan Hogg⁵, Iain Duff⁵, Guangming Tan⁶, Zhenya Zhou⁷, Yuyao Niu¹, Zhengyang Lu¹, Meichen Dong¹, Zhou Jin¹, Haonan Ji¹, Shuhui Song¹, Huimin Song¹, Yuchen Luo¹, Jianqi Zhao¹, Yao Wen¹, Yuechen Lu¹, Xu Fu¹, Bingbin Zhang¹, Jingwen Zhang¹, Tengcheng Wang¹, Yuying Sun¹, Fangying Li¹, Jiayi Wu¹, Zhengyu Qu¹, Dechuang Yang¹, Tianyu Zhao¹, Xiaohan Geng¹, Li Zuo¹, Shibo Lu⁸.

(1) 中国石油大学（北京）超级科学软件实验室（SSSLab）, (2) University of Copenhagen, (3) Virginia Tech, (4) Eindhoven University of Technology, (5) Rutherford Appleton Laboratory, (6) ICT, CAS, (7) Huada Emphyrean Software Co. Ltd., (8) Northeastern University.

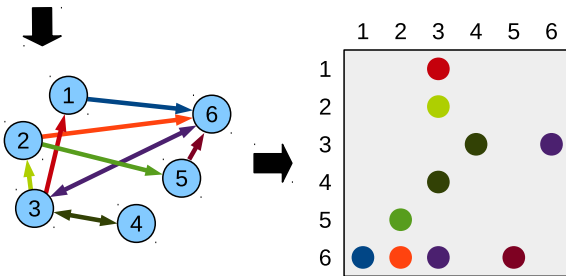
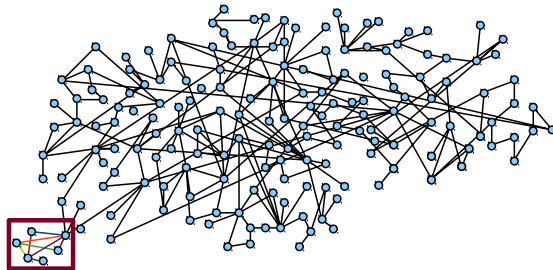
稀疏矩阵

Sparse Matrix

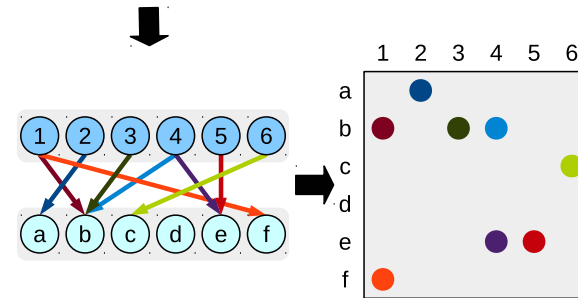
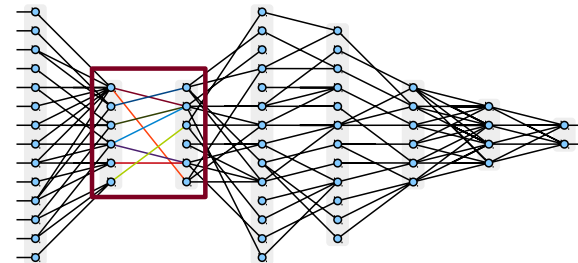
稀疏矩阵无处不在



有限元网格,
计算科学

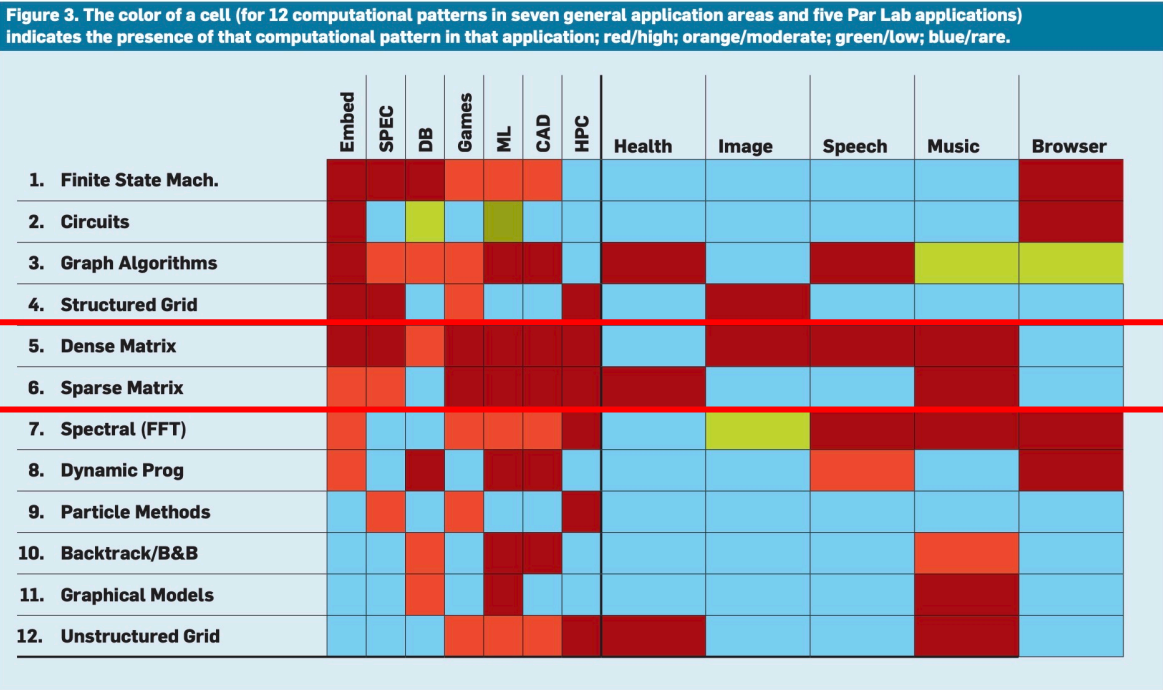


社交网络,
大数据分析



深度神经网络,
人工智能

稀疏矩阵对于超算应用十分重要



Krste Asanovic, Rastislav Bodik, James Demmel, Tony Keaveny, Kurt Keutzer, John Kubiawicz, Nelson Morgan, David Patterson, Koushik Sen, John Wawrzynek, David Wessel, and Katherine Yelick. 2009. A view of the parallel computing landscape. Commun. ACM 52, 10 (October 2009), 56–67.

稀疏矩阵对于超算应用十分重要

Science Area	Number of Teams	Codes	Struct Grids	Unstruct Grids	Dense Matrix	Sparse Matrix	N-Body	Monte Carlo	FFT	PIC	Sig I/O
Climate and Weather	3	CESM, GCRM, CM1/WRF, HOMME	X	X		X		X			X
Plasmas/Magnetosphere	2	H3D(M), VPIC, OSIRIS, Magtail/UPIC	X				X		X		X
Stellar Atmospheres and Supernovae	5	PPM, MAESTRO, CASTRO, SEDONA, ChaNGa, MS-FLUKSS	X			X	X	X		X	X
Cosmology	2	Enzo, pGADGET	X			X	X				
Combustion/Turbulence	2	PSDNS, DISTUF	X						X		
General Relativity	2	Cactus, Harm3D, LazEV	X			X					
Molecular Dynamics	4	AMBER, Gromacs, NAMD, LAMMPS				X	X		X		
Quantum Chemistry	2	SIAL, GAMESS, NWChem			X	X	X	X			X
Material Science	3	NEMOS, OMEN, GW, QMCPACK			X	X	X	X			
Earthquakes/Seismology	2	AWP-ODC, HERCULES, PLSQR, SPECFEM3D	X	X			X				X
Quantum Chromo Dynamics	1	Chroma, MILC, USQCD	X		X	X					
Social Networks	1	EPISIMDEMICS									
Evolution	1	Eve									
Engineering/System of Systems	1	GRIPS, Revisit						X			
Computer Science	1			X	X	X			X		X

美国National Center for Supercomputing Applications (NCSA)统计数据, 该列表来自于UIUC的Wen-mei W. Hwu教授

超级计算机的排名方式一：LU分解求解稠密 $Ax=b$

12	10	1	4
2	3	9	5
16	6	5	8
4	3	15	6

稠密矩阵A

=

1	0	0	0
1/6	1	0	0
4/3	-11/2	1	0
1/3	-1/4	135/418	1

下三角矩阵L

x

12	10	1	4
0	4/3	53/6	13/3
0	0	209/4	53/2
0	0	0	-587/209

上三角矩阵U

求解 $Ax=b$ 步骤:

- 1) $A=LU$
- 2) $LUx=b$
- 3) $Ly=b$
- 4) $Ux=y$

测试包Linpack使用以矩阵LU分解为基础的方法求解稠密 $Ax=b$,
记录计算效率以评测超级计算机

超级计算机的排名方式二：迭代法求解稀疏 $Ax=b$

12	0	0	4
0	3	0	5
0	0	5	0
4	3	0	6

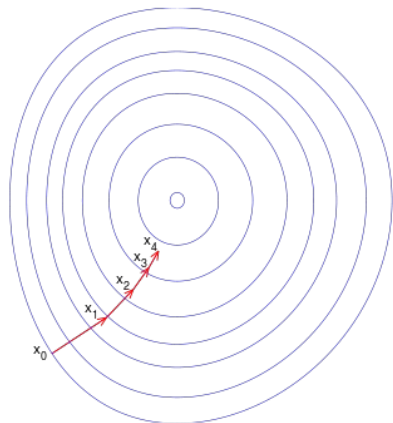
 稀疏矩阵 A

x_1
x_2
x_3
x_4

 \times

=

28
26
15
34

 解向量 x 右手边向量 b


另一个测试包 **HPCG** 使用迭代法求解稀疏 $Ax=b$ 来测试超级计算机。

图片来自维基百科网站

稀疏矩阵和其存储格式

- 如果一个矩阵的绝大多数元素都是0，则可以通过仅存储其非零元素以节省存储和计算开销。
- 可以使用压缩稀疏行（Compressed Sparse Row, CSR）格式存储稀疏矩阵。

0	0	1	0
2	3	0	0
0	0	0	0
4	0	5	6

A
(4x4)
稠密

		1	
2	3		
4		5	6

A
(4x4)
稀疏, 6个非零元

“行指针”数组 =

0	1	3	3	6
---	---	---	---	---

“列索引”数组 =

2	0	1	0	2	3
---	---	---	---	---	---

“值”数组 =

1	2	3	4	5	6
---	---	---	---	---	---

A (in CSR)
(4x4)
6个非零元

稀疏基础线性代数子程序

Sparse Basic Linear Algebra Subprograms (Sparse BLAS)

稀疏BLAS层次1： 向量-向量操作

- 如： 将一个稀疏向量和另一个稀疏向量相乘， 获得一个标量。

$$\begin{array}{ccc}
 \begin{array}{|c|c|c|c|} \hline a & b & & d \\ \hline \end{array} & \times & \begin{array}{|c|} \hline a \\ \hline b \\ \hline \\ \hline d \\ \hline \end{array} & = & \begin{array}{|c|} \hline a^2+b^2+d^2 \\ \hline \end{array} \\
 x^T & & x & & a \\
 (1 \times 4) & & (4 \times 1) & & (1 \times 1) \\
 \text{稀疏, 3个非零元} & & \text{稀疏, 3个非零元} & & \text{标量}
 \end{array}$$

稀疏BLAS层次2：矩阵-向量操作

- 如：令一个稀疏矩阵A和一个稠密向量x相乘，获得一个稠密向量y，即 $y = Ax$ 。

		1	
2	3		
4		5	6

A

(4x4)

稀疏, 6个非零元

x

a
b
c
d

x

(4x1)

稠密

=

1c
2a+3b
0
4a+5c+6d

y

(4x1)

稠密

稀疏BLAS层次3：矩阵-矩阵操作

- 如：相乘两个稀疏矩阵A和B，获得另一个稀疏矩阵C。

		1	
2	3		
4		5	6

A

(4x4)

稀疏, 6个非零元

x

			a
b		c	
	d		e
		f	

B

(4x4)

稀疏, 6个非零元

=

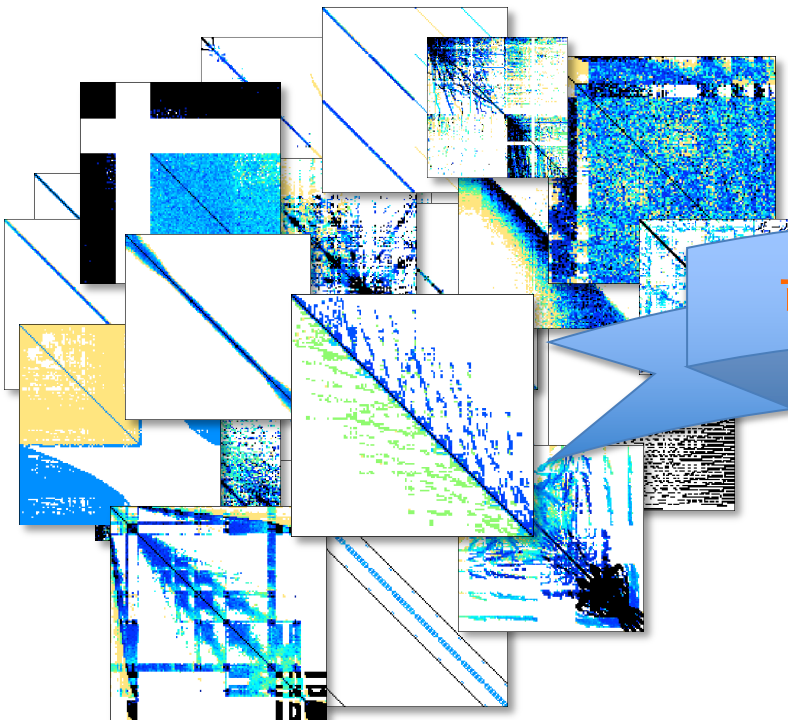
	1d		1e
3b		3c	2a
	5d	6f	4a+5e

C

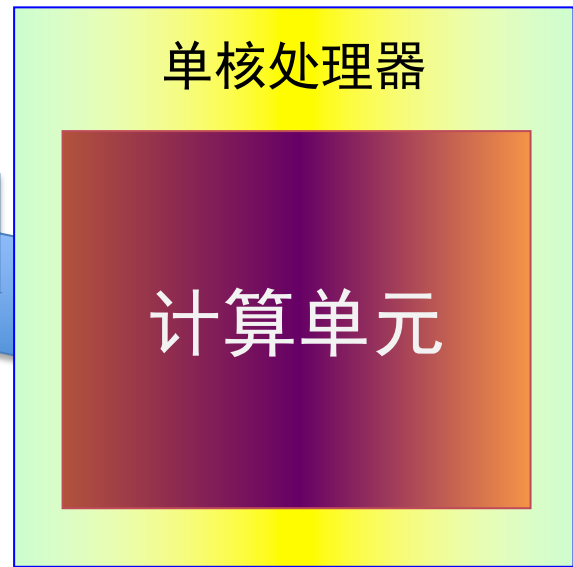
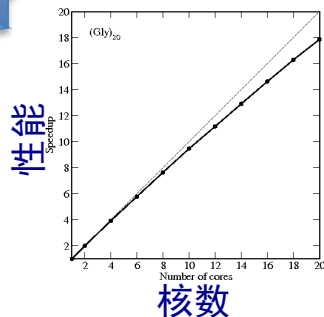
(4x4)

稀疏, 8个非零元

众核处理器上的稀疏矩阵计算



可扩展的高性能?



结构复杂、多种多样的稀疏矩阵

大量具有宽SIMD单元的
“小”核心组成的众核处理器

用四个kernel研究稀疏矩阵计算可扩展性

- 稀疏矩阵-向量乘法(SpMV)

$$\begin{bmatrix} & & 1 & \\ 2 & 3 & & \\ & & & \\ 4 & & 5 & 6 \end{bmatrix} \times \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 1c \\ 2a+3b \\ 0 \\ 4a+5c+6d \end{bmatrix}$$

- 稀疏矩阵转置(SpTRANS)

$$\begin{bmatrix} & & 1 & \\ 2 & 3 & & \\ & & & \\ 4 & & 5 & 6 \end{bmatrix} \rightarrow \begin{bmatrix} & 2 & & 4 \\ & 3 & & \\ 1 & & & 5 \\ & & & 6 \end{bmatrix}$$

- 稀疏矩阵-矩阵乘法(SpGEMM)

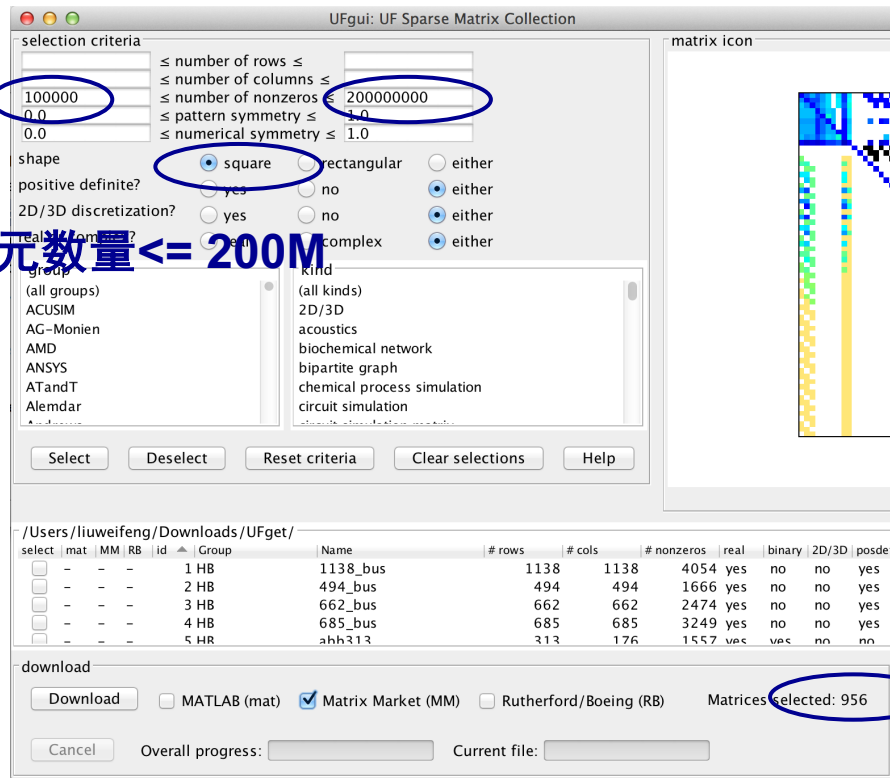
$$\begin{bmatrix} & & 1 & \\ 2 & 3 & & \\ & & & \\ 4 & & 5 & 6 \end{bmatrix} \times \begin{bmatrix} & & & a \\ b & & c & \\ & d & & e \\ & & f & \end{bmatrix} = \begin{bmatrix} & 1d & & 1e \\ 3b & & 3c & 2a \\ & & & \\ & 5d & 6f & 4a+5e \end{bmatrix}$$

- 稀疏三角解(SpTRSV)

$$\begin{bmatrix} 1 & & & \\ & 1 & & \\ & 2 & 1 & \\ 3 & & & 1 \end{bmatrix} \times \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

用于测试的稀疏矩阵（来自SuiteSparse矩阵集）

方阵，
且 $100K \leq \text{非零元数量} \leq 200M$



从总计2757
个矩阵中，
选出956个

用于测试强可扩展性的三块GPU



- nVidia GeForce Titan X, Pascal GP102, **3584 CUDA cores** @ 1.4 GHz, 10.1 Tflops (SP), 3 MB L2 cache, 12GB GDDR5X, **480 GB/s B/W**.



- nVidia GeForce GTX 1080, Pascal GP104, **2560 CUDA cores** @ 1.6 GHz, 8.2 Tflops (SP), 2 MB L2 cache, 8GB GDDR5X, **320 GB/s B/W**.



- nVidia GeForce GTX 1060, Pascal GP106, **1280 CUDA cores** @ 1.6 GHz, 4.1 Tflops (SP), 1.5 MB L2 cache, 6GB GDDR5, **192 GB/s B/W**.

计算能力 1 : 2 : 2.5

设备内存带宽 1.2 : 2 : 3

Kernel 1. 稀疏矩阵-向量乘法

Sparse Matrix-Vector Multiplication (SpMV)

稀疏矩阵-向量乘法 (SpMV)

- 令一个稀疏矩阵 A 和一个稠密向量 x 相乘, 获得一个稠密向量 y , 即 $y = Ax$ 。

		1	
2	3		
4		5	6

 A
 (4×4)

稀疏, 6个非零元

 \times

a
b
c
d

 x
 (4×1)

稠密

 $=$

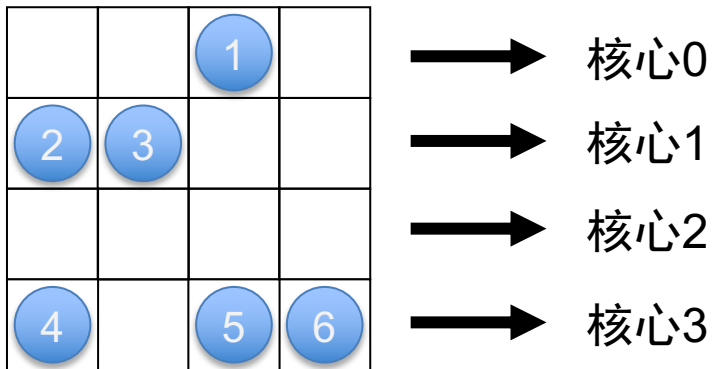
$1c$
$2a+3b$
0
$4a+5c+6d$

 y
 (4×1)

稠密

并行SpMV算法（采用CSR格式）

- 处理器的一个核心负责计算一个行块(多个行)，每个行内的点积操作可向量化。

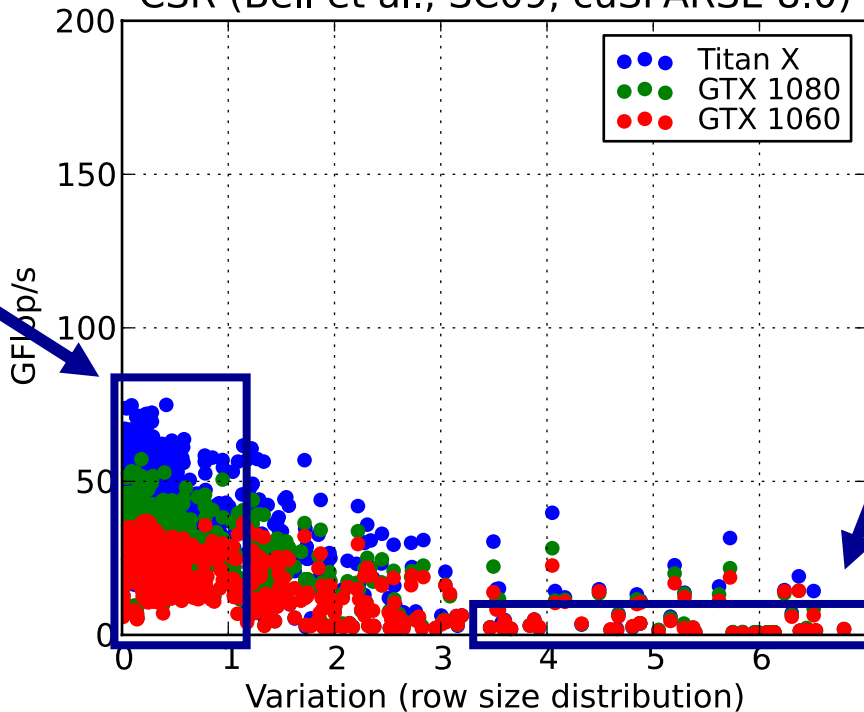


**有可能
负载不均衡，
在众核处理器上
性能严重降低。**

Nathan Bell, Michael Garland. **Implementing Sparse Matrix-Vector Multiplication on Throughput-Oriented Processors.** SC09.

CSR格式SpMV在GPU上的可扩展性

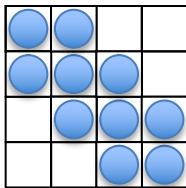
CSR (Bell et al., SC09, cuSPARSE 8.0)



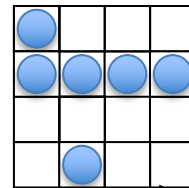
SpMV性能
随处理器
性能扩展

SpMV性能
不随处理器
性能扩展

平均
分布



power-law
分布



并行SpMV算法（采用COO格式）

- 全部计算量均分到处理器所有核心，先获得部分结果，再使用“分段和”（segmented sum）方法汇总。

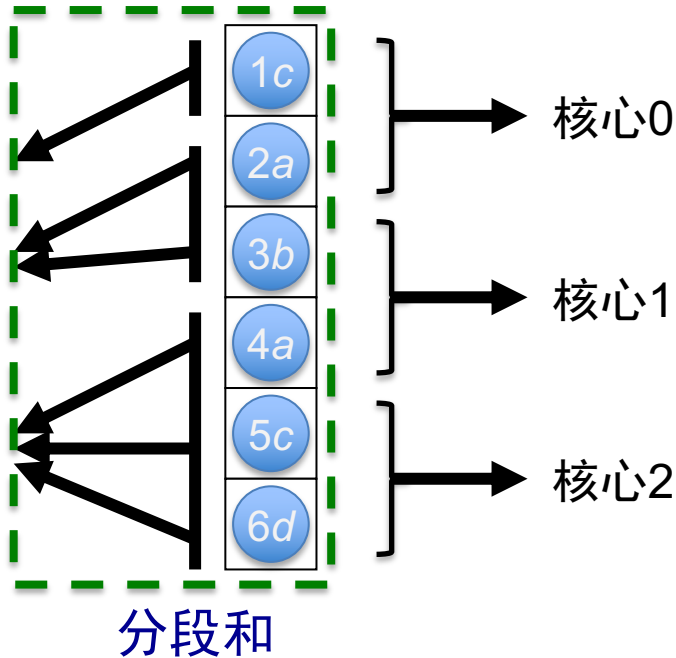
		1	
2	3		
4		5	6

x

a
b
c
d

=

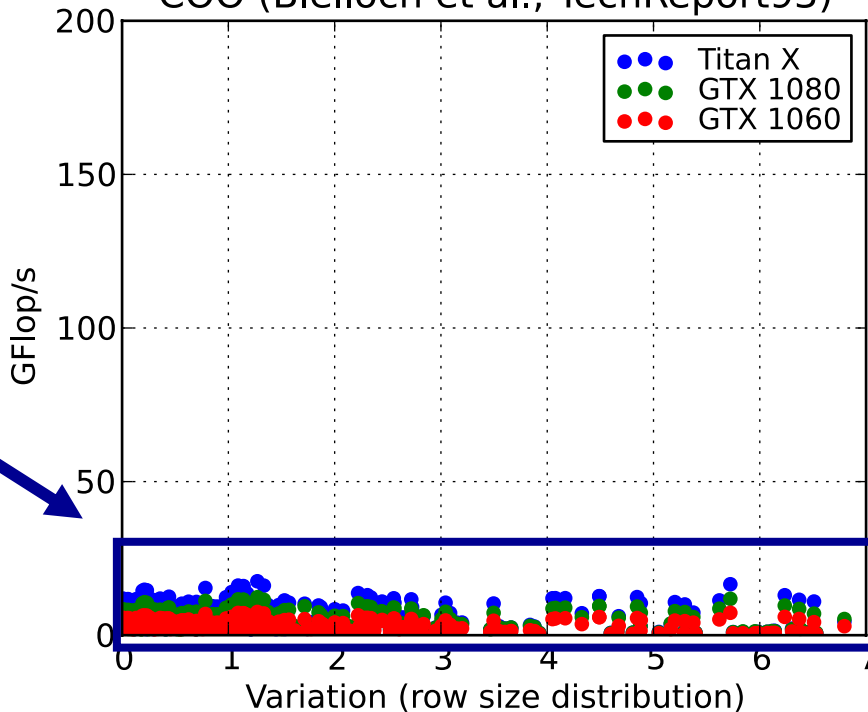
1c
2a+3b
0
4a+5c+6d



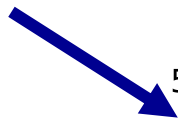
Guy Blelloch, Michael Heroux, Marco Zgha. **Segmented Operations for Sparse Matrix Computation on Vector Multiprocessors**. *TechReport, Carnegie Mellon University, 1993.*

采用“分段和”的SpMV在GPU上的可扩展性

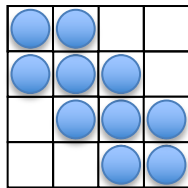
COO (Blelloch et al., TechReport93)



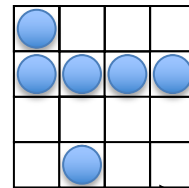
SpMV性能
总是可以
随处理器
性能扩展



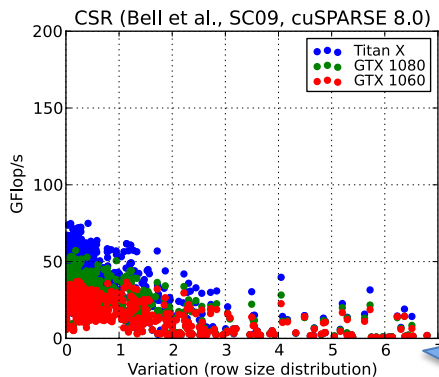
平均分布



power-law
分布

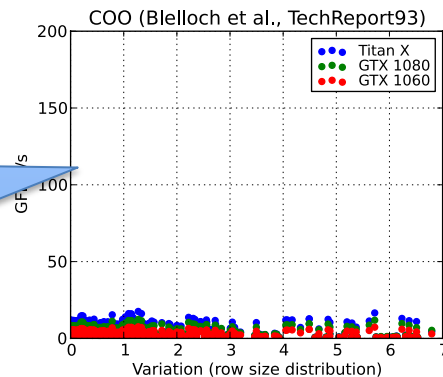


可扩展和性能的矛盾



对于规则问题
(均匀分布矩阵)
性能高的方法
不可扩展

是否存在两全其美
(可扩展+高性能)
的方法?



