LaCSR5：基于 CSR5 的龙芯 3A5000 上的高效 SpMV

汇报人 项强

队伍名 这是一支队

队伍编号 202310246111082

20 级 人工智能 项强

队员 20 级 软件工程 钱逸云

20 级 人工智能 徐哲

指导老师 张凯

学校 复旦大学

目录

项目目标

- **SpMV(Sparse Matrix-Vector Multiplication)**：

$$\mathbf{y} = \mathbf{A}\mathbf{x}$$

- 稀疏矩阵可能具有**各种稀疏结构**，所以选择适当的算法来实现和优化 SpMV 是一个具有挑战性的问题。
- 本题要求在 CSR 格式上进行改进和优化 SpMV 算法，我们在龙芯 3A5000 上选用**CSR5 格式及其 SpMV 算法** [11] 进行实现，原因以及对比分析如下。

CSR(Compressed Sparse Row) 格式

- 为了提高 SpMV 操作的效率，稀疏矩阵的存储格式需要特殊设计，以避免**不必要的零元素存储和运算**；
- 本文主要关注**CSR(Compressed Sparse Row)**格式的改进和优化；CSR 格式的性能问题主要来源于由行长度不均匀导致的稀疏矩阵的**负载不均衡**。

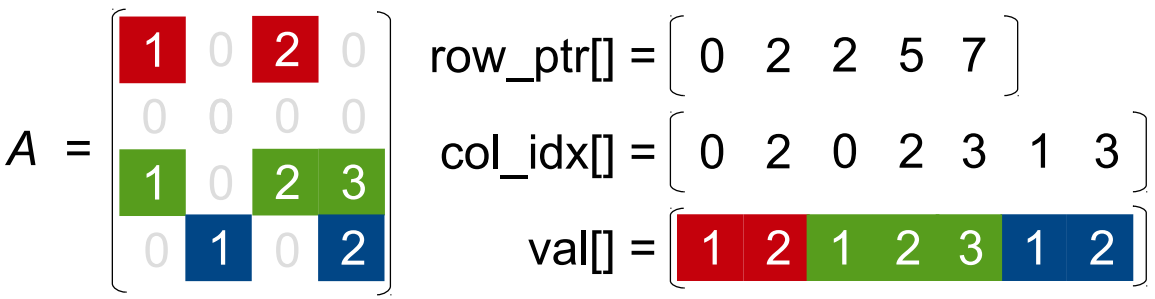


Figure: CSR 格式示意图

CSR 格式目前改进策略

- 数据流式传输 [9]
- 内存合并 [8]
- 数据重新排序或重构 [10, 12, 16]
- 静态或动态分箱 [1, 9]
- 动态并行性 [1]

其他稀疏矩阵存储格式和 SpMV 优化

- **具有块结构的稀疏矩阵** [2, 4–6, 12, 15] 为有块结构的矩阵设计特殊的存储格式和 SpMV 算法；但对于许多不具有块结构的矩阵，试图提取块信息耗时较长且效果有限。且在没有块结构的矩阵中，这种格式的性能并不理想。
- **分段求和方法** [14, 15] 用于加速求和操作，但需要将矩阵存储在类 COO 的格式中才有效果；而在 CSR 格式上性能很差。
- **人工智能** [3, 13] 训练模型自动选择给定稀疏矩阵的最佳格式；但选择过程耗时较长，选项有限，而且训练需要大量不同的稀疏矩阵训练集。

文献调研

[1] Ashari, Arash ; Sedaghati, Naser ; Eisenlohr, John ; Parthasarathy, S. ; Sadayappan, Ponnuswamy: Fast Sparse Matrix-Vector Multiplication on GPUs for Graph Applications. In: *International Conference for High Performance Computing, Networking, Storage and Analysis, SC* 2015 (2015), 01, S. 781–792