既对于规则
又对于不规则问题
(power-law分布矩阵)
总是可扩展的方法性能低

yaSpMV (及BCCOO/BCCOO+格式)

- 在布局上将矩阵分成二维的块 (block)，使用bit-flag存储行偏移以减少存储量，并使用“分段和”保证负载均衡，可以自调优多种参数组合。

		1				2	3
		4	5			6	
				7	8	9	10
11	12			13	14	15	16

A
(4x8)

“bit-flag” 数组 =

T	F	T	T	F
---	---	---	---	---

“列索引” 数组 =

1	3	0	2	3
---	---	---	---	---

“值” 数组 =

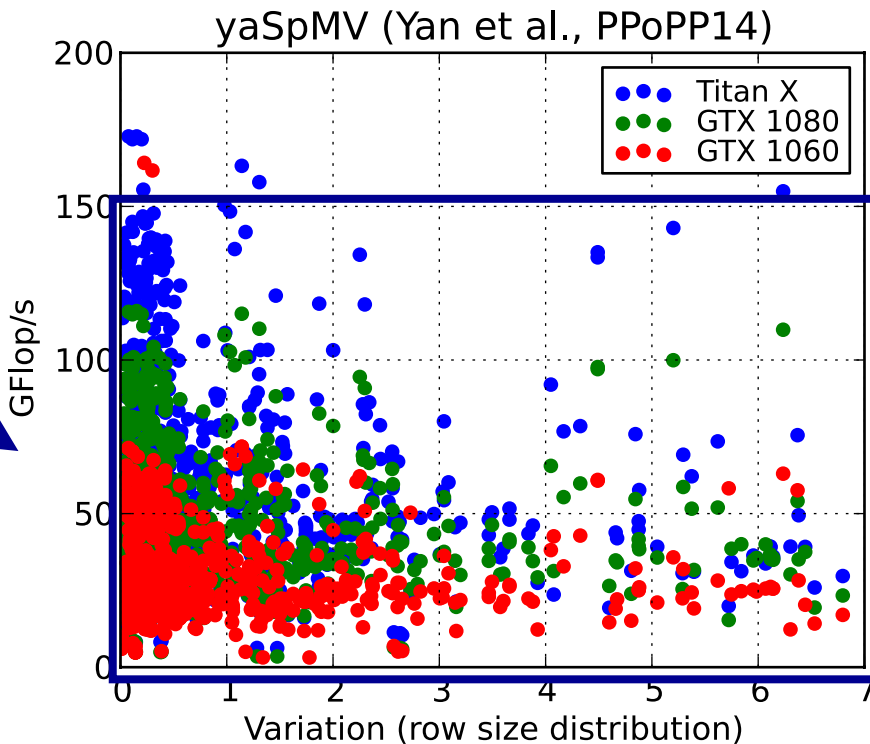
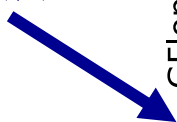
1		2	3			7	8	9	10
4	5	6		11	12	13	14	15	16

A (以BCCOO格式存储)

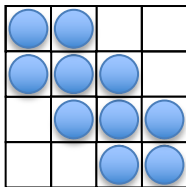
Shengen Yan, Chao Li, Yunquan Zhang, Huiyang Zhou. **yaSpMV: Yet Another SpMV Framework on GPUs. PPOPP14.**

yaSpMV在GPU上的可扩展性

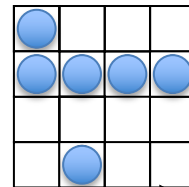
SpMV性能
总是可以
随处理器
性能扩展



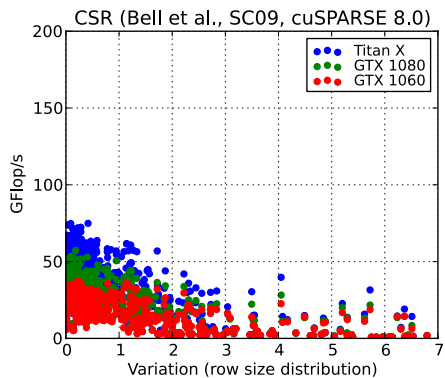
平均分布



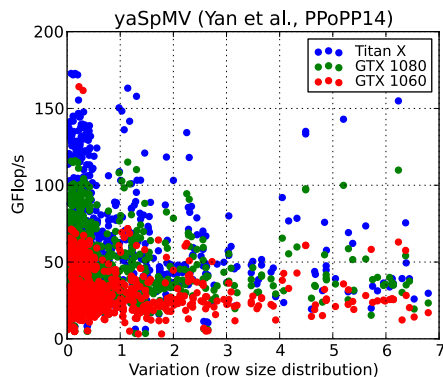
power-law分布



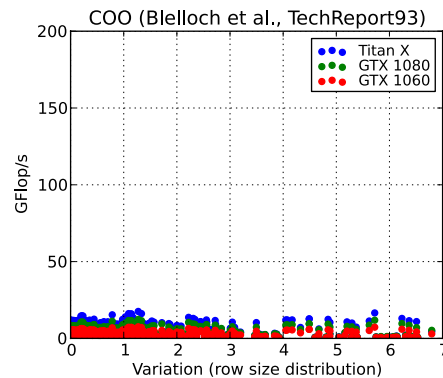
可扩展性和性能的矛盾



对于规则问题
(均匀分布矩阵)
性能高的方法
不可扩展



既性能高，又广泛可扩展



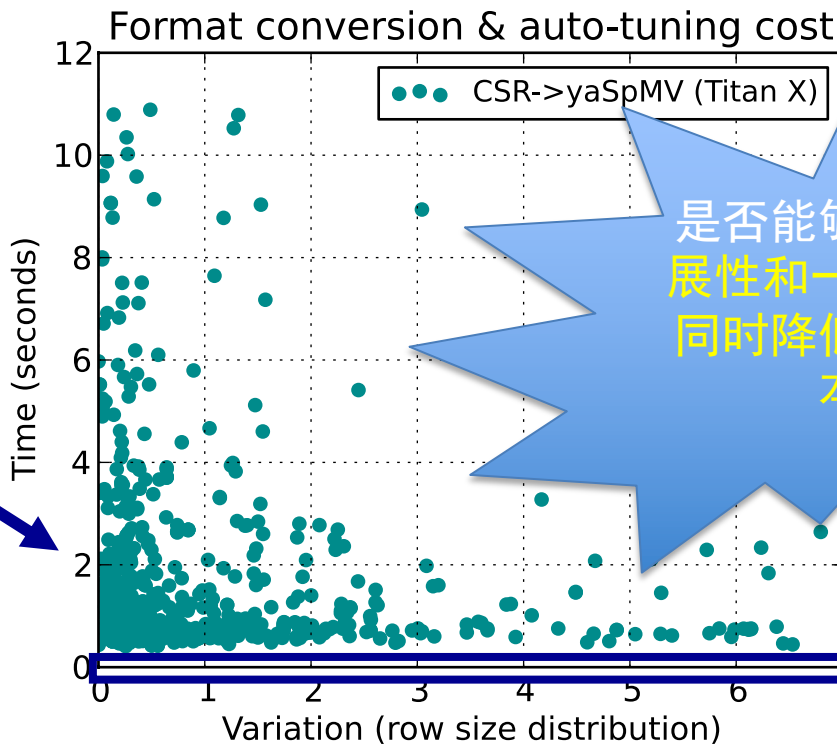
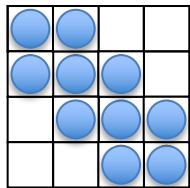
既对于规则
又对于不规则问题
(power-law分布矩阵)
总是可扩展的方法性能低

但是...

yaSpMV的格式转换和自调优成本

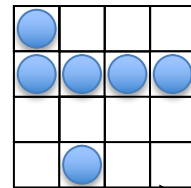
一个SpMV
操作平均
 10^{-4} 秒级别

平均
分布



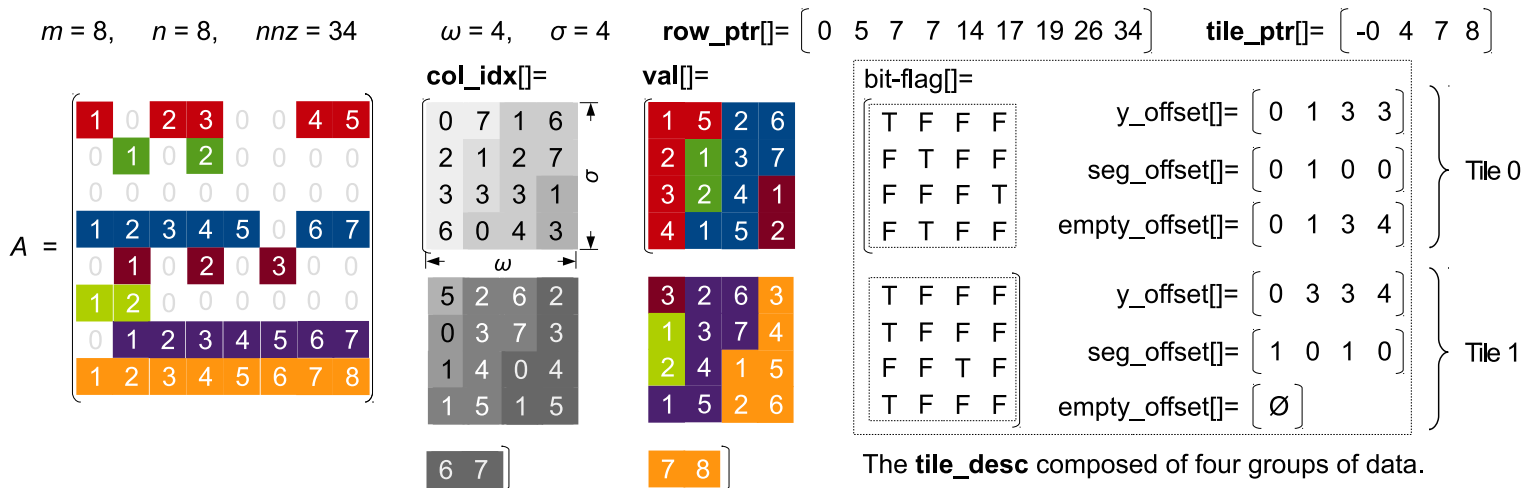
是否能够保持可扩展性和一定的性能，同时降低预处理成本？

power-law
分布



CSR5存储格式（我们的工作）

- 重分配非零元到块（tile）并存储块信息，设计目标包括负载均衡、SIMD友好、控制预处理成本和最小化存储需求。



Weifeng Liu, Brian Vinter. **CSR5: An Efficient Storage Format for Cross-Platform Sparse Matrix-Vector Multiplication.** *ICS15*.

CSR5: 分块

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

A (8x8), 34个非零元

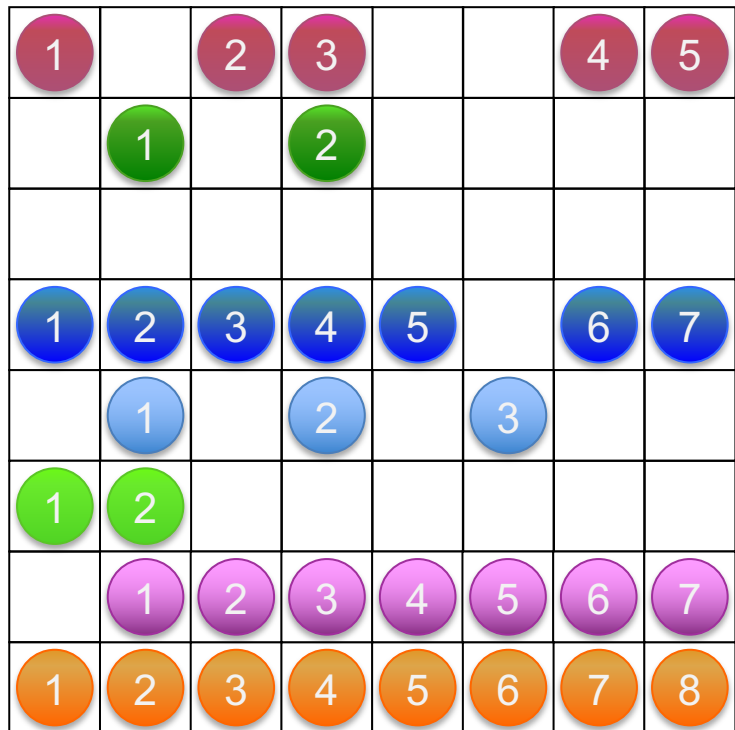
核心0

核心1

以块为单位均衡分配计算量
到计算单元，以保证负载均衡

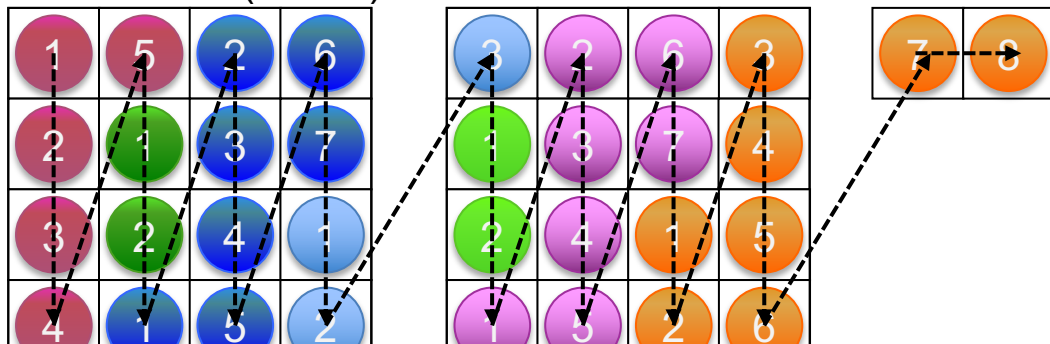
6 7

块内转置CSR数据

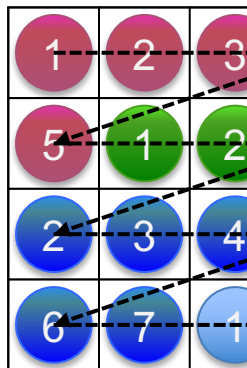


A (8x8), 34个非零元

“值”数组(CSR5) =



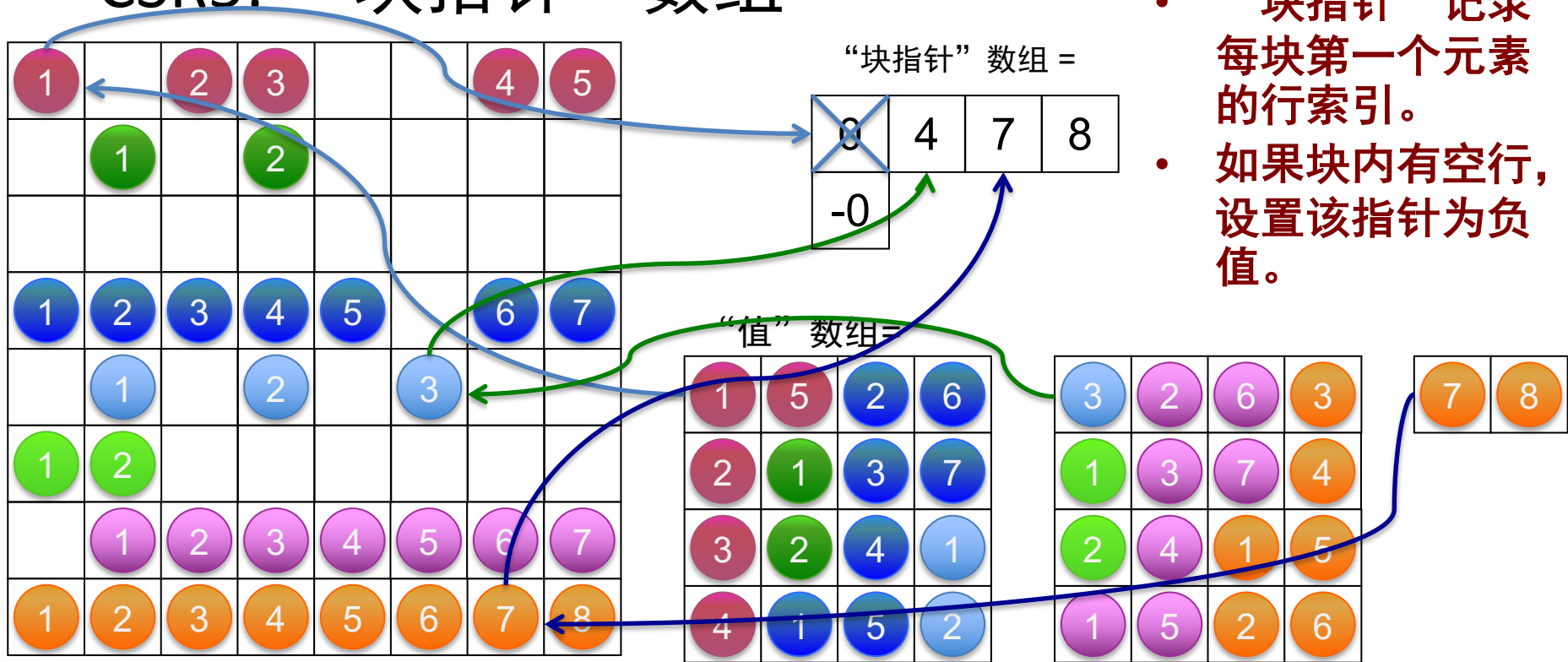
“值”数组



尽量匹配SIMD计算模式，
提高计算效率。

直接使用CSR数据，
降低从CSR格式的转换成本，
并减少额外存储开销。

CSR5: “块指针” 数组

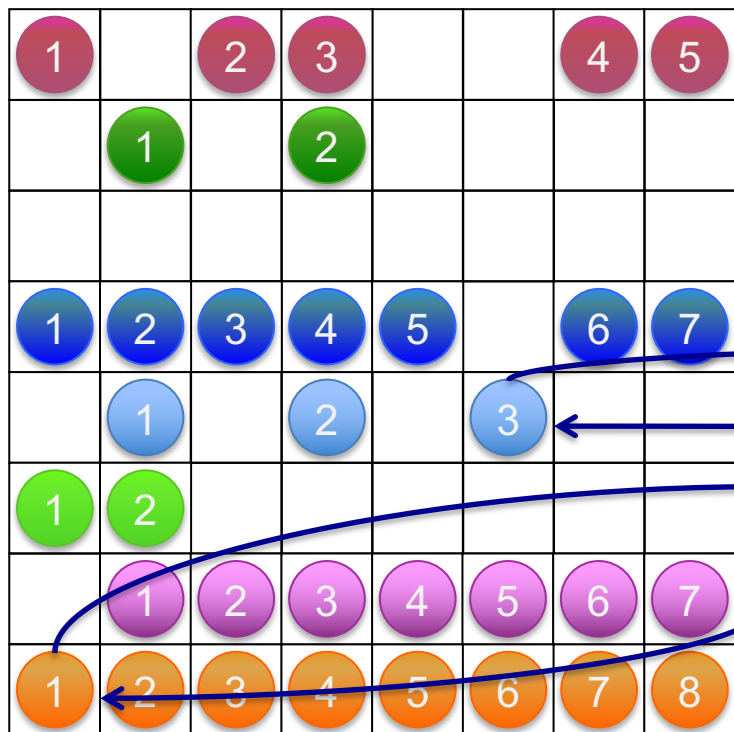


- “块指针” 记录每块第一个元素的行索引。
- 如果块内有空行，设置该指针为负值。

A (8x8), 34个非零元

CSR5: “块描述” 信息

- “y偏移”数组
令块的每一列知道该列计算应存储到的y的位置。



A (8x8), 34个非零元

“块指针”数组 =

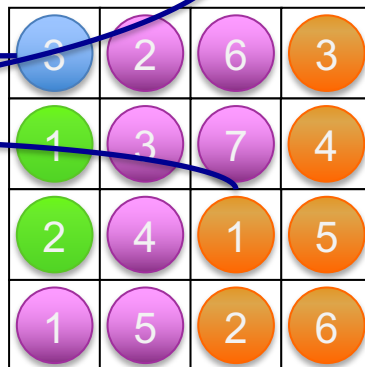
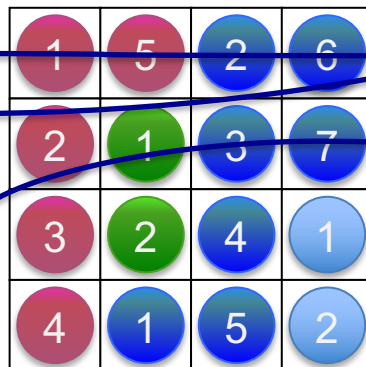
-0	4	7	8
----	---	---	---

“y偏移”数组 =

0	1	3	3
---	---	---	---

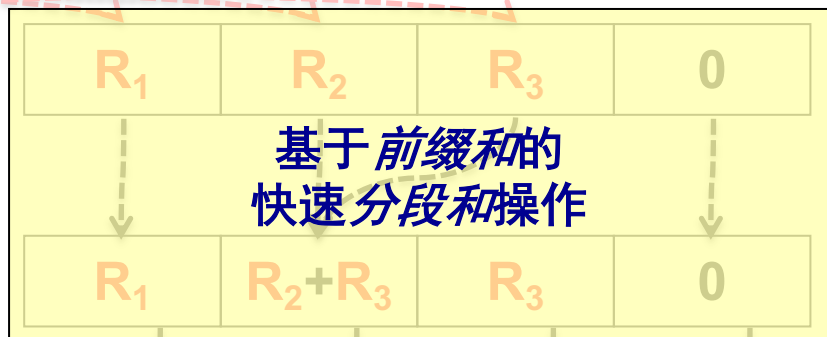
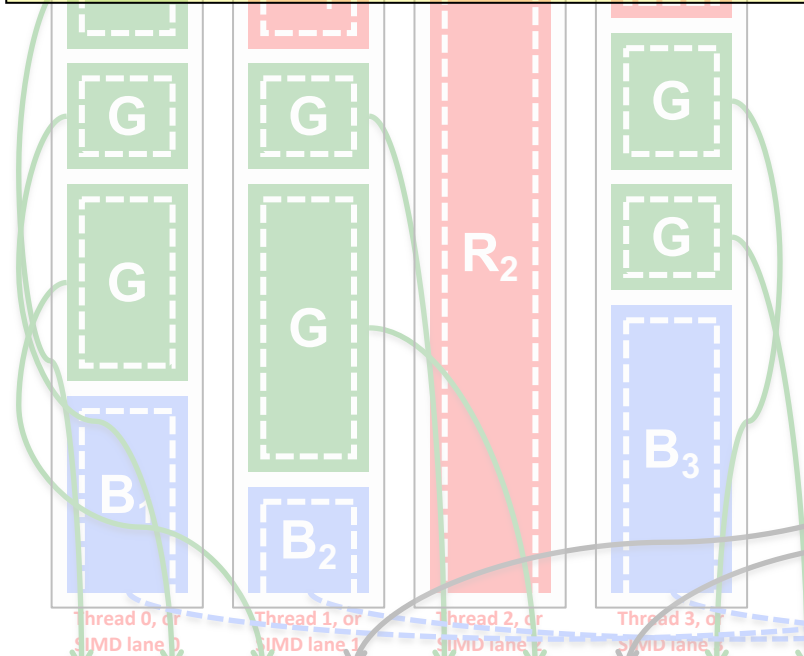
0	3	3	4
---	---	---	---

“值”数组 =



CSR5格式SpMV (一个tile为例)

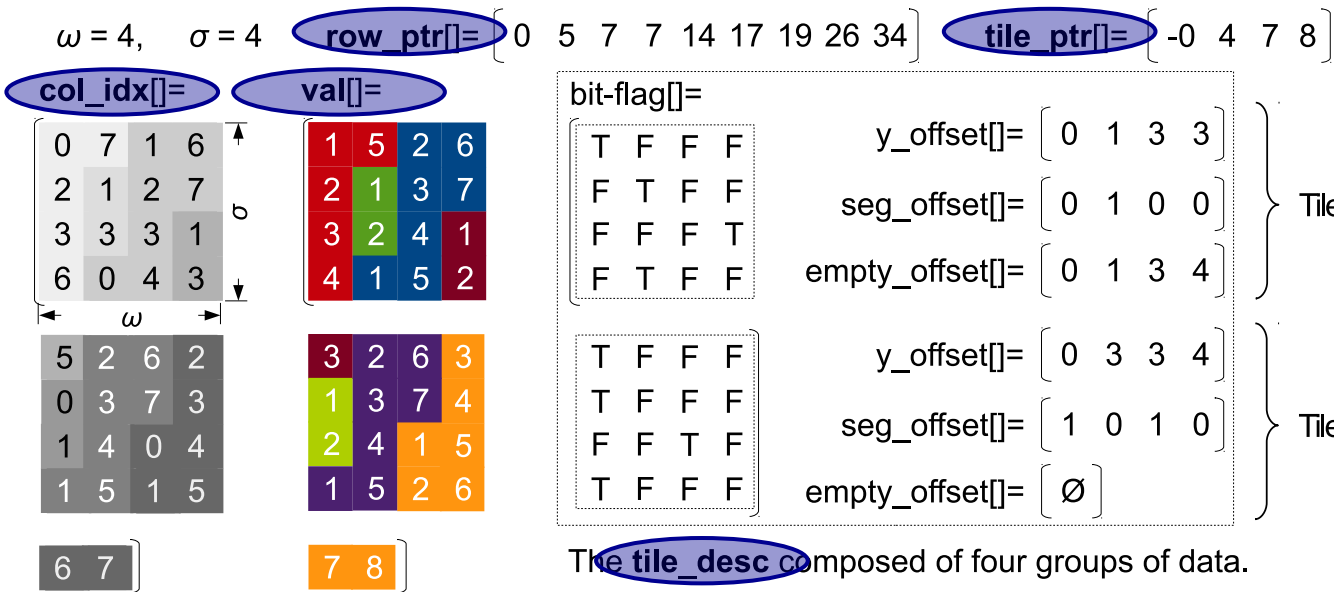
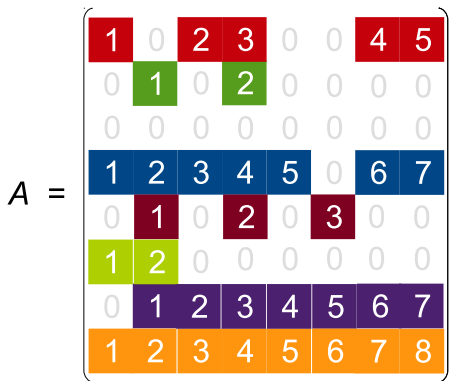
锁步的SIMD操作 + scatter



只采用最基本的SIMD操作，因此易于实现到包括CPU, GPU和Xeon Phi的多核和众核处理器。

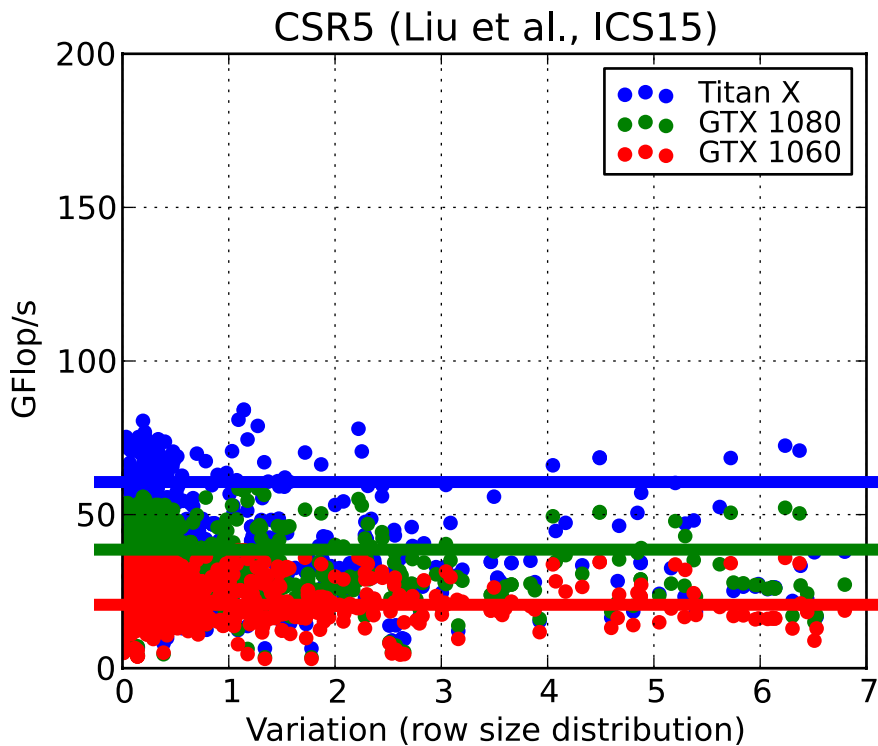
CSR5命名

$m = 8, n = 8, nnz = 34$

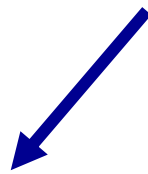


相比于CSR的3组数据，CSR5利用CSR数据并保存5组信息，故命名为CSR5。

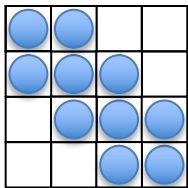
CSR5在GPU上的可扩展性



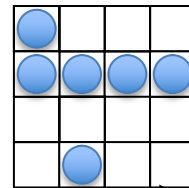
SpMV性能
随处理器
性能扩展,
并基本与
矩阵结构
无关



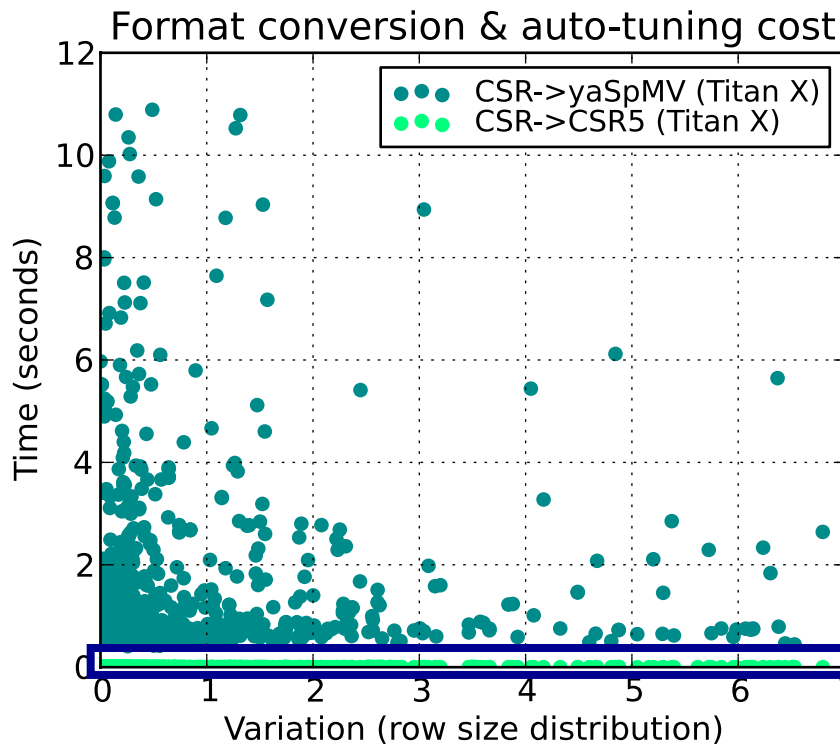
平均分布



power-law
分布

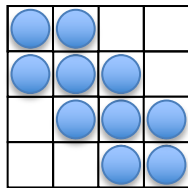


CSR5的格式转换和自调优成本

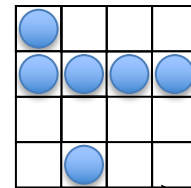


**CSR5的
预处理成本
与一个SpMV
操作为同级别**

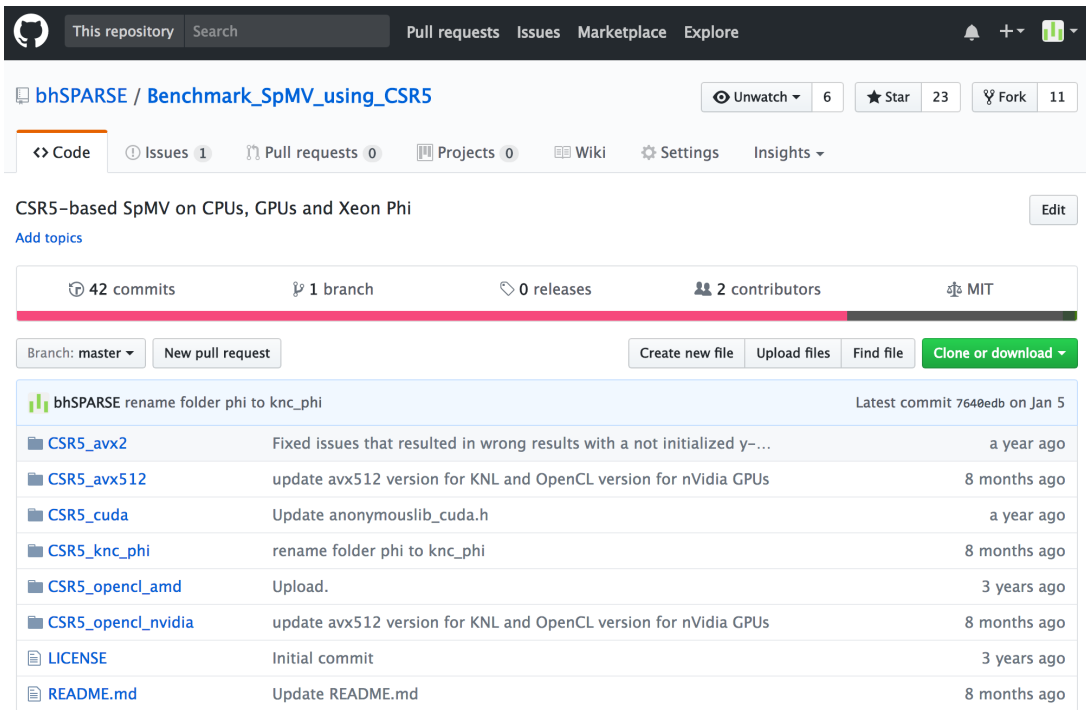
平均
分布



power-law
分布



CSR5源代码 (Github)



CSR5-based SpMV on CPUs, GPUs and Xeon Phi

Add topics

42 commits 1 branch 0 releases 2 contributors MIT

Branch: master New pull request Create new file Upload files Find file Clone or download

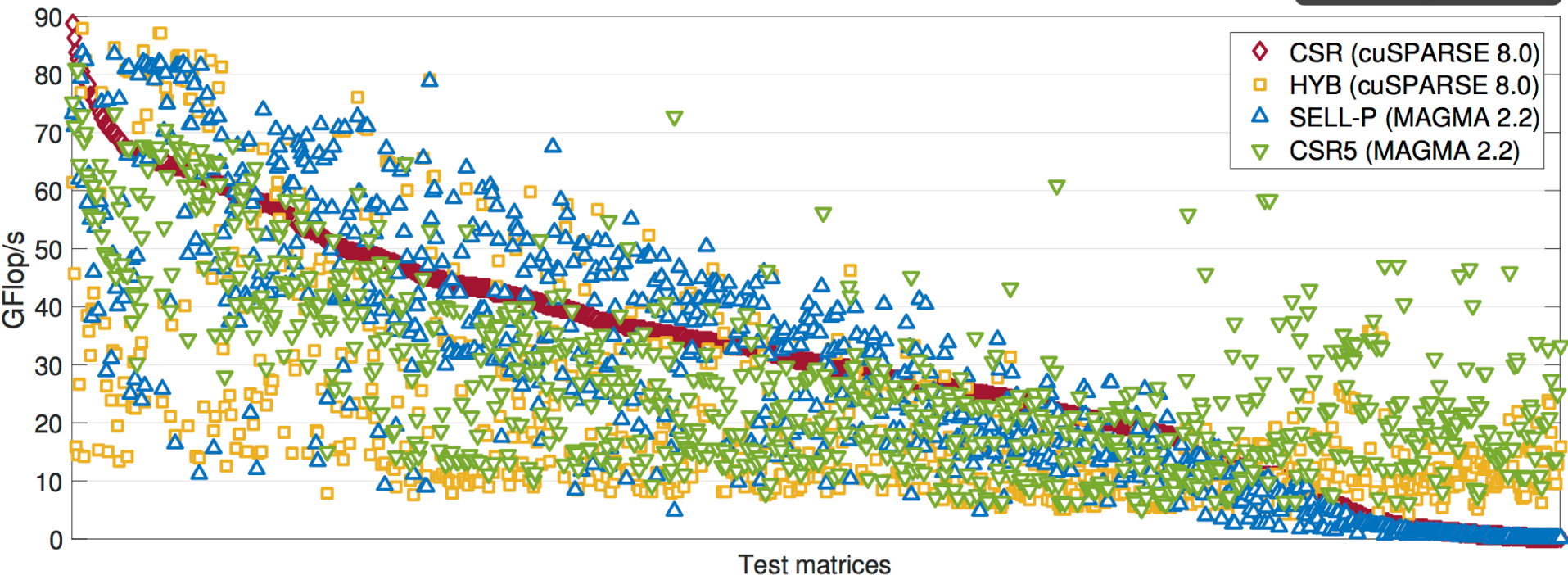
bhSPARSE rename folder phi to knc_phi		Latest commit 7640edb on Jan 5
CSR5_avx2	Fixed issues that resulted in wrong results with a not initialized y-...	a year ago
CSR5_avx512	update avx512 version for KNL and OpenCL version for nVidia GPUs	8 months ago
CSR5_cuda	Update anonymouslib_cuda.h	a year ago
CSR5_knc_phi	rename folder phi to knc_phi	8 months ago
CSR5_openc1_amd	Upload.	3 years ago
CSR5_openc1_nvidia	update avx512 version for KNL and OpenCL version for nVidia GPUs	8 months ago
LICENSE	Initial commit	3 years ago
README.md	Update README.md	8 months ago

https://github.com/bhSPARSE/Benchmark_SpMV_using_CSR5

MAGMA线性代数库中的CSR5

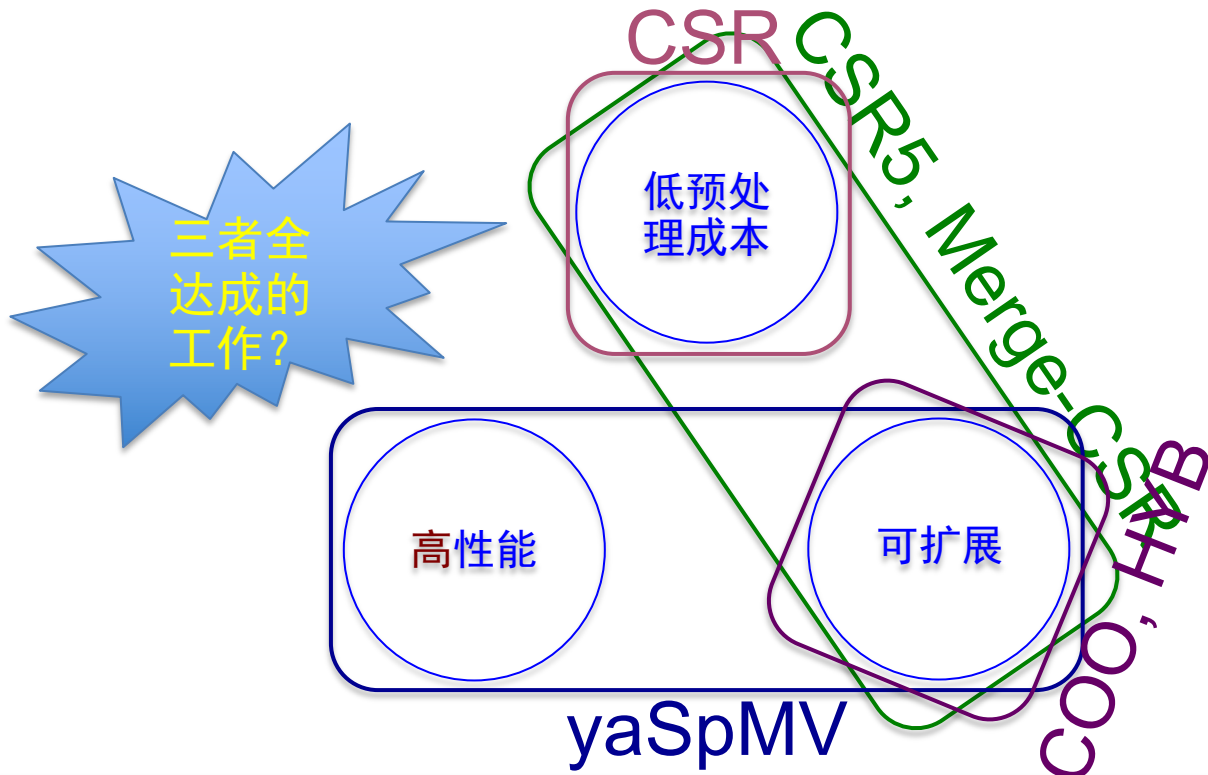
MAGMA

ICL UTK



图片来自Dr. Hartwig Anzt at ICL, UTK; Device: NVIDIA Tesla P100; Precision: double.

低预处理成本、高性能和可扩展 (三者最多择其二)



Kernel 2. 稀疏矩阵-矩阵乘法

Sparse Matrix-Matrix Multiplication

(SpGEMM)

稀疏矩阵-矩阵乘法 (SpGEMM)

- 相乘两个稀疏矩阵A和B, 获得另一个稀疏矩阵C。

		1	
2	3		
4		5	6

A

(4x4)

稀疏, 6个非零元

x

			a
b		c	
	d		e
		f	

B

(4x4)

稀疏, 6个非零元

=

	1d		1e
3b		3c	2a
	5d	6f	4a+5e

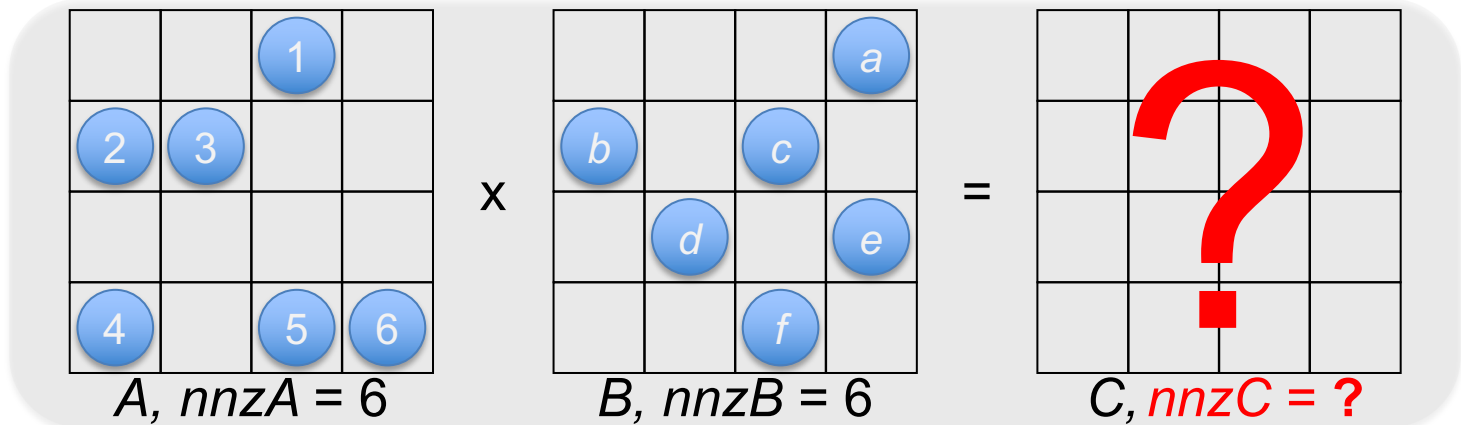
C

(4x4)

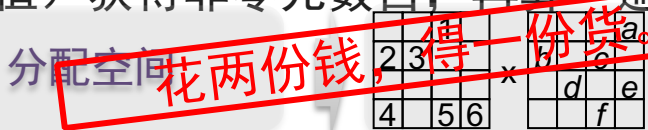
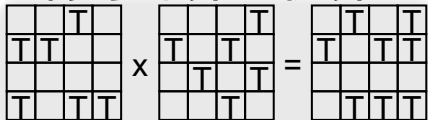
稀疏, 8个非零元

SpGEMM挑战1: C中非零元数目未知

- 因为结果矩阵C是稀疏的，且其非零元数目无法事先预知，故无法在计算前进行内存预分配。



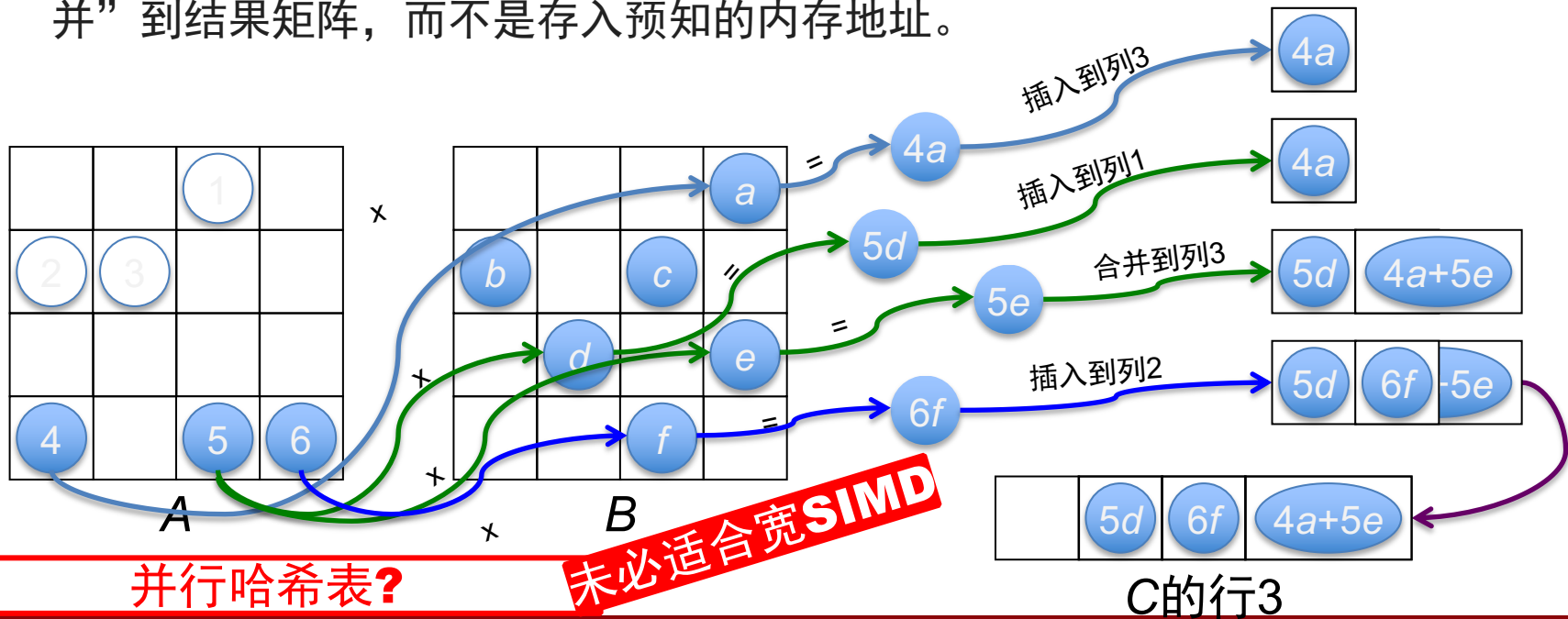
- 先符号计算（仅算结构，不算值）获得非零元数目，再算一遍SpGEMM。



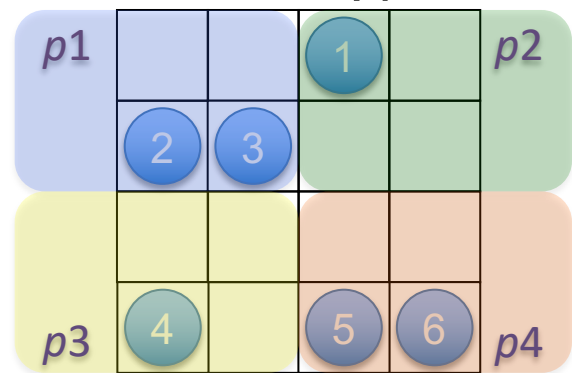
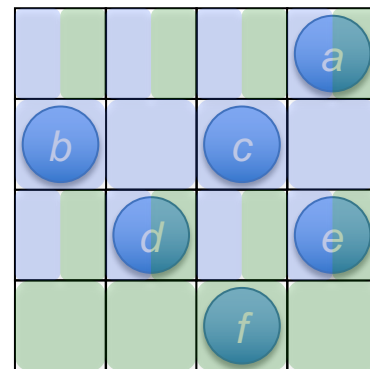
成本高

SpGEMM挑战2：并行插入操作

- 由于稀疏结构，计算出的结果非零元需要调用非直接地址以“插入”或“合并”到结果矩阵，而不是存入预知的内存地址。



SpGEMM挑战3：负载均衡



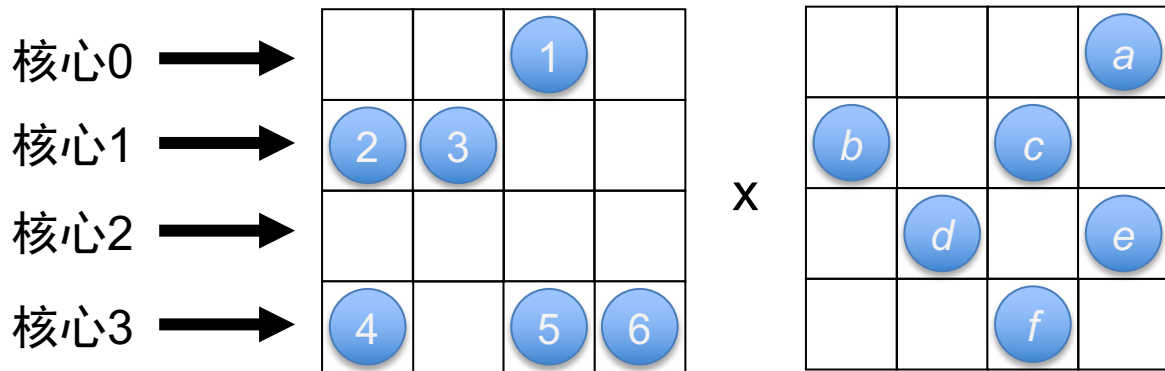
在A上的
行块分割
可能不均衡

在A上的
二维块分割
可能不均衡

平均分配 A的非零元	访问B
均衡	可能不均衡
整体可能不均衡	

并行SpGEMM算法 (cuSPARSE)