[2] Ashari, Arash ; Sedaghati, Naser ; Eisenlohr, John ; Sadayappan, P.: An Efficient Two-Dimensional Blocking Strategy for Sparse Matrix-Vector Multiplication on GPUs. In: *Proceedings of the 28th ACM International Conference on Supercomputing*. New York, NY, USA : Association for Computing Machinery, 2014 (ICS '14), S. 273–282. – URL https://doi.org/10.1145/2597652.2597678. – ISBN 9781450326421

[3] Ashoury, Mina ; Loni, Mohammad ; Khunjush, Farshad ; Daneshtalab, Masoud: *Auto-SpMV: Automated Optimizing SpMV Kernels on GPU*. 2023

[4] Buluç, Aydin ; Fineman, Jeremy T. ; Frigo, Matteo ; Gilbert, John R. ; Leiserson, Charles E.: Parallel Sparse Matrix-Vector and Matrix-Transpose-Vector Multiplication Using Compressed Sparse Blocks. In: *Proceedings of the Twenty-First Annual Symposium on Parallelism in Algorithms and Architectures*. New York, NY, USA : Association for Computing Machinery, 2009 (SPAA '09), S. 233–244. – URL https://doi.org/10.1145/1583991.1584053. – ISBN 9781605586069

[5] Buluç, Aydin ; Williams, Samuel ; Oliker, Leonid ; Demmel, James: Reduced-Bandwidth Multithreaded Algorithms for Sparse Matrix-Vector Multiplication. In: *2011 IEEE International Parallel & Distributed Processing Symposium*, 2011, S. 721–733

[6] Choi, Jee W. ; Singh, Amik ; Vuduc, Richard W.: Model-Driven Autotuning of Sparse Matrix-Vector Multiply on GPUs. In: *SIGPLAN Not.* 45 (2010), jan, Nr. 5, S. 115–126. – URL https://doi.org/10.1145/1837853.1693471. – ISSN 0362-1340

[7] Davis, Timothy A. ; Hu, Yifan: The University of Florida Sparse Matrix Collection. In: *ACM Trans. Math. Softw.* 38 (2011), dec, Nr. 1. – URL https://doi.org/10.1145/2049662.2049663. – ISSN 0098-3500

[8] Deng, Yangdong ; Wang, Bo D. ; Mu, Shuai: Taming irregular EDA applications on GPUs. In: *2009 IEEE/ACM International Conference on Computer-Aided Design - Digest of Technical Papers*, 2009, S. 539–546

[9] Greathouse, Joseph L. ; Daga, Mayank: Efficient Sparse Matrix-Vector Multiplication on GPUs Using the CSR Storage Format. In: *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2014, S. 769–780

[10] Guo, Dahai ; Gropp, William: Adaptive Thread Distributions for SpMV on a GPU. In: *Proceedings of the Extreme Scaling Workshop*. USA : University of Illinois at Urbana-Champaign, 2012 (BW-XSEDE '12)

[11] Liu, Weifeng ; Vinter, Brian: *CSR5: An Efficient Storage Format for Cross-Platform Sparse Matrix-Vector Multiplication*. 2015

CSR5 格式的优点

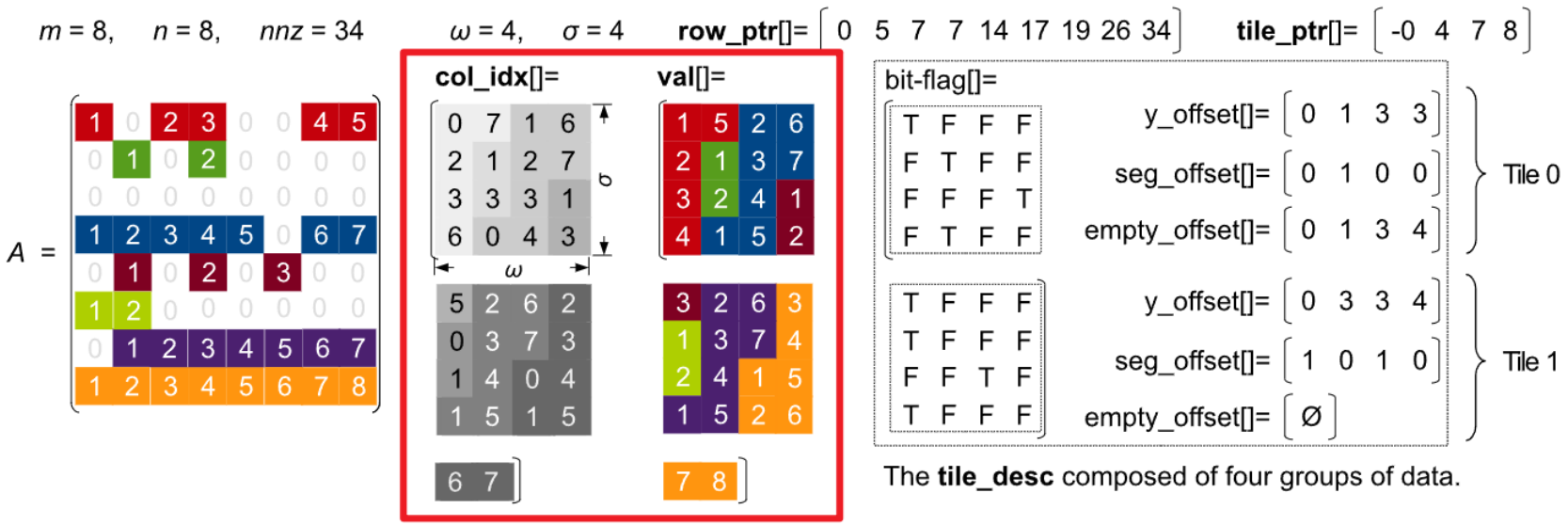
- CSR5 格式是一种**直接扩展 CSR 格式**的新格式，仅添加两组额外的辅助信息，且通常比 CSR 格式中的原始三组信息要短得多；
- CSR5 格式对**SIMD 友好**；
- 可以同时为**规则和不规则矩阵**带来稳定的高性能。

贡献

- 经过对相关文献的深入调研和分析，我们选择在龙芯 3A5000 上使用 CSR5 格式和基于 CSR5 的 SpMV 算法；
- 我们设计和实现了**LaCSR5-SpMV 算法**，一种面向龙芯处理器体系结构的稀疏矩阵乘向量算法，其利用龙芯的 SIMD 指令优化了计算过程，同时充分利用了龙芯的多核心特性；
- 经过对龙芯 3A5000 芯片的分析，我们对 CSR5 格式的参数进行了调优，使其适配龙芯 3A5000 的硬件特性，以达到更高的性能；
- 在 14 个规则稀疏矩阵和 10 个不规则稀疏矩阵上进行了 LaCSR5-SpMV 算法的性能测试，与 CSR-SpMV 算法相比，我们的算法达到了最高 2.78 倍的性能提升。