- 分配A的一个行块到一个核心，产生相应的一个C的行块，且使用哈希表计算每行内的插入和合并。cuSPARSE近年来一直使用此方法。

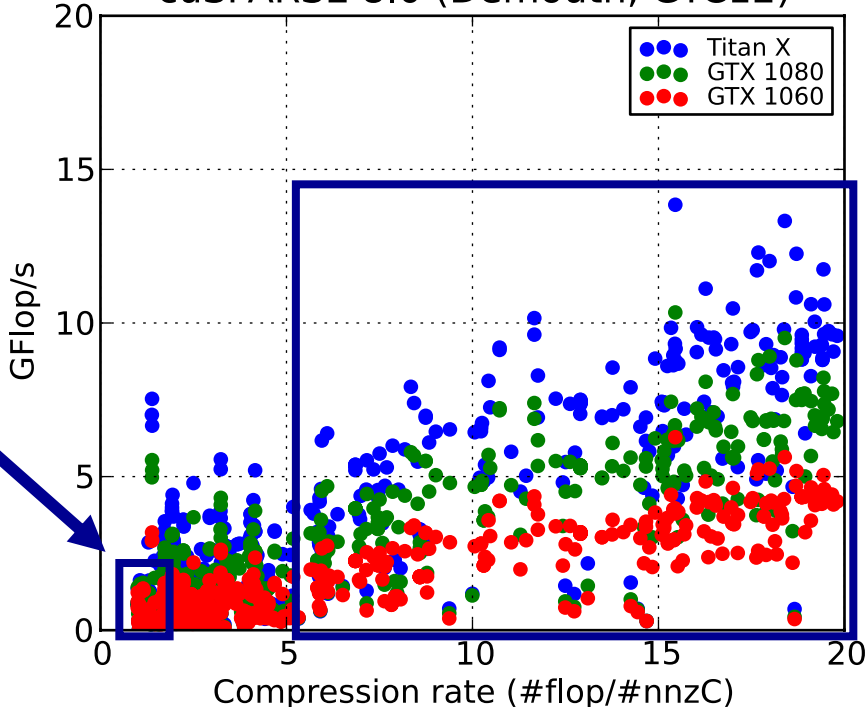


有可能
负载不均衡，
在众核处理器上
性能严重降低。

Julien Demouth, **Sparse Matrix-Matrix Multiplication on the GPU**, NVIDIA GTC 2012 talk.

cuSPARSE在GPU上的可扩展性

cuSPARSE 8.0 (Demouth, GTC12)



SpGEMM性能
不随处理器
性能扩展

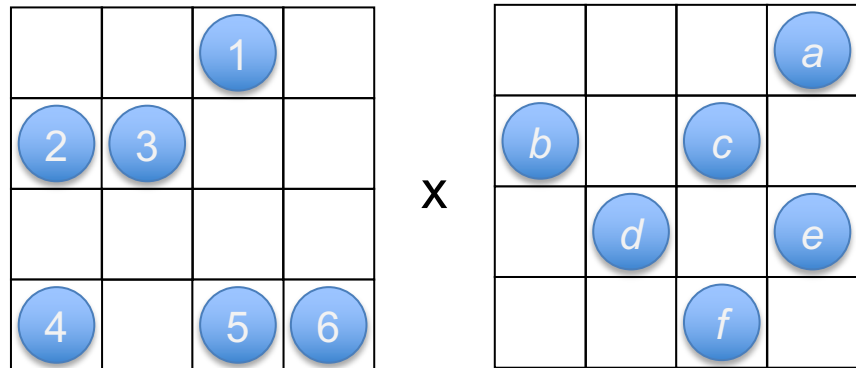
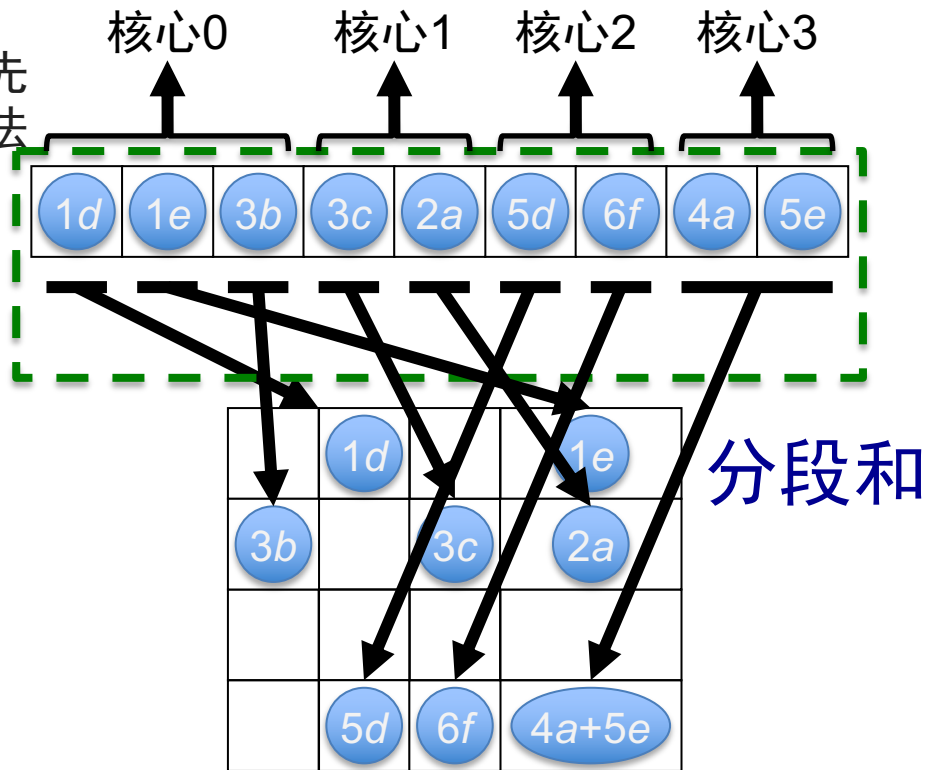
SpGEMM性能
一定程度上
随处理器
性能扩展

“插入”
较多

“合并”
较多

并行SpGEMM算法 (CUSP)

- 全部计算量均分到处理器所有核心，先获得部分结果，再使用“分段和”方法汇总，CUSP库一直使用此方法。



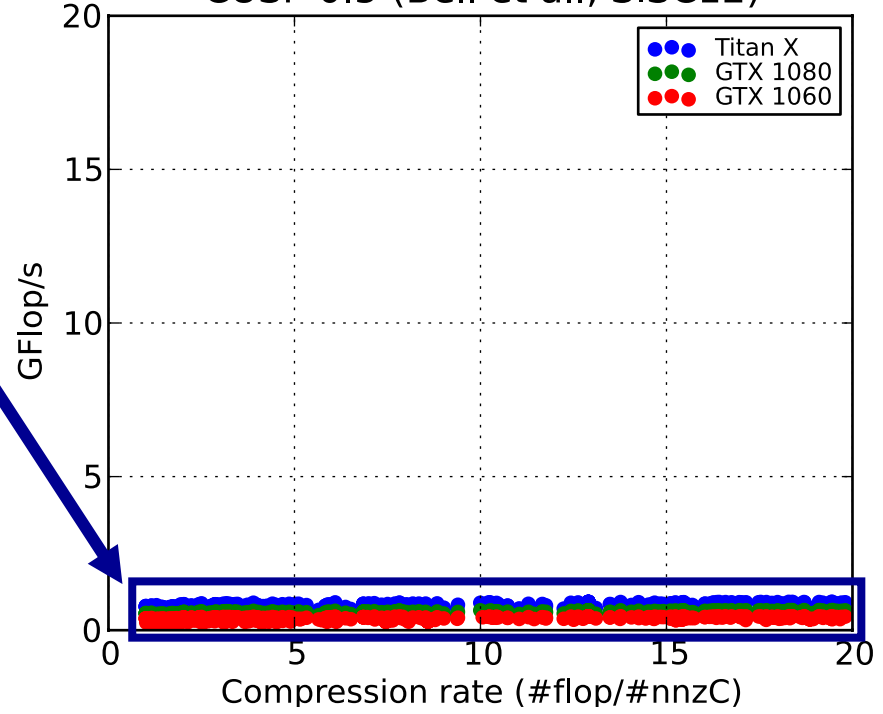
Nathan Bell, Steven Dalton, Luke Olson.
 Exposing Fine-Grained Parallelism in
 Algebraic Multigrid Methods. *SISC*. 2012.

CUSP在GPU上的可扩展性

SpGEMM性能

总是可以
随处理器
性能扩展

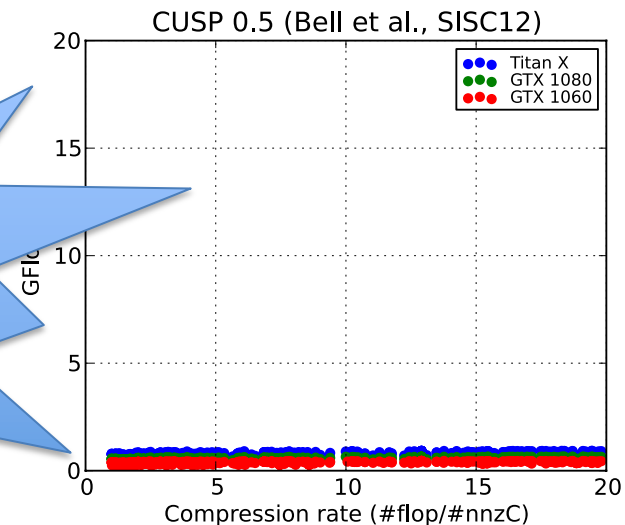
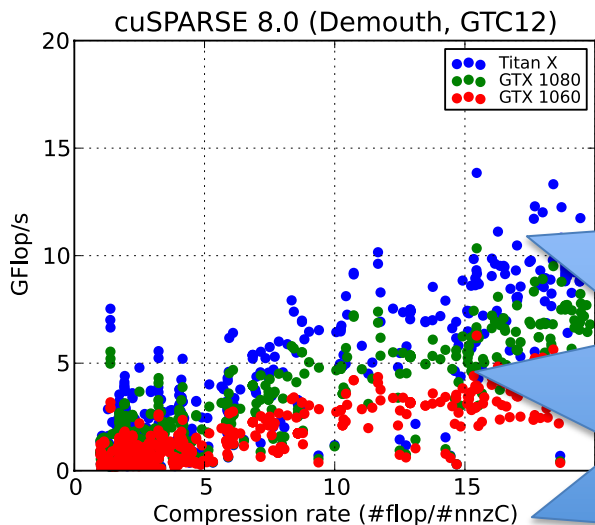
CUSP 0.5 (Bell et al., SISC12)



“插入”
较多

“合并”
较多

可扩展性和性能的矛盾

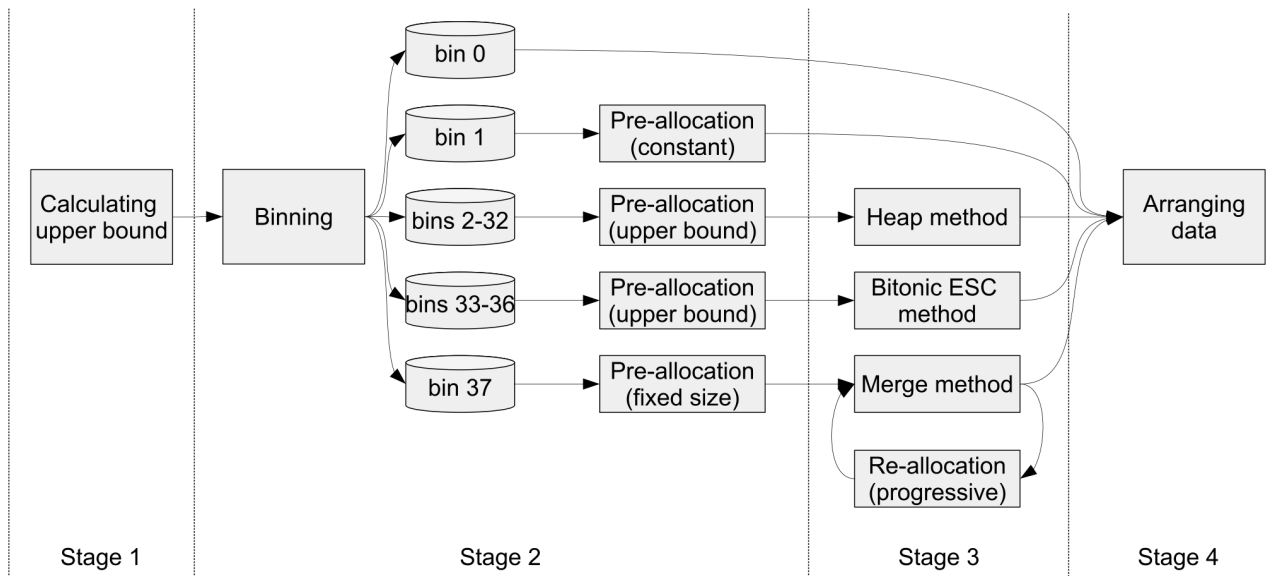


是否存在两全其美（可扩展+高性能）的方法？

对于规则问题
性能高的方法
不可扩展

既对于规则又
对于不规则问题
总是可扩展的方法性能低

SpGEMM框架（我们的工作）

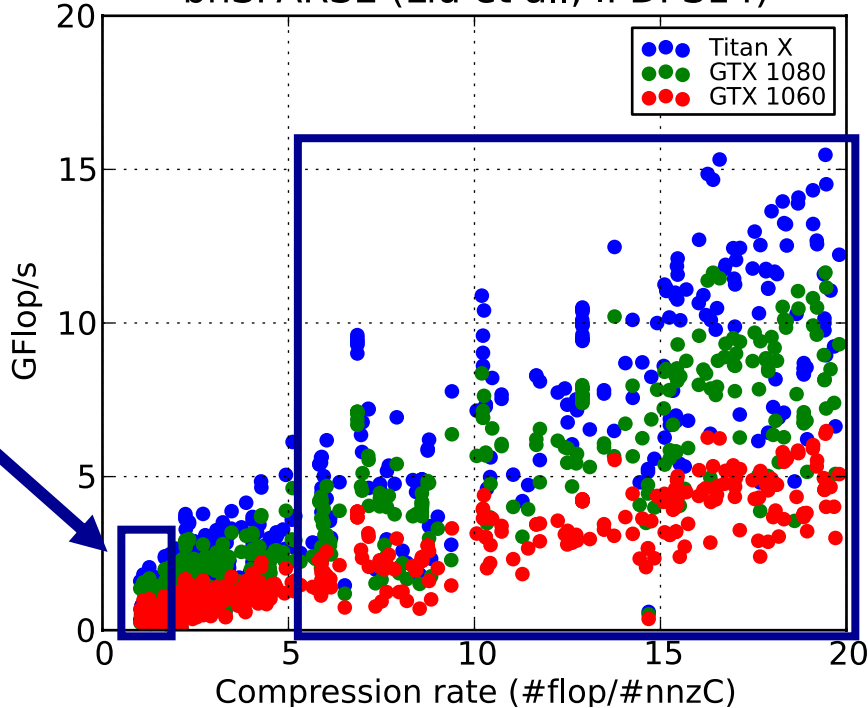


Weifeng Liu, Brian Vinter. **A Framework for General Sparse Matrix-Matrix Multiplication on GPUs and Heterogeneous Processors.** *JPDC. 2015.* (extended from *IPDPS14*)

Weifeng Liu, Brian Vinter. **An Efficient GPU General Sparse Matrix-Matrix Multiplication for Irregular Data.** *IPDPS14.*

bhSPARSE在GPU上的可扩展性

bhSPARSE (Liu et al., IPDPS14)



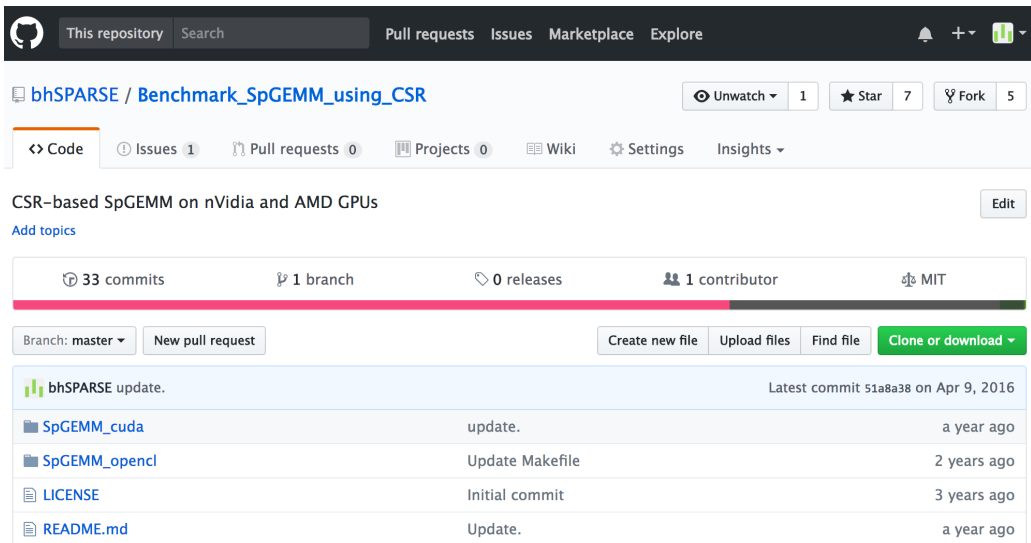
SpGEMM性能
一定程度上
随处理器
性能扩展

SpGEMM性能
一定程度上
随处理器
性能扩展

“插入”
较多

“合并”
较多

bhSPARSE-SpGEMM源代码 (Github)



https://github.com/bhSPARSE/Benchmark_SpGEMM_using_CSR

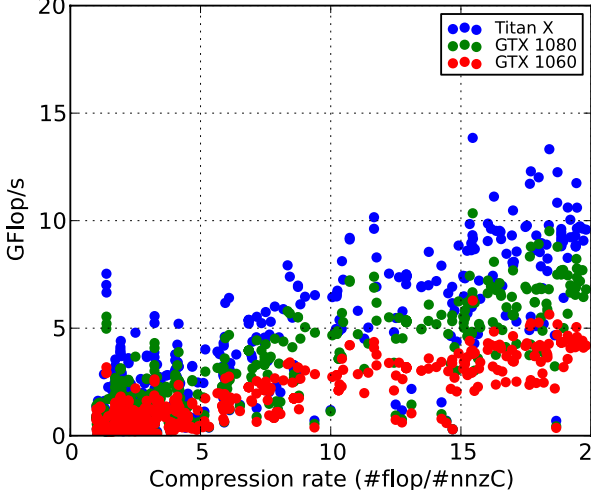
该方法已被集成进AMD的cISPARE稀疏线性代数库

<https://github.com/clMathLibraries/clSPARSE>

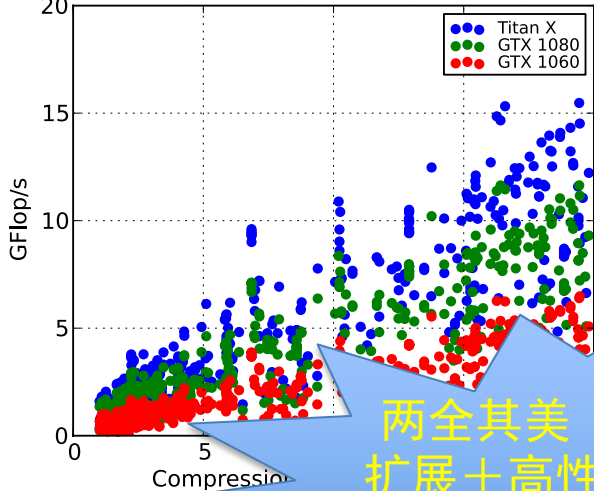


可扩展性和性能的矛盾

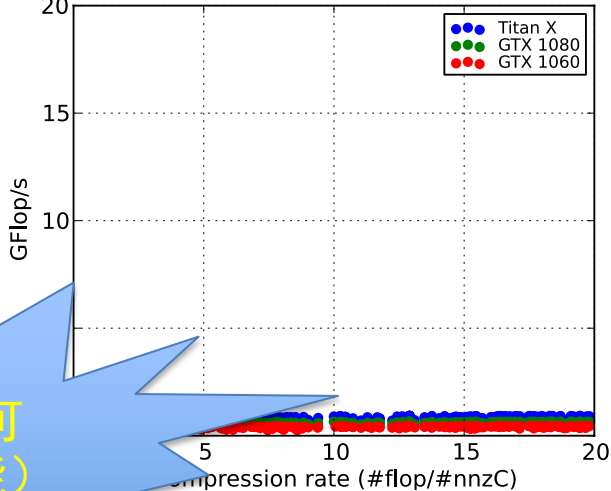
cuSPARSE 8.0 (Demouth, GTC12)



bhSPARSE (Liu et al., IPDPS14)



CUSP 0.5 (Bell et al., SISC12)



两全其美（可扩展+高性能）的工作？

对于规则问题
性能高的方法
不可扩展或扩展的不理想

既对于规则又
对于不规则问题
总是可扩展的方法性能低

Kernel 3.稀疏矩阵转置 Sparse Transposition (SpTRANS)

稀疏矩阵转置 (SpTRANS)

- 转置一个稀疏矩阵A (CSR格式) 到另一个稀疏矩阵B (CSR格式), 即 $B = A^T$ 。
该操作等同于对A的CSR格式到CSC格式的转换。

“行指针”数组 =

0	1	3	3	6
---	---	---	---	---

“列索引”数组 =

2	0	1	0	2	3
---	---	---	---	---	---

“值”数组 =

1	2	3	4	5	6
---	---	---	---	---	---

		1	
2	3		
4		5	6

->

	2		4
	3		
1			5
			6

“行指针”数组 =

0	2	3	5	6
---	---	---	---	---

“列索引”数组 =

1	3	1	0	3	3
---	---	---	---	---	---

“值”数组 =

2	4	3	1	5	6
---	---	---	---	---	---

A

(4x4)

稀疏, 6个非零元

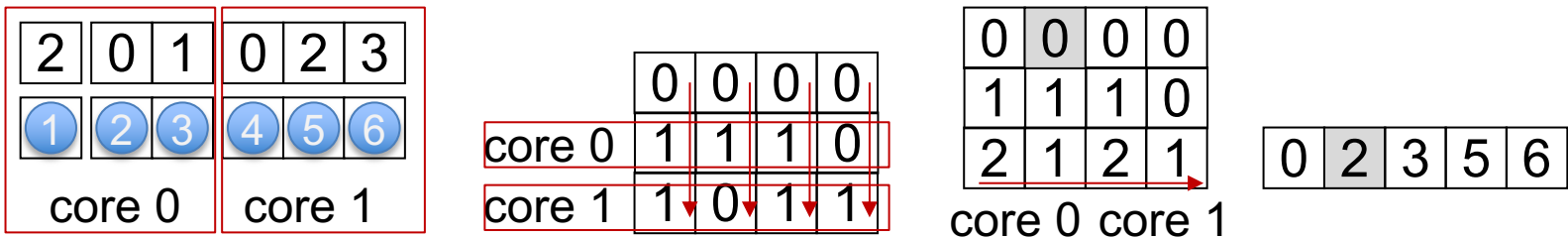
A^T

(4x4)

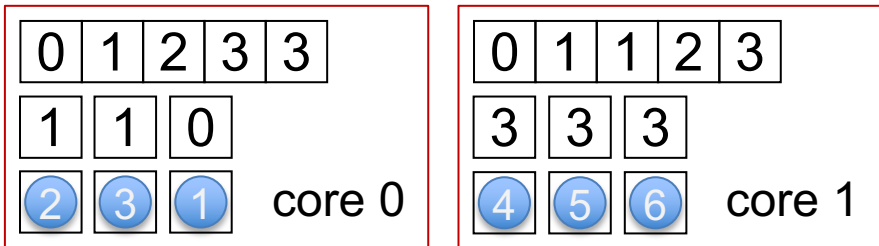
稀疏, 6个非零元

ScanTrans和MergeTrans（我们的工作）

- ScanTrans: 均分非零元, 进行垂直scan和水平scan操作, 再计算偏移位置

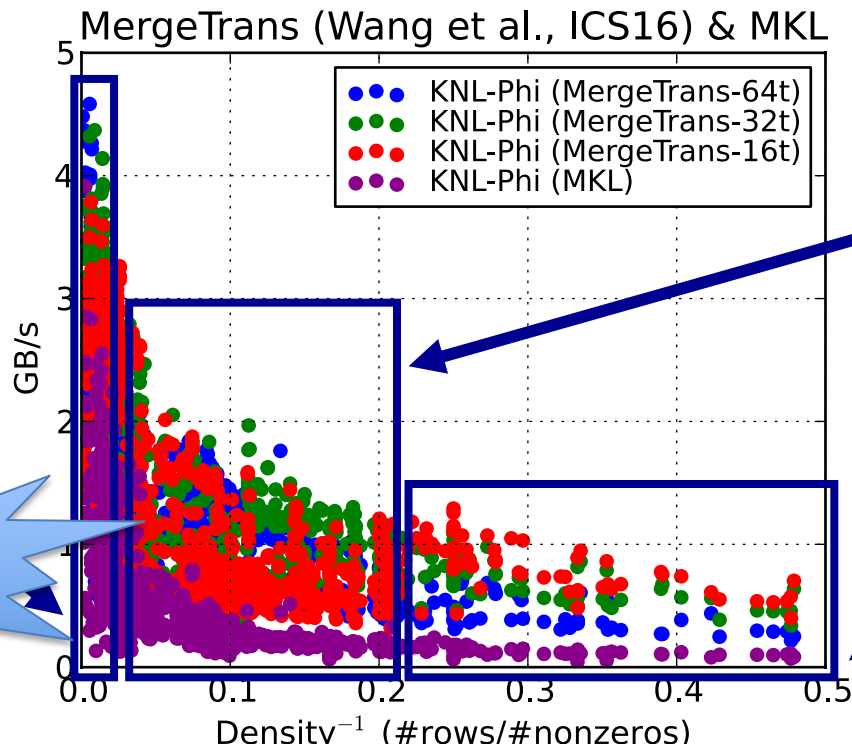


- MergeTrans: 均分非零元, 每线程计算子矩阵的转置, 再两两合并获得转置矩阵。



Hao Wang, Weifeng Liu, Kaixi Hou, Wu-chun Feng. **Parallel Transposition of Sparse Data Structures. ICS16.**

MergeTrans和MKL在KNL-Phi上的可扩展性



SpTRANS性能
 一定程度上
 随处理器
 性能扩展
 (64t性能最好)

SpTRANS性能
 不随处理器
 性能扩展
 (32t性能最好)

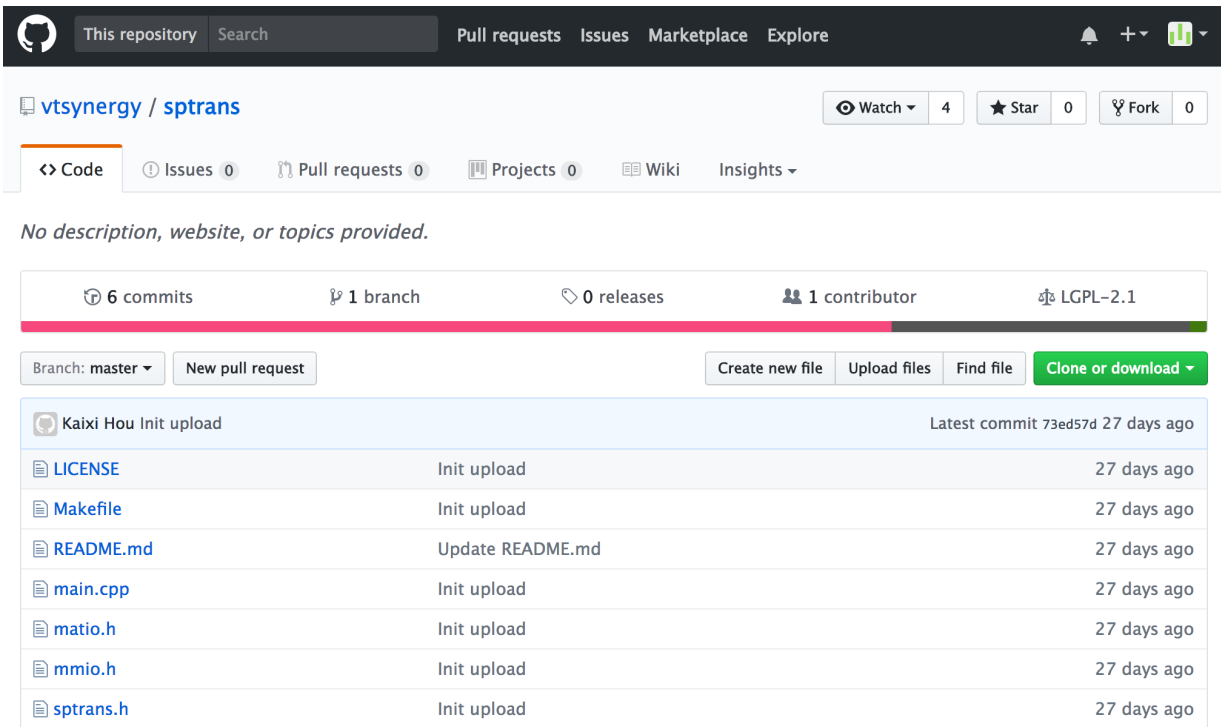
SpTRANS性能
 随处理器
 性能逆向扩展
 (16t性能最好)

可扩展的
 方法?