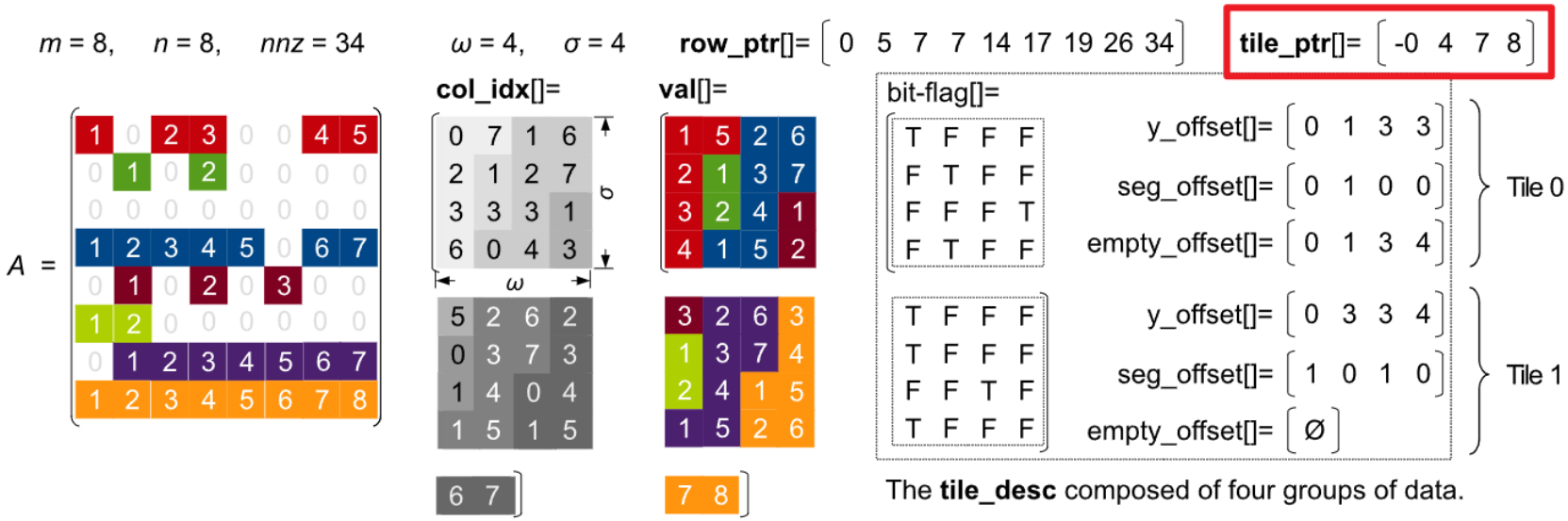
目录

基本数据排布



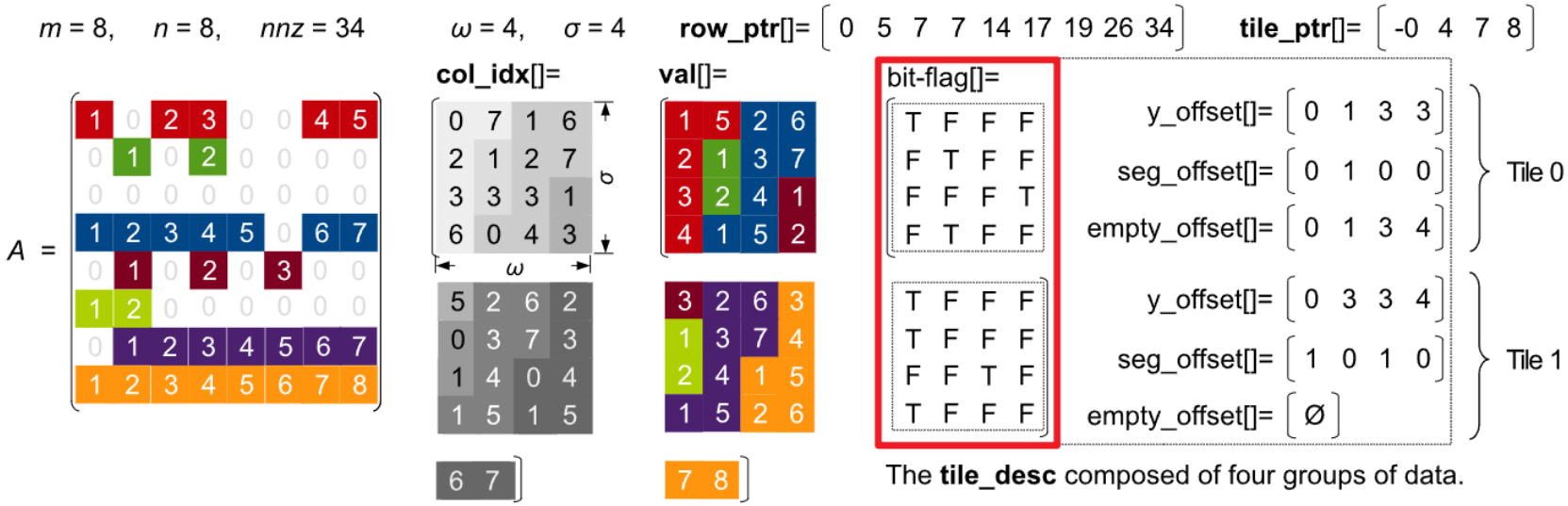
为了在具有任何稀疏结构的矩阵中实现接近最优的负载平衡，我们首先将所有非零元素均匀地分割成**多个相同大小的分块**。
因此，在执行并行 SpMV 操作时，计算核心可以计算一个或多个分块，核心的每个 SIMD 通道可以处理一个分块的一列。