较“稠密”

较“稀疏”

Scan/MergeTrans 源代码 (Github)



This repository Pull requests Issues Marketplace Explore

vtsynergy / sptrans Watch 4 Star 0 Fork 0

<> Code Issues 0 Pull requests 0 Projects 0 Wiki Insights

No description, website, or topics provided.

6 commits 1 branch 0 releases 1 contributor LGPL-2.1

Branch: master New pull request Create new file Upload files Find file Clone or download

File	Commit Message	Time
Kaixi Hou Init upload Latest commit 73ed57d 27 days ago		
LICENSE	Init upload	27 days ago
Makefile	Init upload	27 days ago
README.md	Update README.md	27 days ago
main.cpp	Init upload	27 days ago
matio.h	Init upload	27 days ago
mmio.h	Init upload	27 days ago
sptrans.h	Init upload	27 days ago

<https://github.com/vtsynergy/sptrans>

Kernel 4.稀疏三角解 Sparse Triangular Solve (SpTRSV)

稀疏三角解 (SpTRSV)

- 求解一个稀疏线性系统 $Lx = b$ ，其中 L 为下三角稀疏矩阵， b 为稠密右手边向量， x 为稠密待求解向量。

1			
	1		
	2	1	
3			1

 L

(4x4)

稀疏, 6个非零元

已知

 \times

x_0
x_1
x_2
x_3

 x

(4x1)

稠密

未知

=

a
b
c
d

 b

(4x1)

稠密

已知

待解系统:

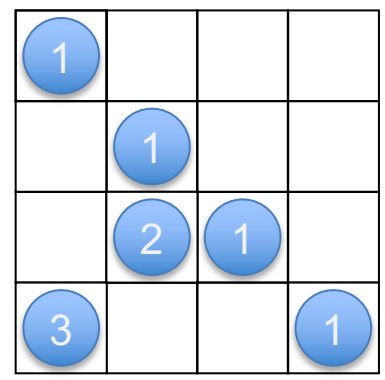
$$\begin{cases} 1 \cdot x_0 = a \\ 1 \cdot x_1 = b \\ 2 \cdot x_1 + 1 \cdot x_2 = c \\ 3 \cdot x_0 + 1 \cdot x_3 = d \end{cases}$$


 解向量 x :

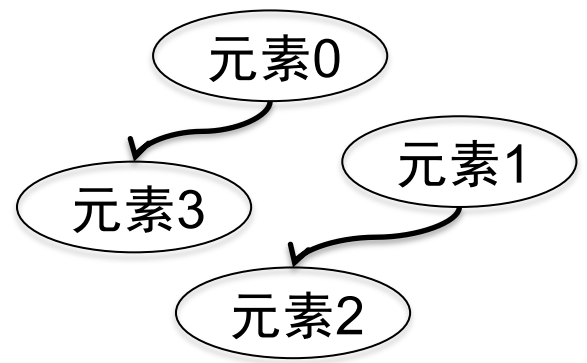
$$\begin{cases} x_0 = a \\ x_1 = b \\ x_2 = c - 2b \\ x_3 = d - 3a \end{cases}$$

层次集 (Level-set) 方法进行并行SpTRSV

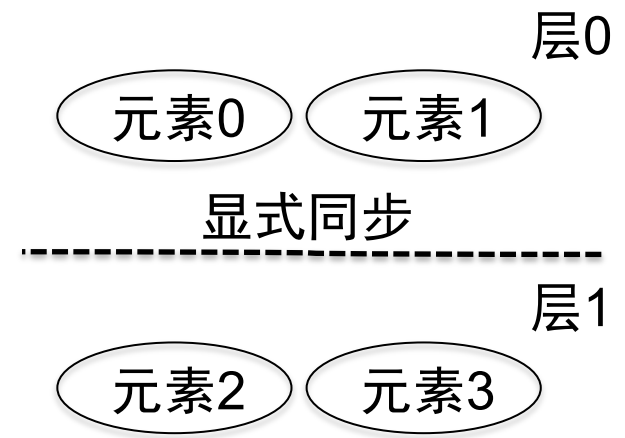
- 层次集 (level-set) 方法先对矩阵结构进行分析 (预处理) 获得层次信息, 再进行求解, 即层次内并行, 层次间串行 (需要同步)。



矩阵形态



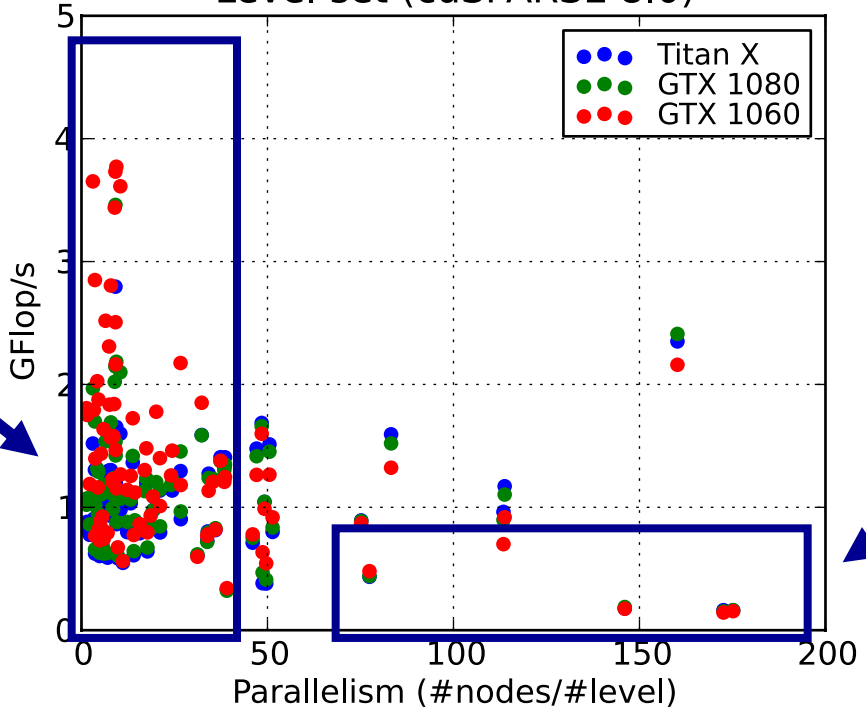
图形态



层次调度

层次集方法在GPU上的可扩展性

Level-set (cuSPARSE 8.0)



SpTRSV性能
随处理器
性能**逆向**扩展

SpTRSV性能
不随处理器
性能扩展

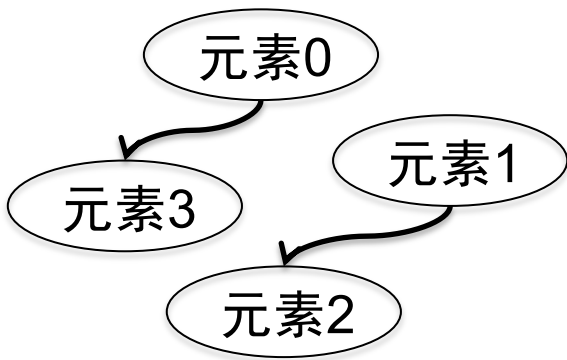
每层次
可并行元素“少”

每层次
可并行元素“多”

无同步 (Sync-Free) 方法 (我们的工作)

- 激活所有核心 (每核心负责一个解向量的元素), 一部分计算已解除依赖的元素, 一部分忙等待直至其偏序依赖解除。

1			
	1		
	2	1	
3			1



CSR “行指针” 数组:

0	1	2	4	6
---	---	---	---	---

CSC “列指针” 数组:

0	2	4	5	6
---	---	---	---	---

“入度” 数组:

1	1	2	2
---	---	---	---

“出度” 数组:

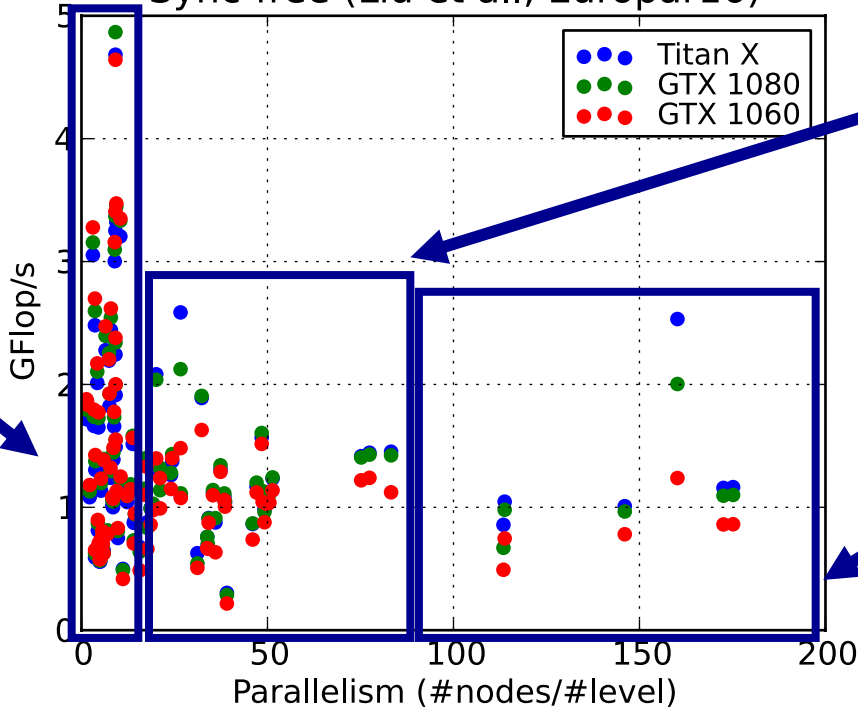
2	2	1	1
---	---	---	---

Weifeng Liu, Ang Li, Jonathan Hogg, Iain S. Duff, Brian Vinter. **Fast Synchronization-Free Algorithms for Parallel Sparse Triangular Solves with Multiple Right-Hand Sides.** *CCPE. 2017.* (extended from *Europar16*)

Weifeng Liu, Ang Li, Jonathan Hogg, Iain S. Duff, Brian Vinter. **A Synchronization-Free Algorithm for Parallel Sparse Triangular Solves.** *Europar16.*

无同步方法在GPU上的可扩展性

Sync-free (Liu et al., Europar16)



SpTRSV性能
随处理器
性能**逆向**扩展

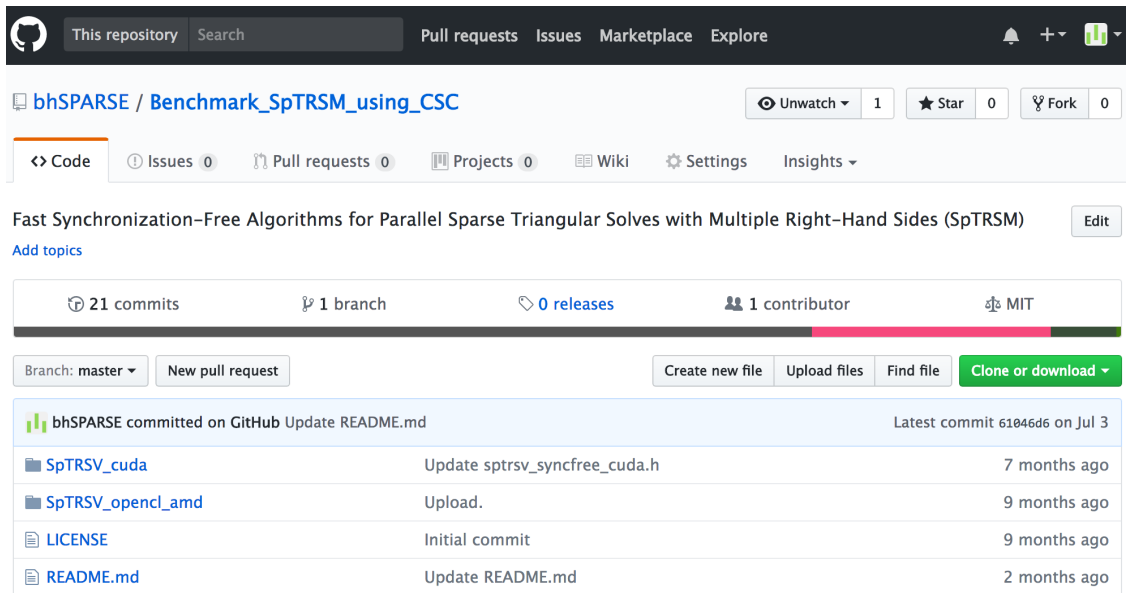
SpTRSV性能
不随处理器
性能扩展

SpTRSV性能
随处理器
性能扩展

每层次
可并行元素“少”

每层次
可并行元素“多”

无同步方法的源代码 (Github)



https://github.com/bhSPARSE/Benchmark_SpTRSM_using_CSC

该方法已被集成进UTK的MAGMA线性代数库

<https://bitbucket.org/icl/magma>



低预处理成本、高性能和可扩展 (三者最多择其一)

无同步方法



P2P (CPU) [Park et al.]

2020年综述论文

2020 年 12 月

数值计算与计算机应用

第 41 卷第 4 期

Dec., 2020

Journal on Numerical Methods and Computer Applications

Vol.41, No.4

青年评述

高可扩展、高性能和高实用的稀疏矩阵计算 研究进展与挑战^{*1)}

刘伟峰²⁾

(中国石油大学(北京)计算机科学与技术系, 超级科学软件实验室, 北京 102249)

摘 要

稀疏矩阵算法是超级计算领域的热点和难点研究内容之一。本文从高可扩展、高性能和高实用这三个角度, 对过去 30 年来国内外稀疏矩阵计算的部分主要研究工作进行了综述, 并配合在三个 GPU 上十余个稀疏 BLAS 算法的测试数据, 讨论了同时达到高可扩展、高性能和高实用这三个目标的主要难点。最后提出了未来稀疏矩阵计算领域的一系列挑战。

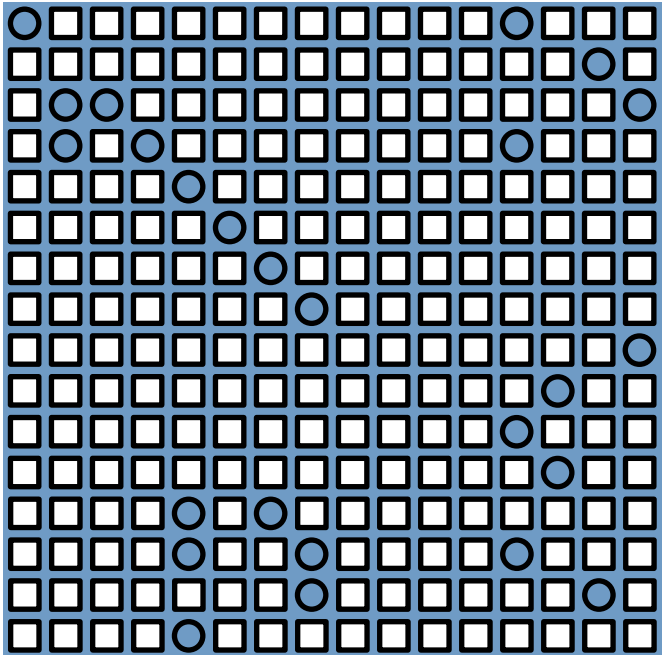
关键词: 数值线性代数; 数学软件; 稀疏矩阵计算; 高性能计算; 并行算法。

MR (2010) 主题分类: 65-xx, 68-xx.

HIGHLY-SCALABLE, HIGHLY-PERFORMANT AND

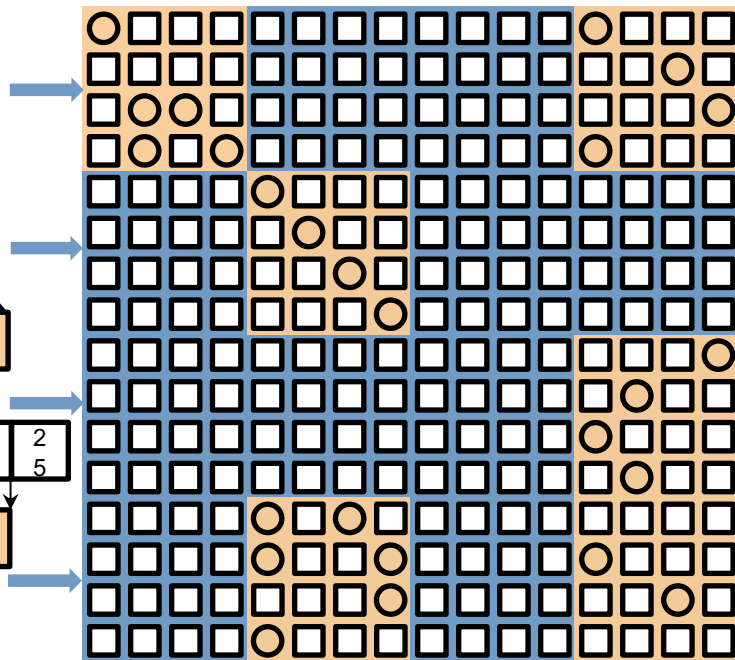
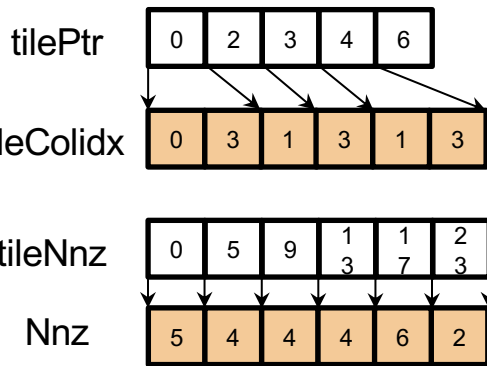
我们的最新进展1： 稀疏BLAS的分块算法和BeidouBLAS

分块数据结构



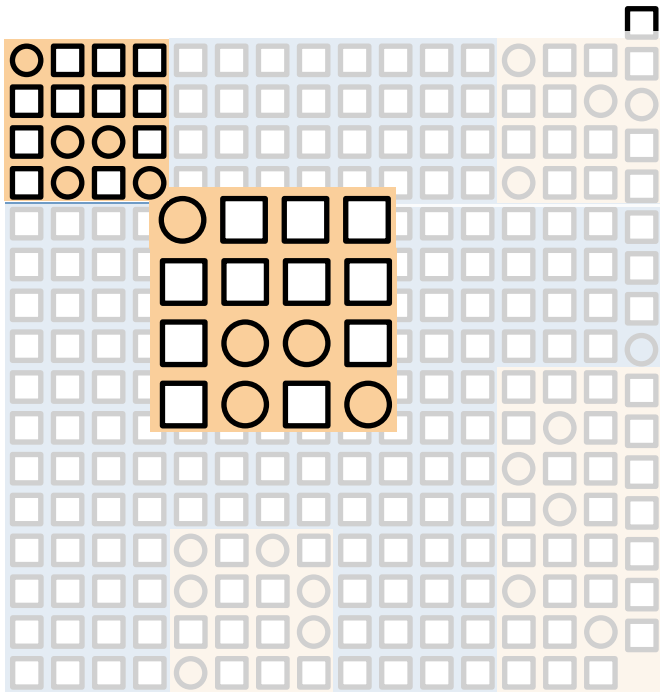
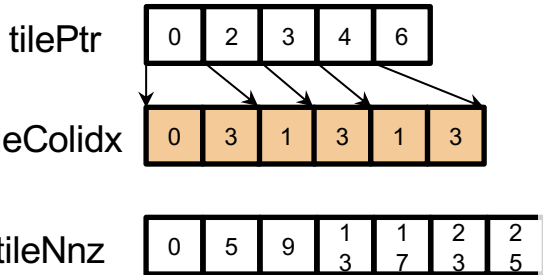
分块数据结构