分块指针信息 (Tile Pointer)



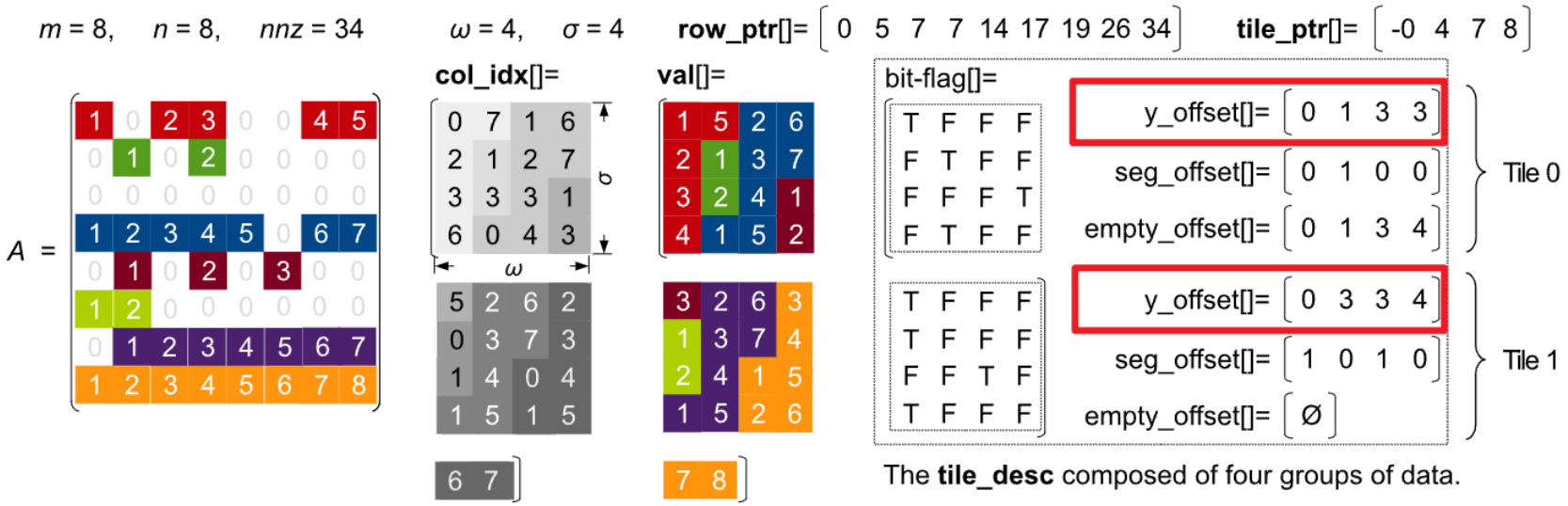
- 存储了**每个分块中第一个矩阵行的行索引**，指示了将其分段和存储到向量 y 的起始位置；
- `tile_ptr` 数组的长度为 $p+1$，其中 $p = \lceil nnz/(\omega\sigma) \rceil$ 为分块的数量；
- 对于具有空行的分块，特殊处理。