higher level

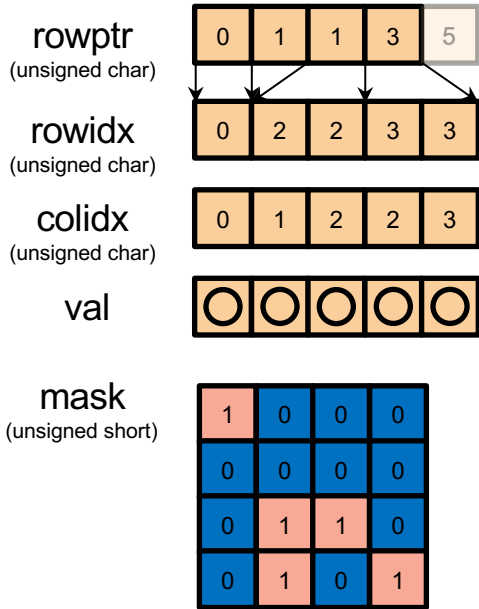


分块数据结构

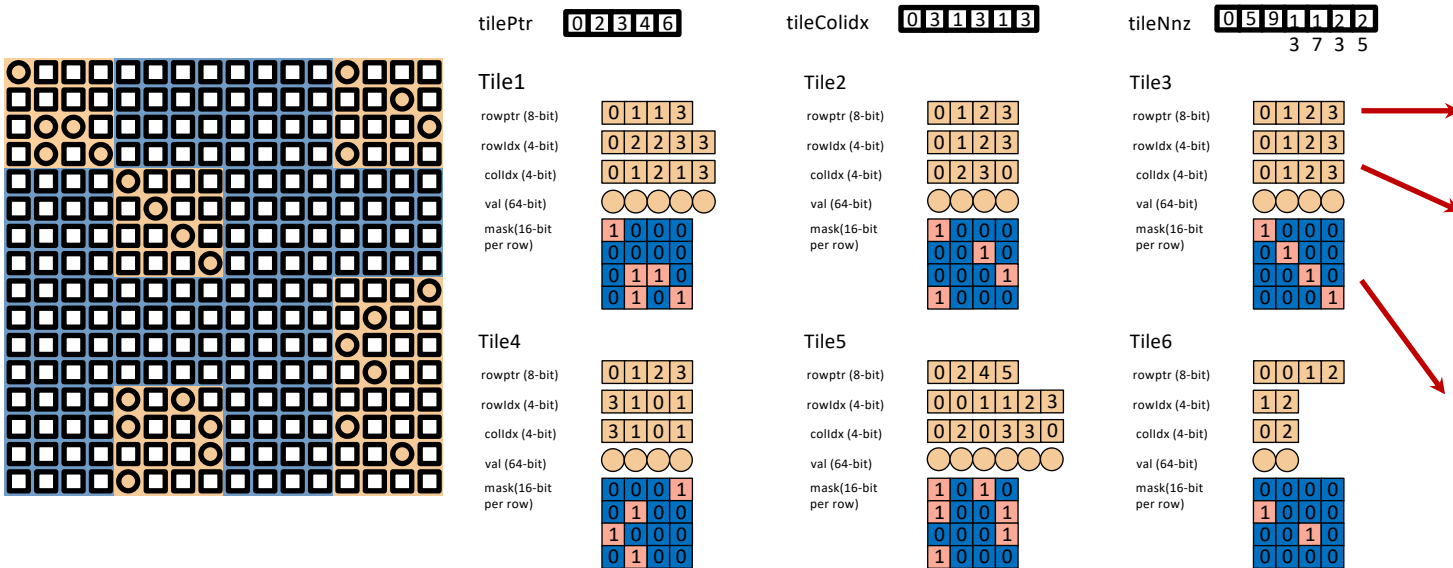
higher level



lower level



分块数据结构



8-bit unsigned char 存储行指针

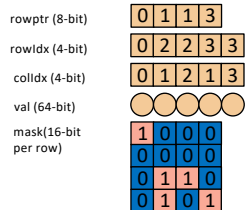
8-bit unsigned char 存储非零元的行、列索引

16-bit unsigned short 存储块位掩码

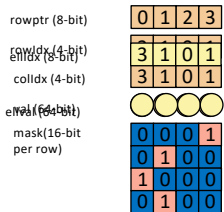
分块数据结构

tilePtr **0 2 3 4 6**

Tile1-CSR

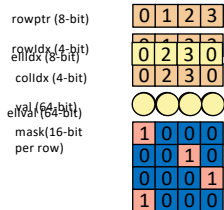


Tile4-ELL

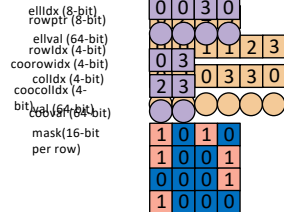


tileColidx **0 3 1 3 1 3**

Tile2-ELL

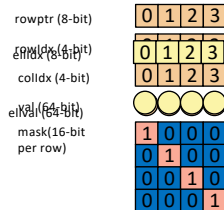


Tile5-HYB

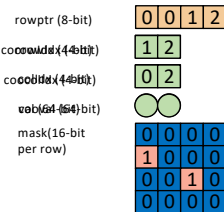


tileNnz **0 5 9 1 1 2 2**
3 7 3 5

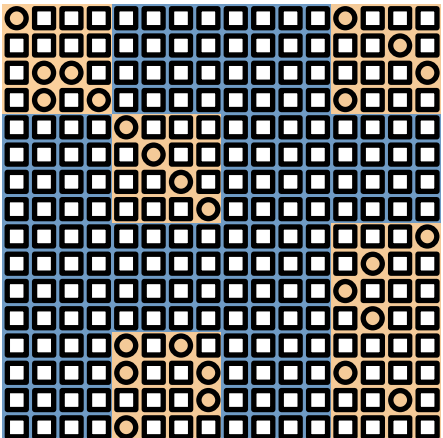
Tile3-ELL



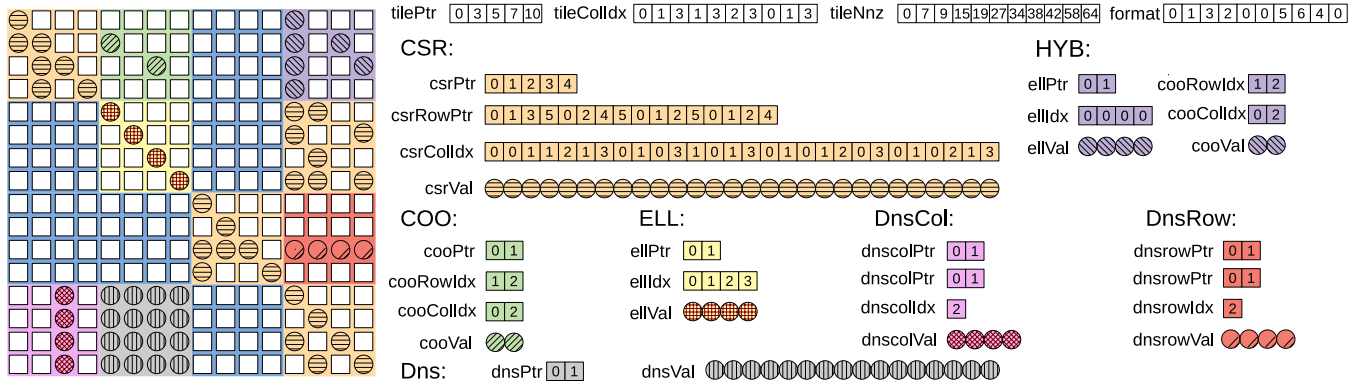
Tile6-COO



format **0 2 2 2 3 1**

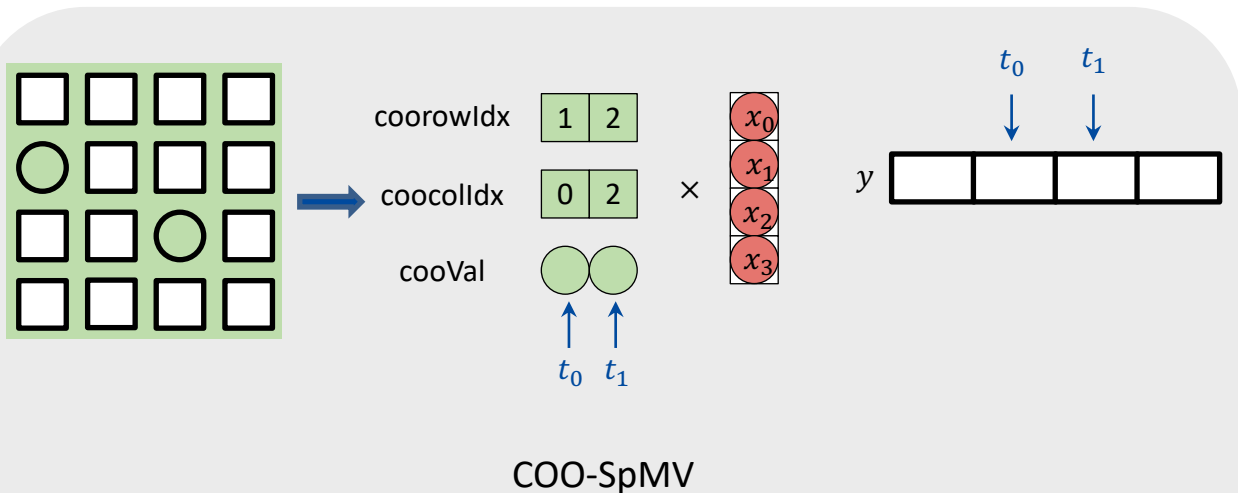


分块数据结构



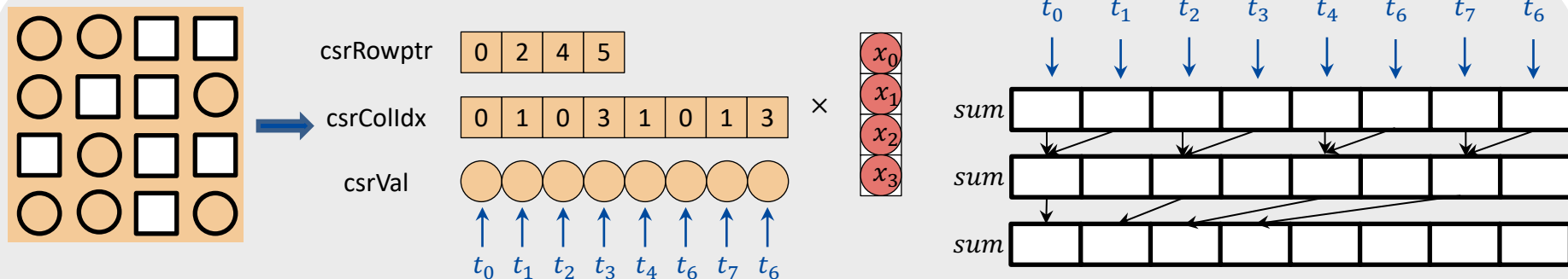
CSR
COO
ELL
HYB
Dns
DnsRow
DnsCol

TileSpMV: 分块数据结构上的SpMV算法



将一个 warp 中的 32 个线程分配处理所有非零元，并通过使用 `atomicAdd` 操作将所得的部分和添加到共享内存中。

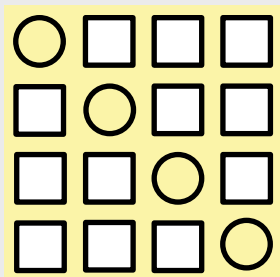
TileSpMV: 分块数据结构上的SpMV算法



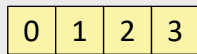
CSR-SpMV

一个32线程的warp被分配去处理16行的块，即每两个连续的线程处理一行。在进行计算之前，我们将向量 x 中16个的相应段加载到片上暂存器的共享内存中，以实现更好的数据局部性。所有线程计算结束后，将所得 y 的对应部分加在一起。

TileSpMV: 分块数据结构上的SpMV算法



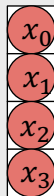
ellcolIdx



cooVal


 t_0 t_1 t_2 t_3

×



sum



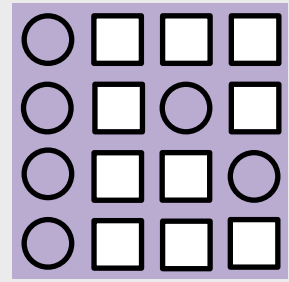
sum



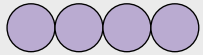
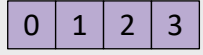
ELL-SpMV

使用 32 个线程的 warp 处理按列优先存储的非零元。一半 warp (16 线程) 中的每个线程被分配去处理一行, 当达到 ELL 宽度时, 计算完成。

TileSpMV: 分块数据结构上的SpMV算法

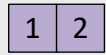


ellIdx
ellVal

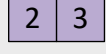


t_0 t_1 t_2 t_3

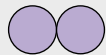
coorowIdx



coocolIdx



cooVal

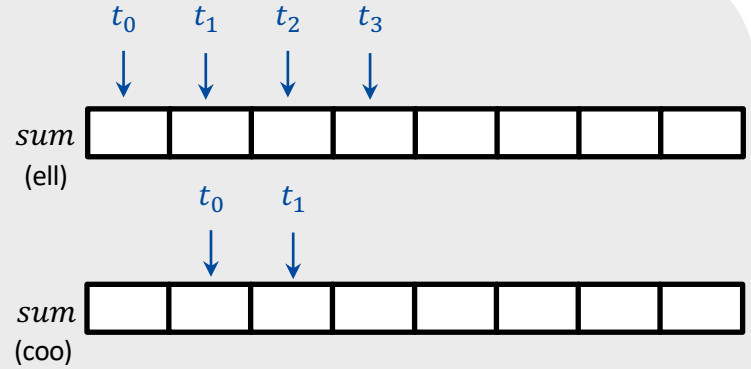


t_0 t_1

\times

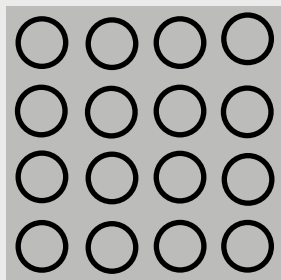


HYB-SpMV

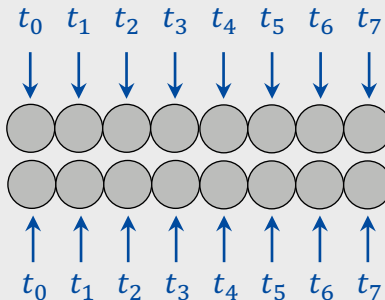


由于HYB格式是ELL和COO格式的组合，因此计算时可直接调用Warp级 ELL-SpMV 和Warp级COO-SpMV即可。

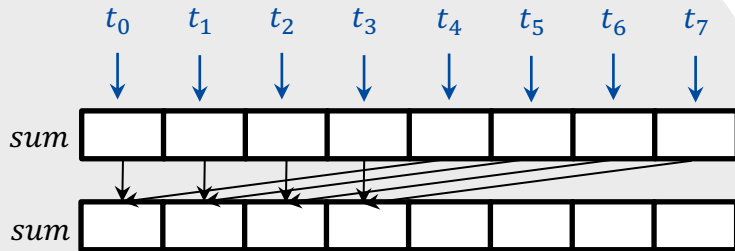
TileSpMV: 分块数据结构上的SpMV算法



dnsVal



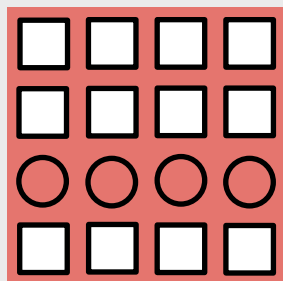
×



Dns-SpMV

一个32线程的warp需要处理一个16x16的稠密块，需要八轮完成计算，每个线程处理8个元素。计算后，结果将存储为每个线程的sum中，shuffle操作将用于加处理同一行的线程的sum值。

TileSpMV: 分块数据结构上的SpMV算法



dnsrowidx

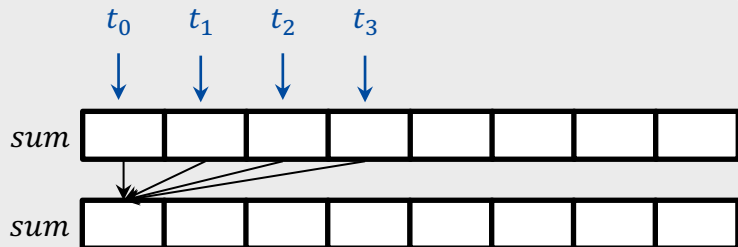
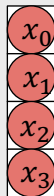
2

dnsrowVal



t_0 t_1 t_2 t_3

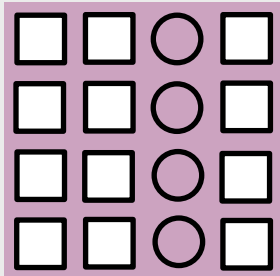
×



DnsRow-SpMV

用reduction-sum操作来计算， x 中相应部分被加载到寄存器中。

TileSpMV: 分块数据结构上的SpMV算法



dnsrowidx

2

dnsrowVal



t_0 t_1 t_2 t_3

×



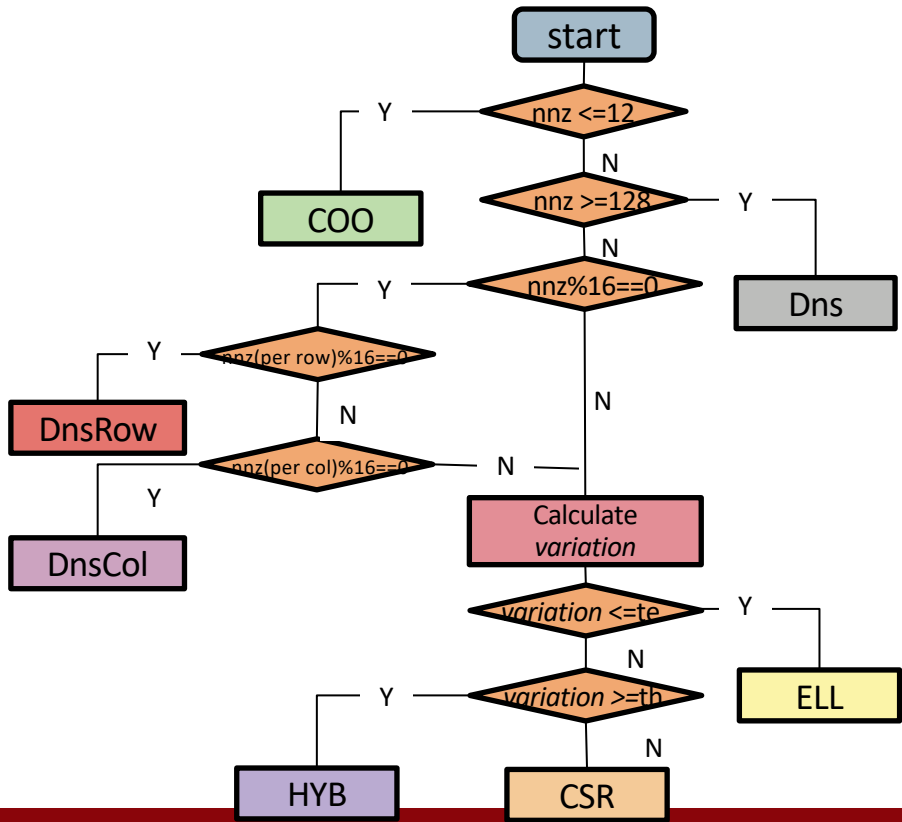
sum



DnsCol-SpMV

计算过程与Dns-SpMV相似。

TileSpMV: 分块数据结构上的SpMV算法



- COO : 非常稀疏的块($nnz \leq 12$), 且非零元在行中的分布较为分散。

- Dns : 非零元较多($nnz \geq 128$), 只记录块中元素的值。

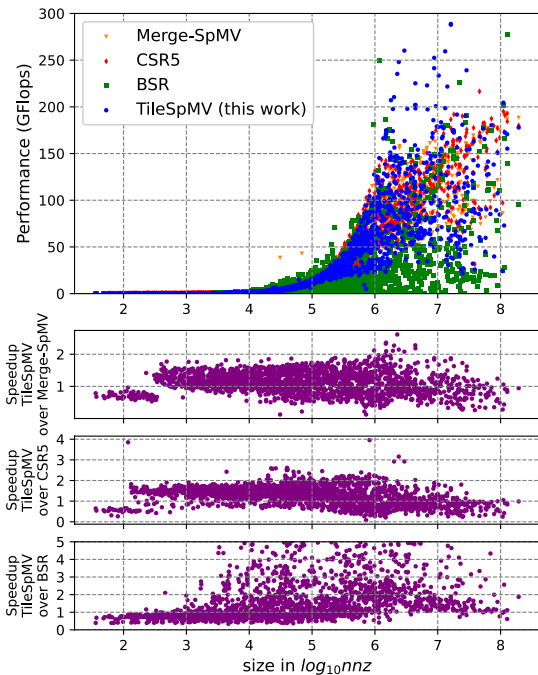
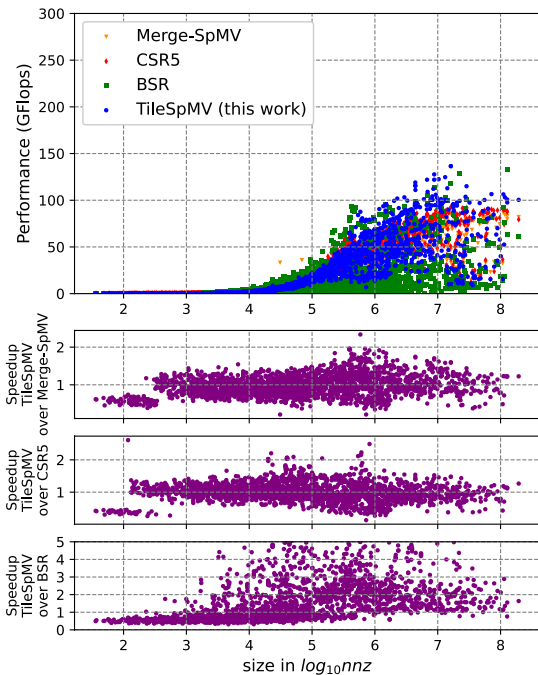
- DnsRow/DnsCol : 块中非零元具有以下分布: 所有非零元都位于特定的行/列中, 而所有其他行/列均为空。

- ELL : $0 < variation < te$

- HYB : $variation > th$

- CSR : $te < variation < th$

TileSpMV: 算法性能评估



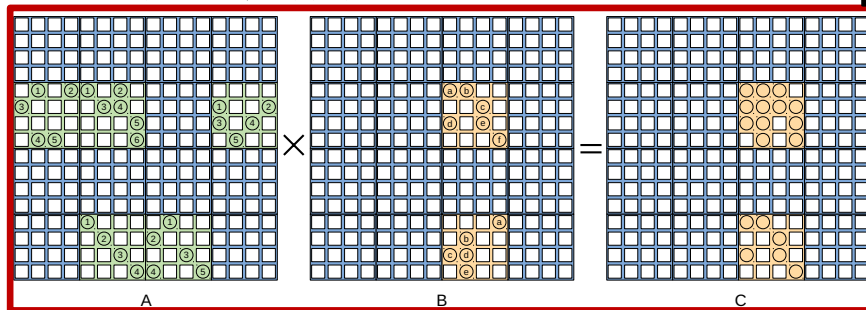
在所有2757个矩阵中:

比Merge-SpMV快的有1813个矩阵;
比CSR5快的有2040个矩阵;
比BSR快的有1638个矩阵。

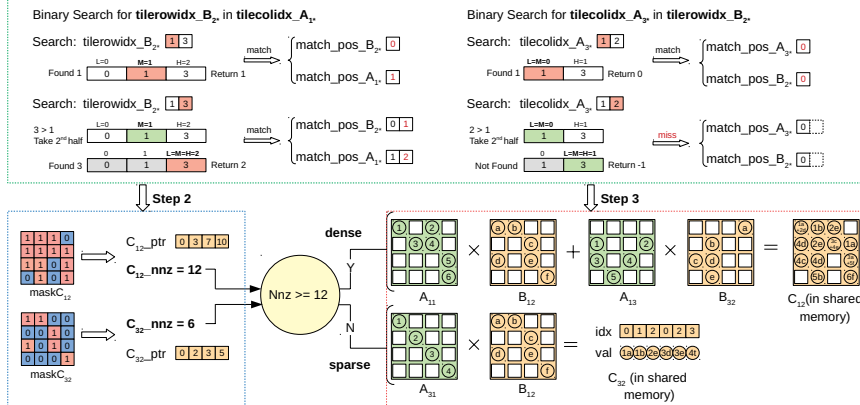
TileSpMV获得的最高加速比:
2.61x, 3.96x, 426.59x

TileSpGEMM: 分块数据结构上的SpGEMM算法

Step 1: 找出结果矩阵C中所有可能的非空块。

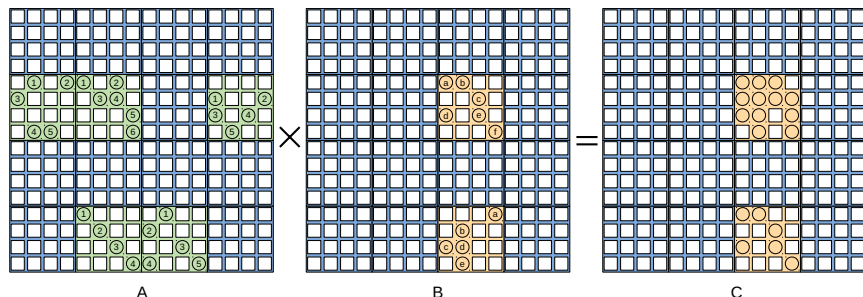


$tilePtr_C$
 $tileColidx_C$

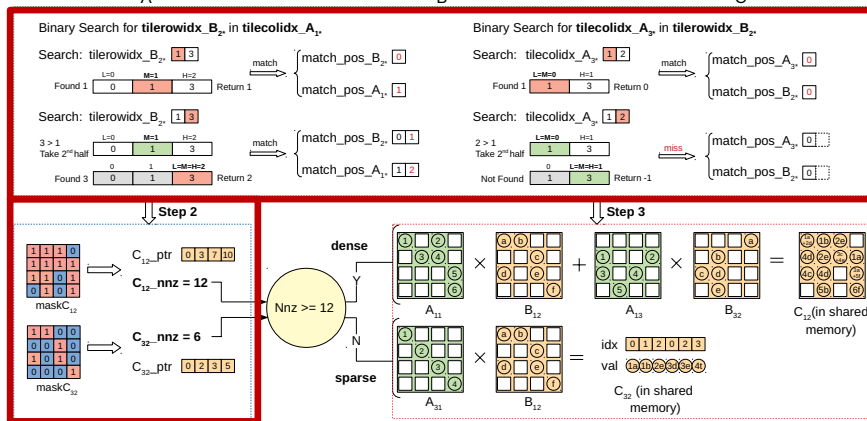


TileSpGEMM: 分块数据结构上的SpGEMM算法

Step 1: 找出结果矩阵C中所有可能的非空块。



Step 2: 确定结果矩阵C中每个块的非零数以及rowPtr数组, 并且为C分配空间。

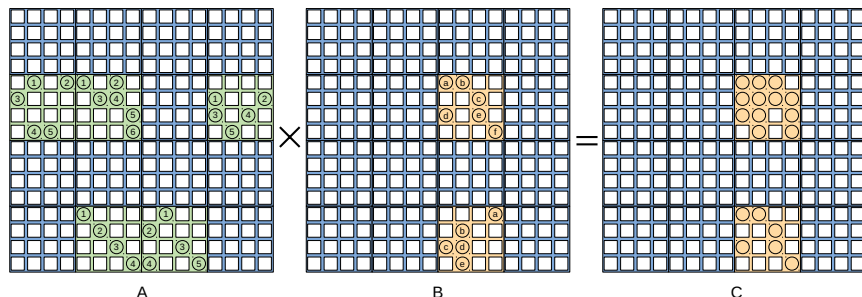


tilePtr_C
tileColidx_C

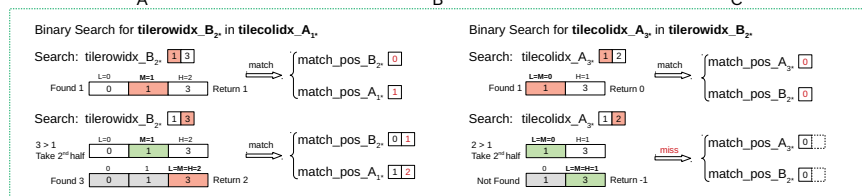
tileNnz_C
mask_C

TileSpGEMM: 分块数据结构上的SpGEMM算法

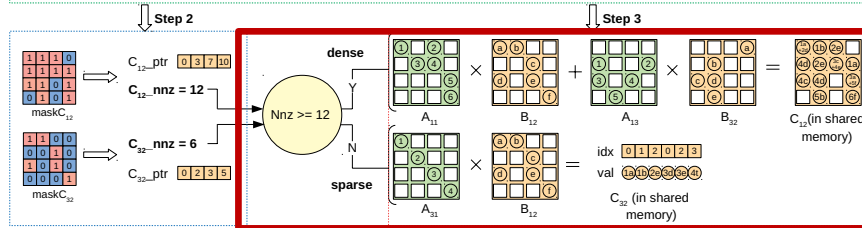
Step 1: 找出结果矩阵C中所有可能的非空块。



Step 2: 确定结果矩阵C中每个块的非零数以及rowPtr数组, 并且为C分配空间。



Step 3: 计算结果矩阵C每一个块中的非零元的值和行、列索引。



tilePtr_C
tileColidx_C

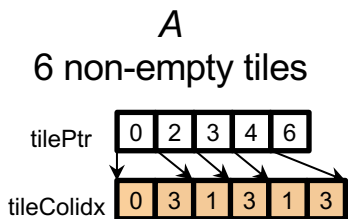
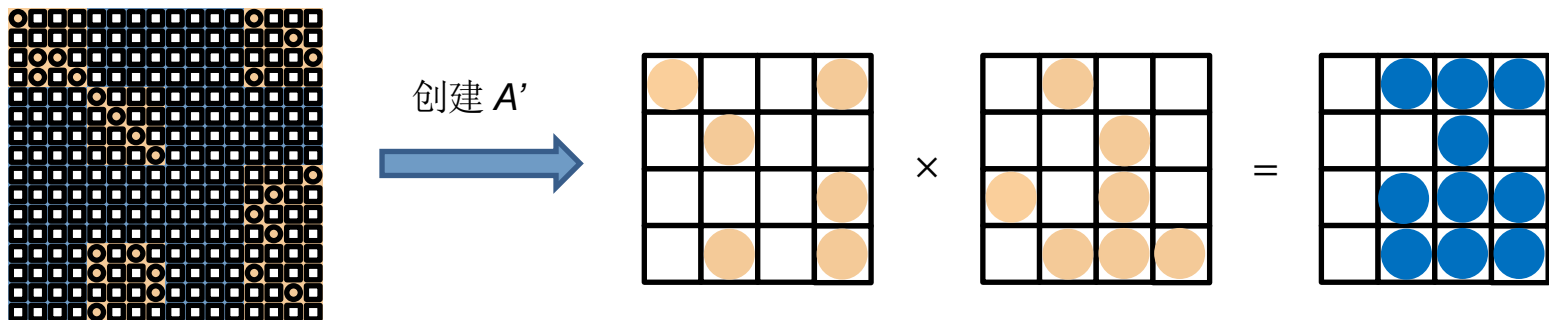
tileNnz_C
mask_C

rowidx_C
colidx_C
val_C

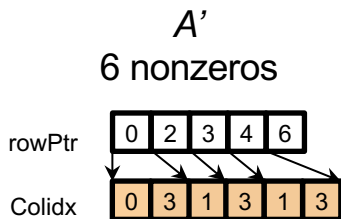
TileSpGEMM: 分块数据结构上的SpGEMM算法

Step 1

- 运行一次SpGEMM, 将两个输入矩阵A和B的高层次数据结构相乘
- 计算出的矩阵C的非零数是C中的非空块数



=



NSPARSE:

- 更简单的调用接口
- 在小矩阵上更好的性能

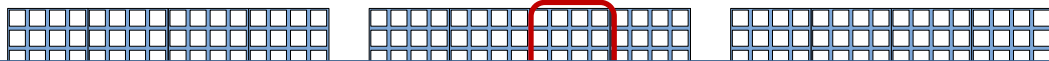
Y. Nagasaka, S. Matsuoka, A. Azad, and A. Buluç. High-Performance Sparse Matrix-Matrix Products on Intel KNL and Multicore Architectures. In Proceedings of the 47th International Conference on Parallel Processing Companion (ICPP '18), 2018, 1–10.

TileSpGEMM: 分块数据结构上的SpGEMM算法

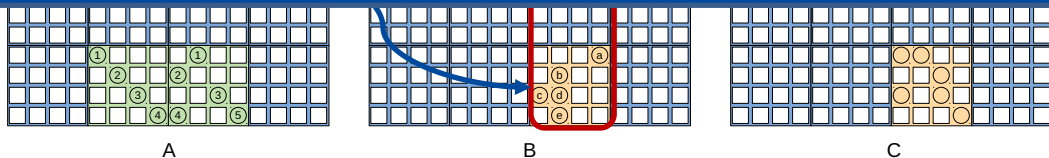
Step 2

- A 和 B 中的空块不参与计算。

如何在 A 和 B 中找到应该相乘的块？



如何生成 C 的每个非空块的行指针数组和非零元数？

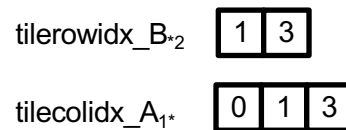


TileSpGEMM: 分块数据结构上的SpGEMM算法

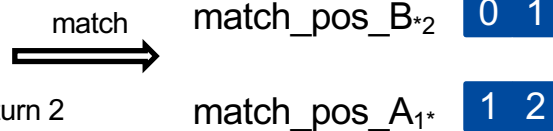
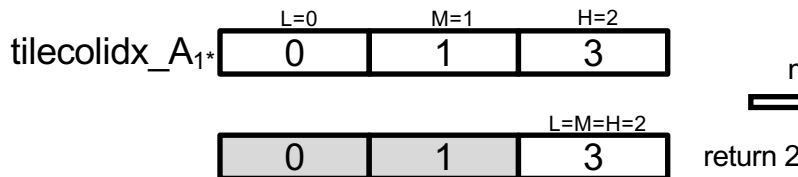
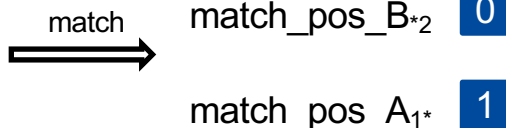
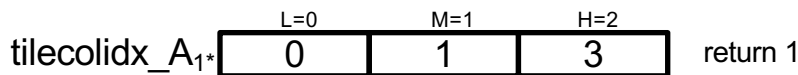
Step 2

- 二分查找获得集合相交结果：在较长的索引数组中逐个检索较短索引数组中的每一个元素。

Binary Search $tilerowidx_B_{*2}$ in $tilecolidx_A_{1*}$

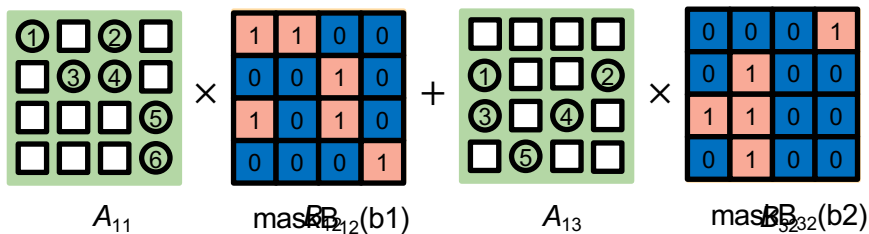


Search: $tilerowidx_B_{*2}$

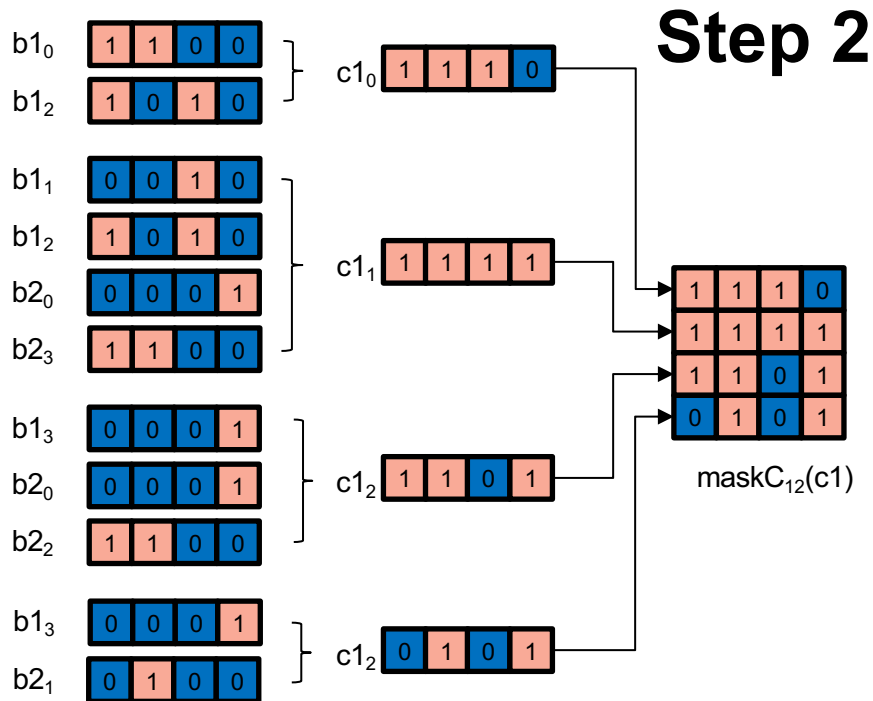


TileSpGEMM: 分块数据结构上的SpGEMM算法

- 符号SpGEMM的位掩码操作: 使用每个块的 **mask** 数组来减少在 B 中重复加载块结构信息的内存传输成本。



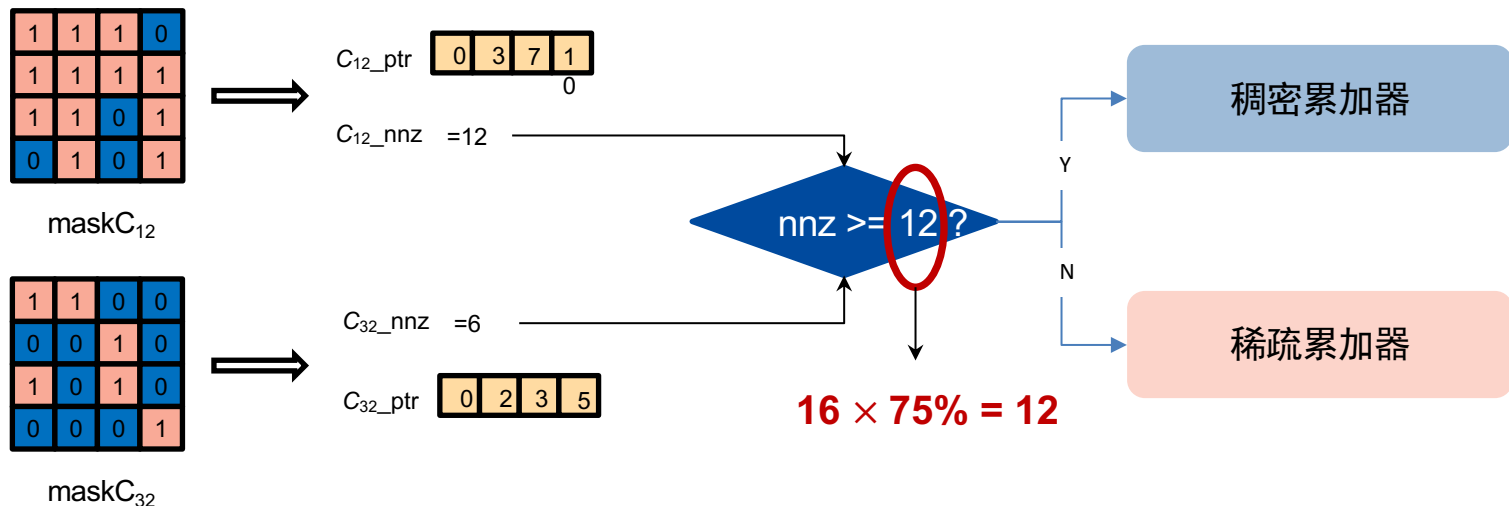
- 在暂存器上对所有提取出的行掩码做 **AtomicOR** 操作



TileSpGEMM: 分块数据结构上的SpGEMM算法

Step 3

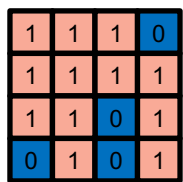
- 在片上存储器中选择稀疏或稠密累加器的自适应方法



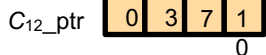
TileSpGEMM: 分块数据结构上的SpGEMM算法

Step 3

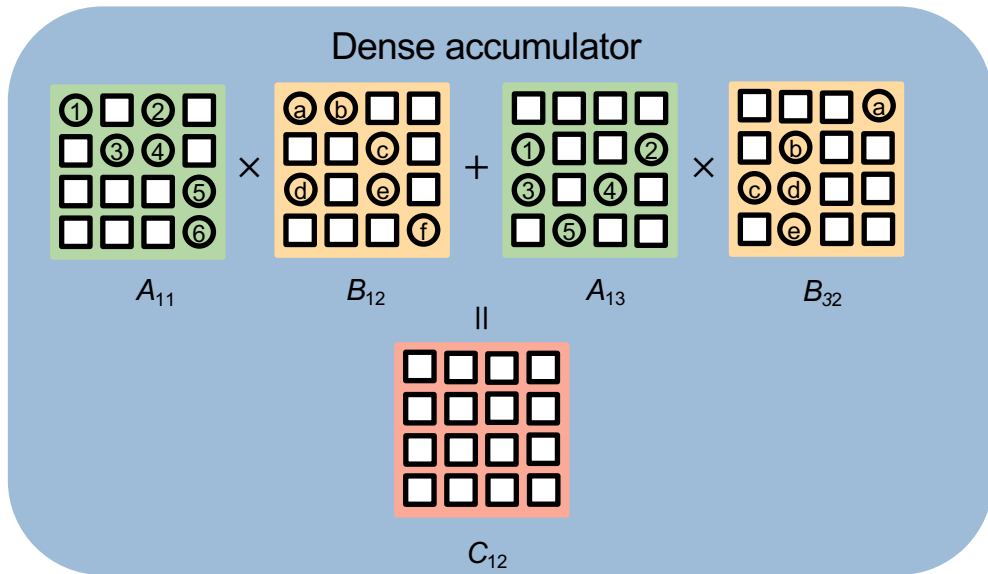
- 若块相对较稠密($\frac{\text{nnz}}{\text{tile-size}} \geq 0.75$), 则选择稠密累加器, 直接在片上共享内存上开辟一个稠密的空间(即256), 并在上面进行运算, 最后将计算结果存为稀疏的格式



maskC₁₂



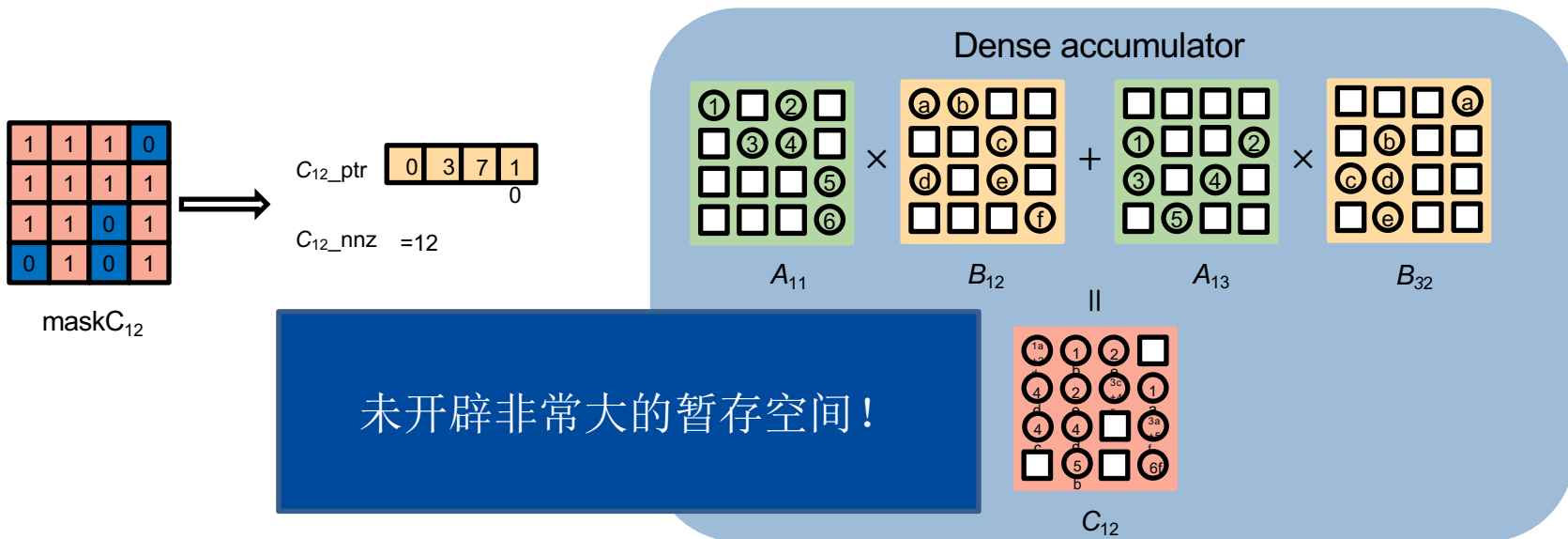
C₁₂_{nnz} = 12



TileSpGEMM: 分块数据结构上的SpGEMM算法

Step 3

- 若块相对较稠密($\frac{\text{nnz}}{\text{tile-size}} \geq 0.75$), 则选择稠密累加器, 直接在片上共享内存上开辟一个稠密的空间(即256), 并在上面进行运算, 并将最后将计算结果存为稀疏的格式



TileSpGEMM: 分块数据结构上的SpGEMM算法

Step 3

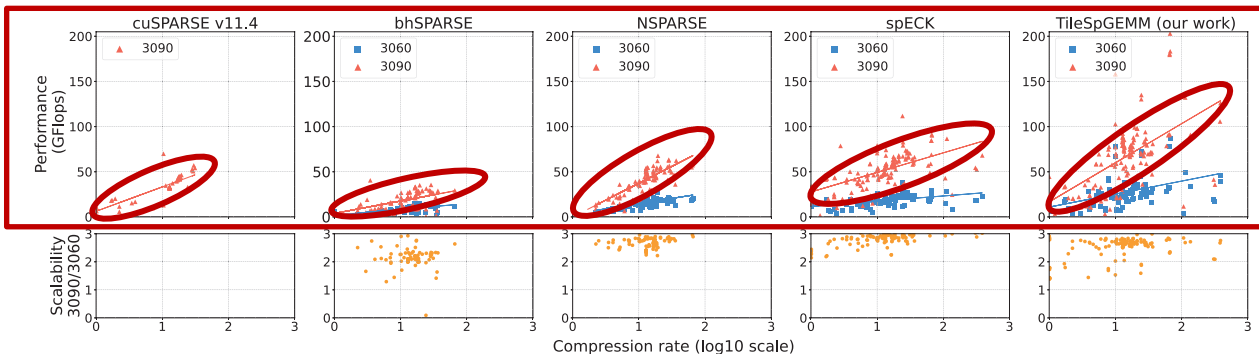
- 若块相对较稀疏($\frac{\text{nnz}}{\text{tile-size}} < 0.75$), 则选择稀疏累加器, 通过第二步中得到的C的块掩码来确定非零元的列索引, 并在相应位置累加计算其value值



未开辟非常大的暂存空间!

TileSpGEMM：算法性能评估

Comparison: $C = A^2$ and $C = AA^T$ (double precision)



spECK和TileSpGEMM可以计算出全部142个矩阵

相比于cuSPARSE, bhSPARSE, NSPARSE 和 spECK, TileSpGEMM在RTX 3090上分别在139, 138, 127 以及 94个矩阵上比其他四种方法快, 在RTX 3060上分别在142, 128, 114 和 92个矩阵上比其他方法快。

TileSpGEMM获得的平均加速比（几何平均）：1.77x, 4.73x, 1.45x, 1.16x
 最高加速比：2.78x, 145.35x, 97.86x, 3.70x

更多细节请参看相关论文

TileSpMV: A Tiled Algorithm for Sparse Matrix-Vector Multiplication on GPUs

Yuyao Niu, Zhengyang Lu, Meichen Dong, Zhou Jin, Weifeng Liu, Guangming Tan.

35th IEEE International Parallel and Distributed Processing Symposium (IPDPS'21), 2021.

<https://github.com/SuperScientificSoftwareLaboratory/TileSpMV>

<https://youtu.be/LcP82bJRro>

TileSpGEMM: A Tiled Algorithm for Parallel Sparse General Matrix-Matrix Multiplication on GPUs

Yuyao Niu, Zhengyang Lu, Haonan Ji, Shuhui Song, Zhou Jin, Weifeng Liu.

27th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming (PPoPP '22). 2022.

<https://github.com/SuperScientificSoftwareLaboratory/TileSpGEMM>

<https://youtu.be/FTMkZbFNSqI>

TileSpMSpV: A Tiled Algorithm for Sparse Matrix-Sparse Vector Multiplication on GPUs

Haonan Ji, Huimin Song, Shibo Lu, Zhou Jin, Guangming Tan, Weifeng Liu.

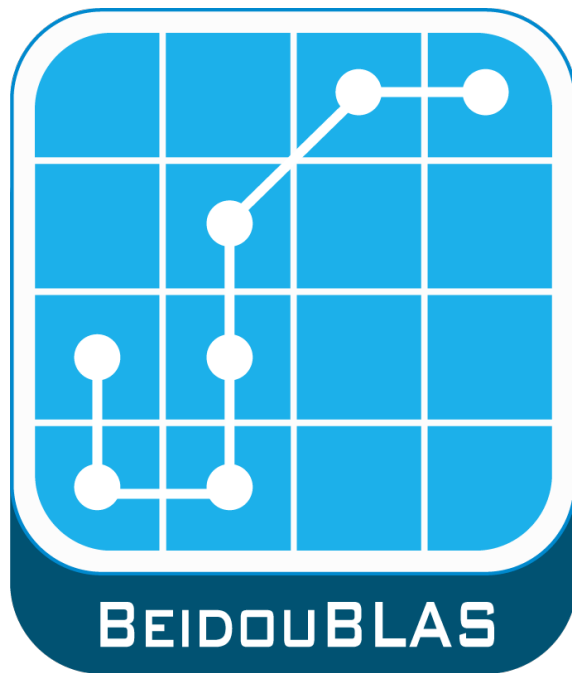
51st International Conference on Parallel Processing (ICPP '22). 2022.

Open source code and video will be online soon.

BeidouBLAS: 一个分块的开源稀疏BLAS软件包

BeidouBLAS是一个基于现代并行处理器的分块稀疏基本线性代数子程序库。它在CPU和GPU上都提供了高性能的C语言接口。

BeidouBLAS将广泛应用于机器学习、GpaphBLAS等领域，特别是在需要调用各种稀疏矩阵计算的高层次应用中。



<https://gitee.com/ssslab/beidoublas>

我们的新进展2： 利用稀疏BLAS的稀疏LU软件包PanguLU

稀疏LU分块算法

Algorithm LU Factorization

```

1. for k = 1, ..., p do
    //LU分解
2.     DGETRF( $A_{kk}, L_{kk}, U_{kk}$ ) →  $DGETRF: A_{kk} \rightarrow L_{kk}, U_{kk} = LU(A_{kk})$ 
3.     for j = k+1, ..., p do
4.         //上三角矩阵块的三角解
5.         DGESSM( $A_{kj}, L_{kk}, U_{kj}$ ) →  $DGESSM: A_{kj}, L_{kk} \rightarrow U_{kj} = L_{kk}^{-1}A_{kj}$ 
6.     end for
7.     //下三角矩阵块的三角解
8.     for i = k+1, ..., p do
9.         //更新子矩阵块 $A_{ij}$ 
10.        DSSSSM( $A_{ij}, L_{ik}, U_{kj}$ ) →  $DSSSSM: A_{ij}, L_{ik}, U_{kj} \rightarrow A_{ij} = A_{ij} - L_{ik}U_{kj}$ 
11.    end for
12. end for
    
```

稀疏LU分块算法

Algorithm LU Factorization

```

1. for k = 1, ..., p do
2.     DGETRF( $A_{kk}, L_{kk}, U_{kk}$ ) //LU分解
3.     for j = k+1, ..., p do
4.         DGESSM( $A_{kj}, L_{kk}, U_{kj}$ ) //上三角解
5.     end for
6.     for i = k+1, ..., p do
7.         DTSTRF( $A_{ik}, L_{ik}, U_{kk}$ ) //下三角解
8.         for j = k+1, ..., i do
8.             DSSSSM( $A_{ij}, L_{ik}, U_{kj}$ ) //更新子矩阵块 $A_{ij}$ 
9.         end for
10.    end for
11. end for
    
```



DGETRF



DGESSM



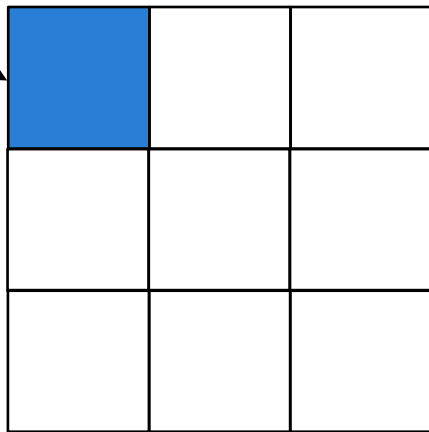
DSSSSM



DTSTRF



DONE



稀疏LU分块算法

Algorithm LU Factorization

```

1. for k = 1, ..., p do
2.     DGETRF( $A_{kk}, L_{kk}, U_{kk}$ ) //LU分解
3.     for j = k+1, ..., p do
4.         DGESSM( $A_{kj}, L_{kk}, U_{kj}$ ) //上三角解
5.     end for
6.     for i = k+1, ..., p do
7.         DTSTRF( $A_{ik}, L_{ik}, U_{kk}$ ) //下三角解
8.         for j = k+1, ..., i do
9.             DSSSSM( $A_{ij}, L_{ik}, U_{kj}$ ) //更新子矩阵块 $A_{ij}$ 
10.        end for
11.    end for
12. end for
    
```



DGETRF



DGESSM



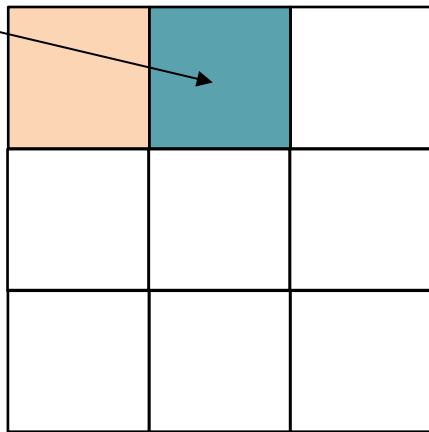
DSSSSM



DTSTRF



DONE

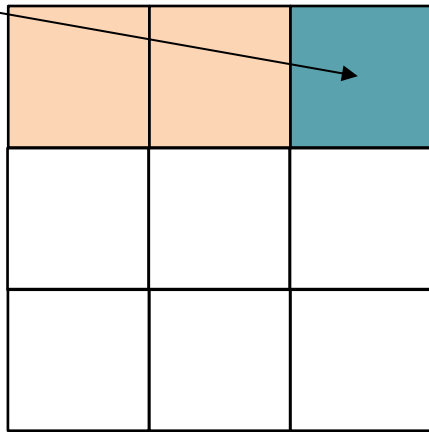


稀疏LU分块算法

Algorithm LU Factorization

```

1. for k = 1, ..., p do
2.     DGETRF( $A_{kk}, L_{kk}, U_{kk}$ ) //LU分解
3.     for j = k+1, ..., p do
4.         DGESSM( $A_{kj}, L_{kk}, U_{kj}$ ) //上三角解
5.     end for
6.     for i = k+1, ..., p do
7.         DTSTRF( $A_{ik}, L_{ik}, U_{kk}$ ) //下三角解
8.         for j = k+1, ..., i do
8.             DSSSSM( $A_{ij}, L_{ik}, U_{kj}$ ) //更新子矩阵块 $A_{ij}$ 
9.         end for
10.    end for
11. end for
    
```



稀疏LU分块算法

Algorithm LU Factorization

```

1. for k = 1, ..., p do
2.     DGETRF( $A_{kk}, L_{kk}, U_{kk}$ ) //LU分解
3.     for j = k+1, ..., p do
4.         DGESSM( $A_{kj}, L_{kk}, U_{kj}$ ) //上三角解
5.     end for
6.     for i = k+1, ..., p do
7.         DTSTRF( $A_{ik}, L_{ik}, U_{kk}$ ) //下三角解
8.         for j = k+1, ..., i do
8.             DSSSSM( $A_{ij}, L_{ik}, U_{kj}$ ) //更新子矩阵块 $A_{ij}$ 
9.         end for
10.    end for
11. end for

```



DGETRF



DGESSM



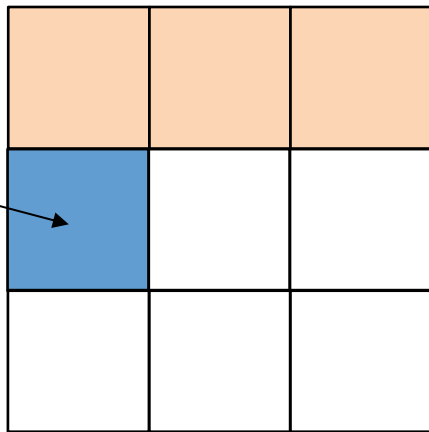
DSSSSM



DTSTRF



DONE



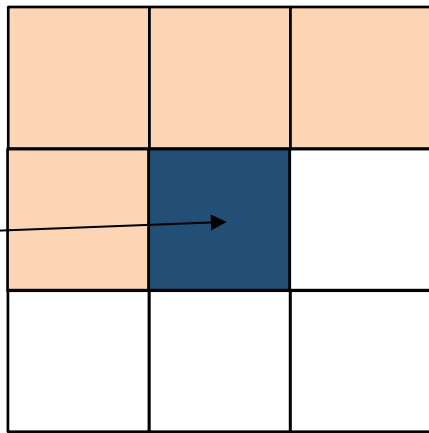
稀疏LU分块算法

Algorithm LU Factorization

```

1. for k = 1, ..., p do
2.     DGETRF( $A_{kk}, L_{kk}, U_{kk}$ ) //LU分解
3.     for j = k+1, ..., p do
4.         DGESSM( $A_{kj}, L_{kk}, U_{kj}$ ) //上三角解
5.     end for
6.     for i = k+1, ..., p do
7.         DTSTRF( $A_{ik}, L_{ik}, U_{kk}$ ) //下三角解
8.         for j = k+1, ..., i do
8.             DSSSSM( $A_{ij}, L_{ik}, U_{kj}$ ) //更新子矩阵块 $A_{ij}$ 
9.         end for
10.    end for
11. end for