分块描述符信息 bit_flag



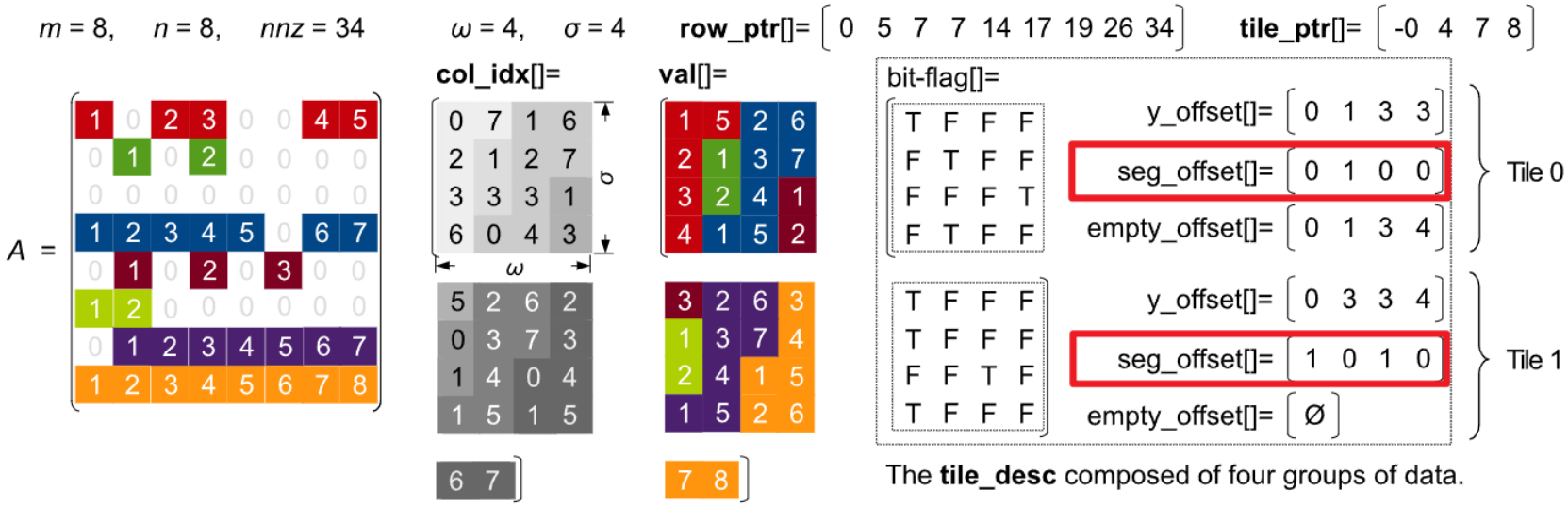
- 用于指示一个元素是否是矩阵行的第一个非零元素；
- 长度为 $\omega \times \sigma$；
- 每个分块的 `bit_flag` 的第一个元素设置为 `TRUE`，以封闭从顶部开始的第一个段，并使分块彼此独立。

分块描述符信息 y_offset



- 用于指示每一列存储其本地分段和的起始点位置；
- 长度为 ω；
- 对于第 *tid* 个分块中的第 *i* 列，通过计算 `tile_ptr[`*tid*`]` + `y_offset[`*i*`]`，得到该列在 *y* 中的存储位置。

分块描述符信息 seg_offset

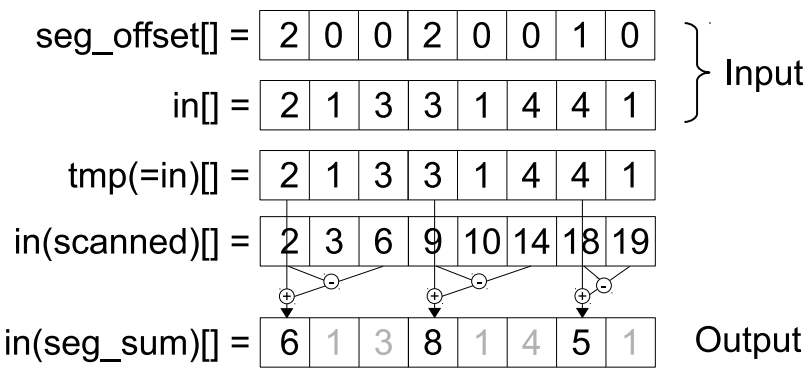


- 用于加速分块内的**局部分段和**操作；
- 长度为 ω；
- `seg_offset[i]` 表示与第 i 列最末一个分段为同一行的右邻连续列的数量。

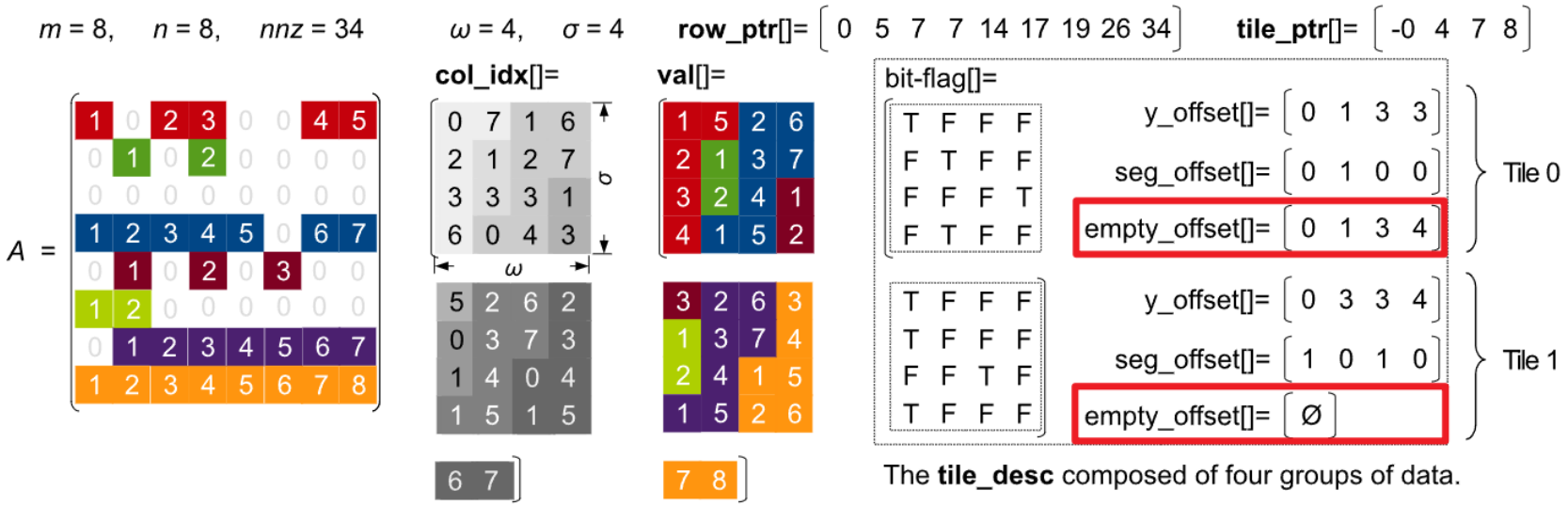
快速分段求和

Algorithm 1 利用 `seg_offset` 快速分段求和

```
1:  function FAST_SEGMENTED_SUM(*in, *seg_offset)
2:      length ← SIZEOF(*in)
3:      MALLOC(*tmp, length)
4:      MEMCPY(*tmp, *in)
5:      INCLUSIVE_PREFIX_SUM_SCAN(*in)
6:      for i = 0 to length − 1 in parallel do
7:          in[i] ← in[i+seg_offset[i]] − in[i] + tmp[i]
8:      end for
9:      FREE(*tmp)
10: end function
```