```

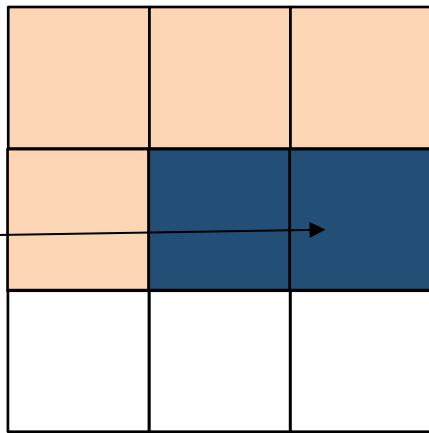


稀疏LU分块算法

Algorithm LU Factorization

```

1. for k = 1, ..., p do
2.     DGETRF( $A_{kk}, L_{kk}, U_{kk}$ ) //LU分解
3.     for j = k+1, ..., p do
4.         DGESSM( $A_{kj}, L_{kk}, U_{kj}$ ) //上三角解
5.     end for
6.     for i = k+1, ..., p do
7.         DTSTRF( $A_{ik}, L_{ik}, U_{kk}$ ) //下三角解
8.         for j = k+1, ..., i do
8.             DSSSSM( $A_{ij}, L_{ik}, U_{kj}$ ) //更新子矩阵块 $A_{ij}$ 
9.         end for
10.    end for
11. end for
    
```



稀疏LU分块算法

Algorithm LU Factorization

```

1. for k = 1, ..., p do
2.     DGETRF( $A_{kk}, L_{kk}, U_{kk}$ ) //LU分解
3.     for j = k+1, ..., p do
4.         DGESSM( $A_{kj}, L_{kk}, U_{kj}$ ) //上三角解
5.     end for
6.     for i = k+1, ..., p do
7.         DTSTRF( $A_{ik}, L_{ik}, U_{kk}$ ) //下三角解
8.         for j = k+1, ..., i do
8.             DSSSSM( $A_{ij}, L_{ik}, U_{kj}$ ) //更新子矩阵块 $A_{ij}$ 
9.         end for
10.    end for
11. end for
    
```



DGETRF



DGESSM



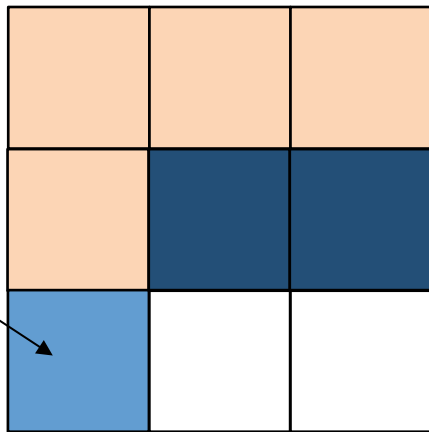
DSSSSM



DTSTRF



DONE

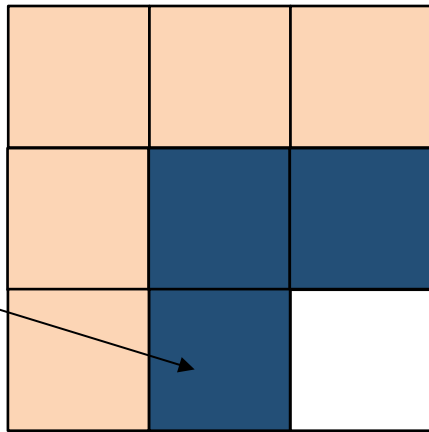


稀疏LU分块算法

Algorithm LU Factorization

```

1. for k = 1, ..., p do
2.     DGETRF( $A_{kk}, L_{kk}, U_{kk}$ ) //LU分解
3.     for j = k+1, ..., p do
4.         DGESSM( $A_{kj}, L_{kk}, U_{kj}$ ) //上三角解
5.     end for
6.     for i = k+1, ..., p do
7.         DTSTRF( $A_{ik}, L_{ik}, U_{kk}$ ) //下三角解
8.         for j = k+1, ..., i do
8.             DSSSSM( $A_{ij}, L_{ik}, U_{kj}$ ) //更新子矩阵块 $A_{ij}$ 
9.         end for
10.    end for
11. end for
    
```

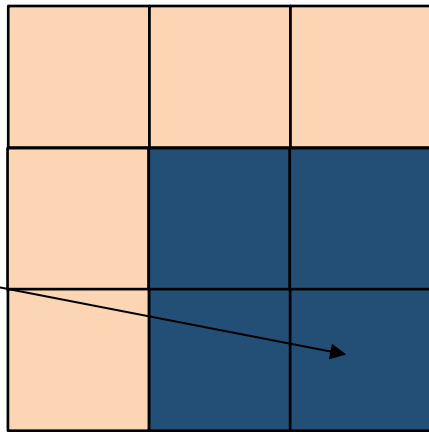


稀疏LU分块算法

Algorithm LU Factorization

```

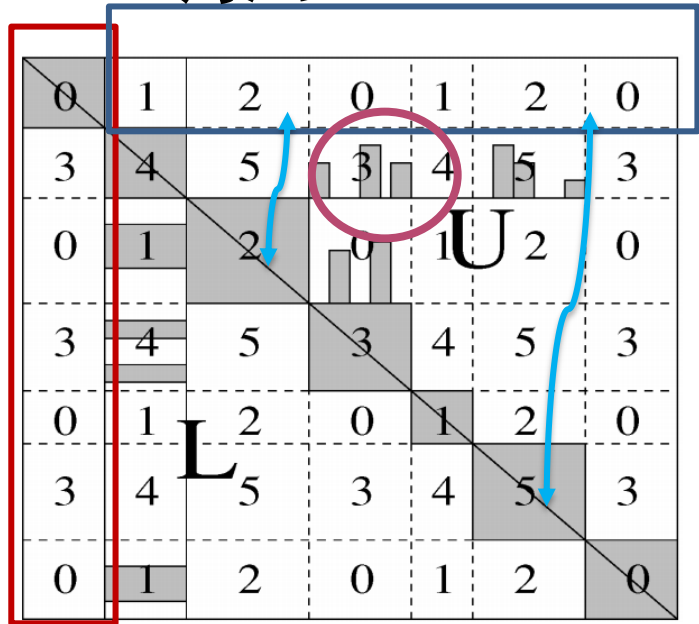
1. for k = 1, ..., p do
2.     DGETRF( $A_{kk}, L_{kk}, U_{kk}$ ) //LU分解
3.     for j = k+1, ..., p do
4.         DGESSM( $A_{kj}, L_{kk}, U_{kj}$ ) //上三角解
5.     end for
6.     for i = k+1, ..., p do
7.         DTSTRF( $A_{ik}, L_{ik}, U_{kk}$ ) //下三角解
8.         for j = k+1, ..., i do
8.             DSSSSM( $A_{ij}, L_{ik}, U_{kj}$ ) //更新子矩阵块 $A_{ij}$ 
9.         end for
10.    end for
11. end for
    
```



SuperLU分块算法和稠密BLAS调用

```

for block  $K = 1$  to  $N$  do
  (1) if [  $me \in PROC_C(K)$  ] then
    Factorize block column  $L(K : N, K)$ 
    Send  $L(K : N, K)$  to the processes in my row who need it
  else
    Receive  $L(K : N, K)$  from one process in  $PROC_C(K)$ 
  endif
  (2) if [  $me \in PROC_R(K)$  ] then
    Factorize block row  $U(K, K + 1 : N)$ 
    Send  $U(K, K + 1 : N)$  to processes in my column who need it
  else
    Receive  $U(K, K + 1 : N)$  from one process in  $PROC_R(K)$  if I need it
  endif
  (3) for  $J = K + 1$  to  $N$  do
    for  $I = K + 1$  to  $N$  do
      if [  $me \in PROC_R(I)$  and  $me \in PROC_C(J)$ 
        and  $L(I, K) \neq 0$  and  $U(K, J) \neq 0$  ] then
        Update trailing submatrix  $A(I, J) \leftarrow A(I, J) - L(I, K) \cdot U(K, J)$ 
      endif
    end for
  end for
end for
    
```



SuperLU将超级节点策略引入LU分解，通过右视算法、稠密GEMM和2D模型扩展到分布式系统。

Li, X. S., & Demmel, J. W. (2003). SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. ACM Transactions on Mathematical Software (TOMS), 29(2), 110-140.

PanguLU: 一个异构分布式开源稀疏LU软件包

PanguLU是一个面向异构分布式平台的开源直接法解法器。该解法器使用C语言进行编写，使用MPI和CUDA完成并行架构。

PanguLU在节点内调用优化的稀疏BLAS进行块内的计算，SpTRSV、SpGEMM和稀疏LU在很多情况下能比稠密计算带来更高的效率。



<https://gitee.com/ssslab/pangulu>

PanguLU面临的挑战

该解法器不同于之前的工作是矩阵块之间的计算kernel均采用**稀疏的方式计算**，所以需要设计**稀疏kernel**。

在设计好稀疏kernel以后我们还需要设计相应的**分布式通信策略**来完成 $A = L U$ 的矩阵分解，为了加速计算，还对其中的一部分**稀疏kernel**针对GPU的计算架构进行优化。

在完成LU分解后，只是完成了 $A = L U$ ，但是整个 $Ax = b$ 未求解出 x ，所以需要**一个分布式三角解**来完成 $LUx = b$ 。

稀疏kernel



GETRF

稀疏LU分解：一个稀疏矩阵A分解成一个下三角矩阵L和一个上三角矩阵U



GESSM

稀疏上三角解：一个上三角稀疏矩阵L与一个稀疏矩阵A进行n次三角解，并将所得到的n维向量组成一个稀疏矩阵x



TSTRF

稀疏下三角解：一个下三角稀疏矩阵U与一个稀疏矩阵A进行n次三角解，并将所得到的n维向量组成一个稀疏矩阵x



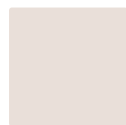
SSSSM

稀疏矩阵矩阵乘：两个稀疏矩阵A和B进行相乘并且将结果加在稀疏矩阵C上

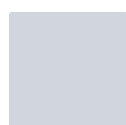
稀疏kernel



GETRF



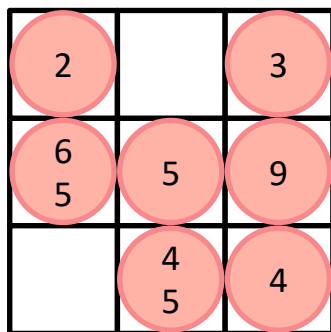
GESSM



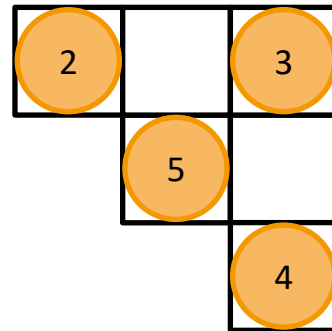
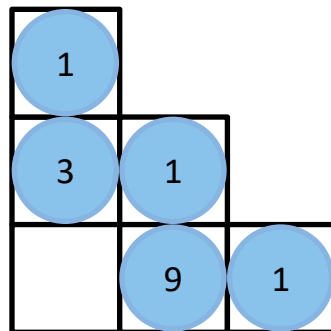
TSTRF



SSSSM



=



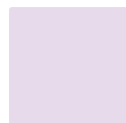
Function GETRF

Rowptr	0 2 5 7	0 1 3 5	0 2 3 4
Colindex	0 2 0 1 2 1 2	0 0 1 1 2	0 2 1 2
val	2 3 6 5 9 4 4	1 3 1 9 1	2 3 5 4

$$A = L U$$

该kernel是将稀疏矩阵A拆分为稀疏下三角矩阵L与稀疏上三角矩阵U

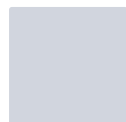
稀疏kernel



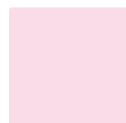
GETRF



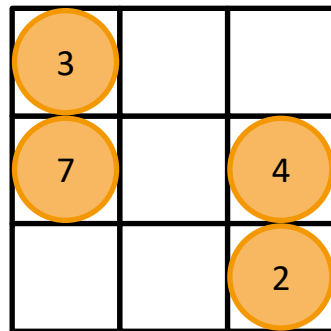
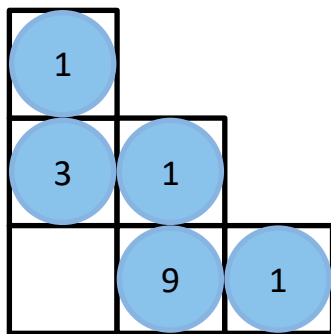
GESSM



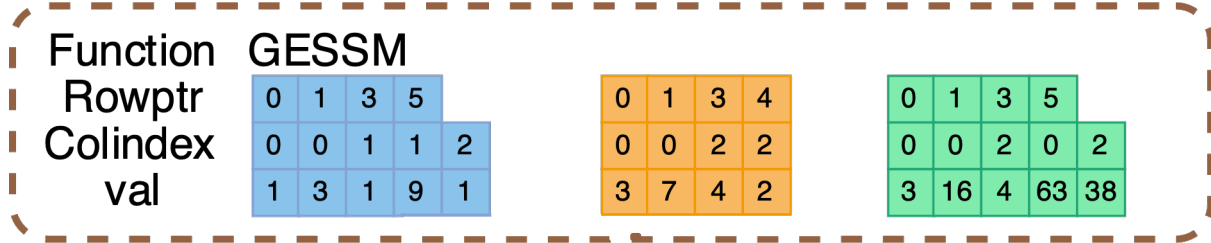
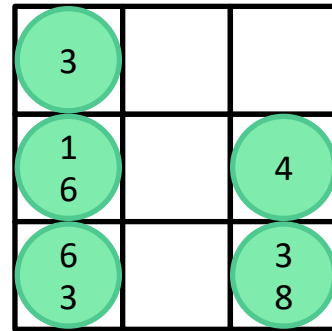
TSTRF



SSSSM



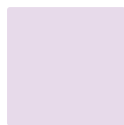
=



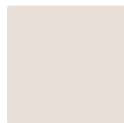
$$L X = A$$

该kernel是使用稀疏下三角矩阵L对稀疏矩阵A中的多个列向量进行下三角解，再将求解出来的向量重组为稀疏矩阵X

稀疏kernel



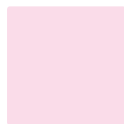
GETRF



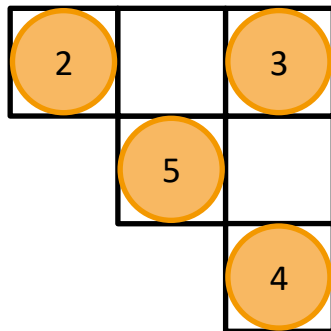
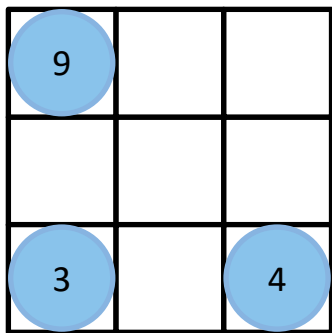
GESSM



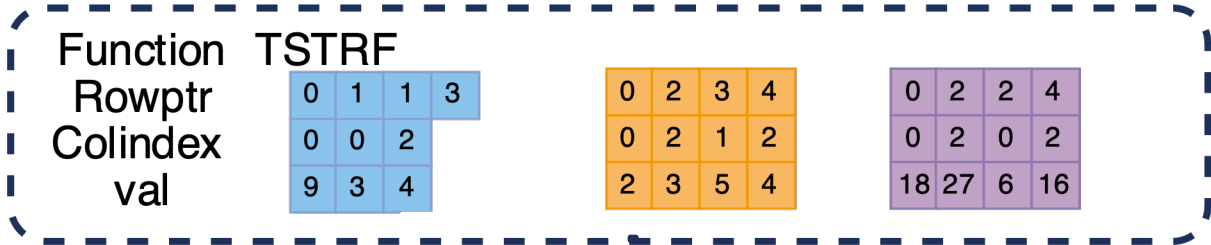
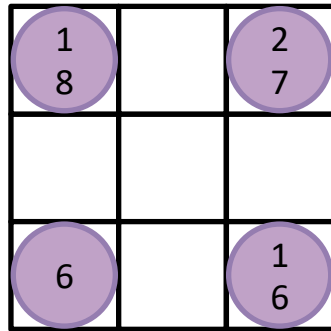
TSTRF



SSSSM



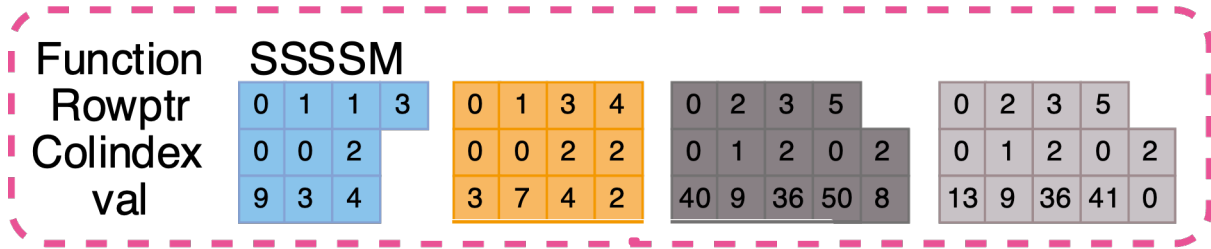
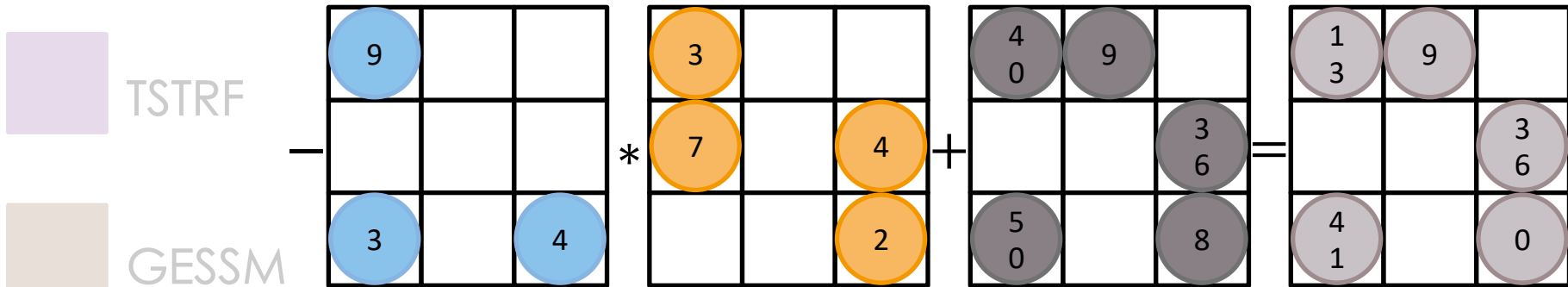
=



$$X U = A$$

该kernel是使用稀疏上三角矩阵U对稀疏矩阵A中的多个列向量进行上三角解，再将求解出来的向量重组为稀疏矩阵X

稀疏kernel



$$C = -A * B + C$$

该kernel是使用稀疏矩阵C减去稀疏矩阵A与稀疏矩阵B的乘积

分布式通信策略

Function GETRF
Rowptr 0 2 5 7
Colindex 0 2 0 1 2 1 2
val 2 3 6 5 9 45 4

0	1	3	5
0	0	1	1 2
1	3	1	9 1

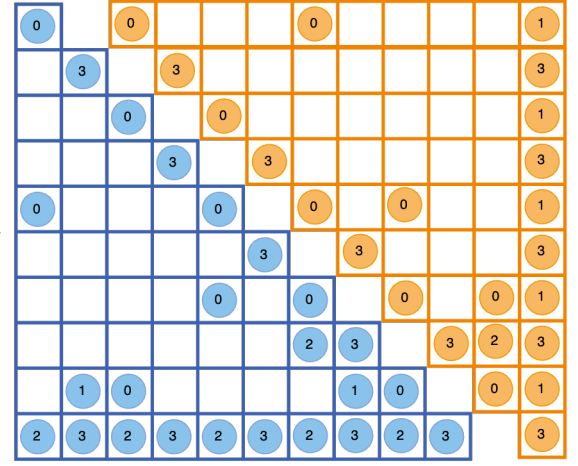
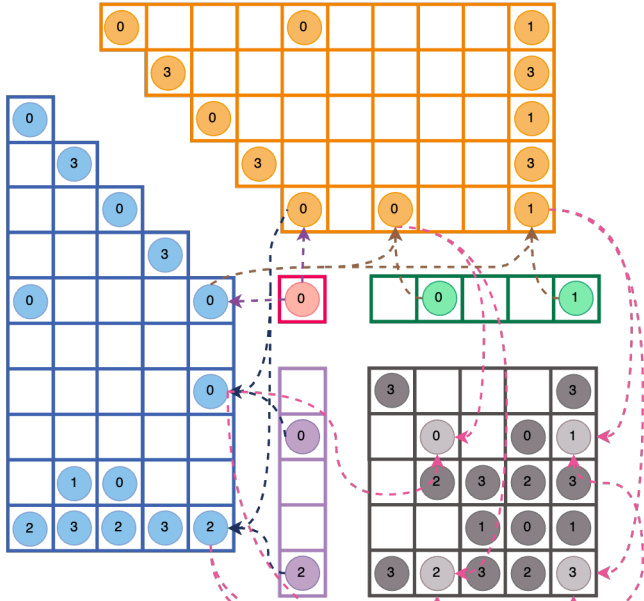
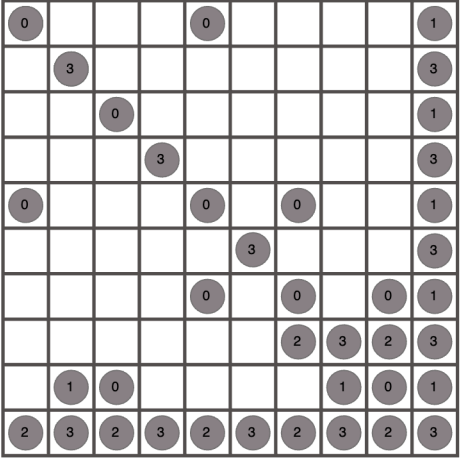
0	2	3	4
0	2	1	2
2	3	5	4

URowptr 0 3 5 7 9 12 14 16 19 22 23

Function GESSM
Rowptr 0 1 3 5
Colindex 0 0 1 1 2
val 1 3 1 9 1

0	1	3	4
0	0	2	2
3	7	4	2

0	1	3	5
0	0	2	0 2
3	16	4	63 38



Function TSTRF
Rowptr 0 1 1 3
Colindex 0 0 2
val 9 3 4

0	2	3	4
0	2	1	2
2	3	5	4

0	2	2	4
0	2	0	2
18	27	6	16

Function SSSSM
Rowptr 0 1 1 3
Colindex 0 0 2
val 9 3 4

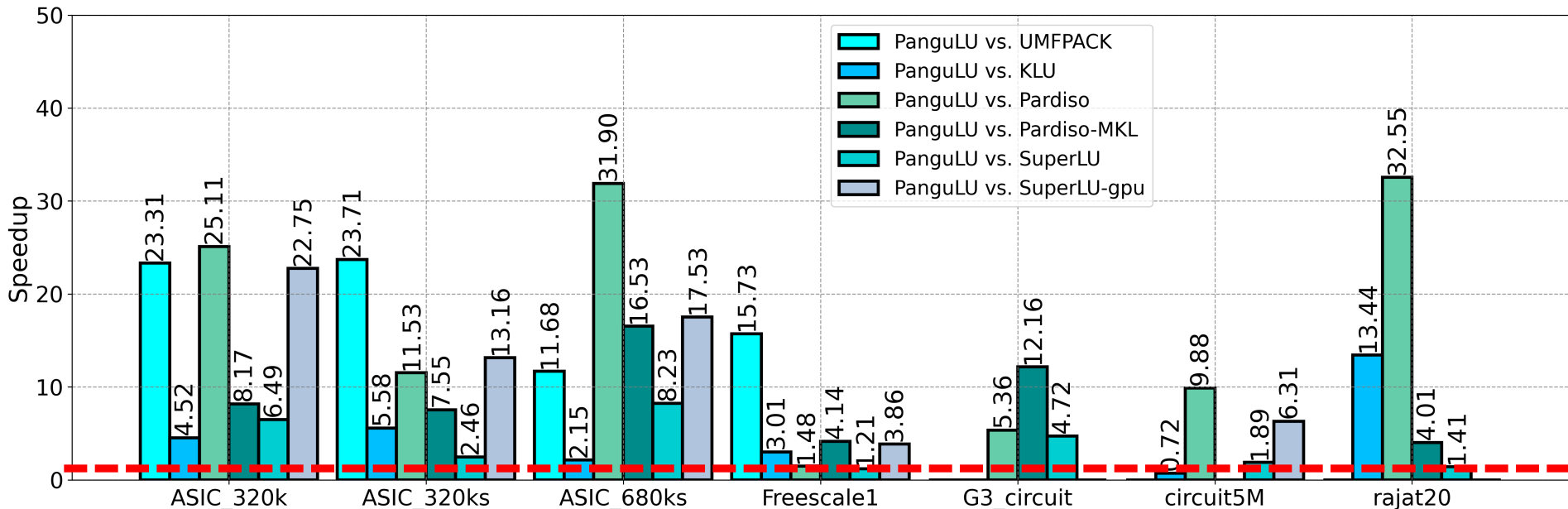
0	1	3	4
0	0	2	2
3	7	4	2

0	2	3	5
0	1	2	0 2
40	9	36	50 8

0	2	3	5
0	1	2	0 2
13	9	36	41 0

Viewer does not support full SVG 1.1

PanguLU的性能



GPU实验平台：4个节点，每个节点使用两个Intel Xeon Silver 4210@2.2GHz,以及8个RTX 2060 super

CPU实验平台：AMD 2nd EPYC 7702 , 2x64c@2GHz,2x32MB L2,2x256MB L3,2x8ch 3200MHz DDR4

观察（总结）

- 观察1：众核处理器的出现放大了稀疏矩阵结构给算法性能带来的影响，引入了新的研究内容。
- 观察2：SpMV、SpTRSV、SpGEMM等稀疏BLAS算法优化的追求依然是“高可扩展+高性能+高实用”，分块数据结构和算法可能是一个可行的解决方案。
- 观察3：稀疏BLAS是可以被有效用于稀疏LU分解的。
- 观察4：异构和分布式系统上的稀疏计算还有很多工作可做。

T k u !

0	4	8	9
---	---	---	---

A y Q s n s ?

0	2	4	7	11	12	13
---	---	---	---	----	----	----