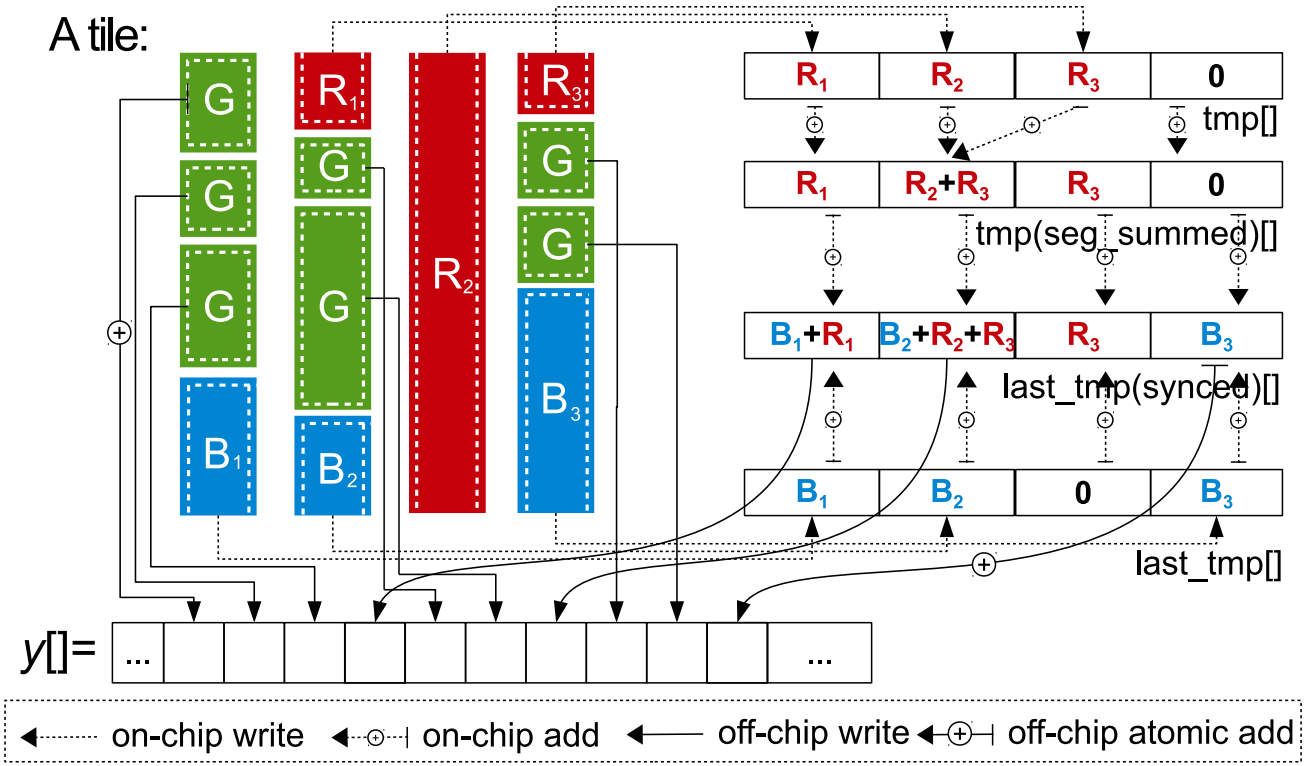


分块描述符信息 empty_offset



- 用于分块中包含空行的情况下，指示分段和存储到 y 中正确的位置；
- 长度为分块中段的数量，不超过 $\omega \times \sigma$；
- 指示每个段真实的相对行所索引。

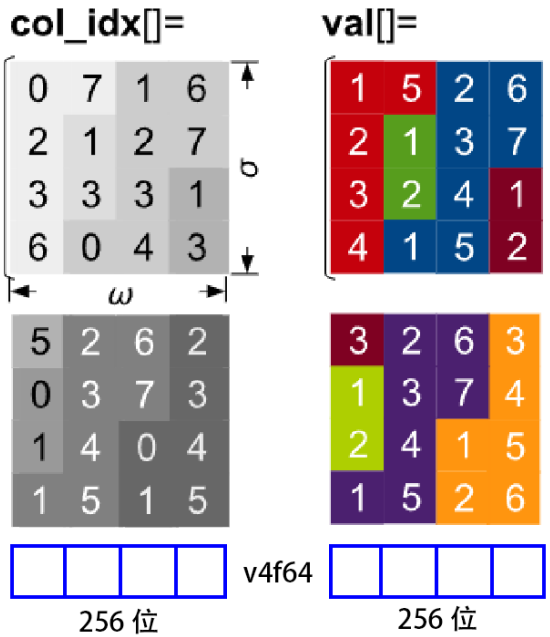
LaCSR5-SpMV 算法



- 绿色段的分段和可以直接保存到 y 中，无需任何同步，因为可以使用 `tile_ptr` 和 `y_offset` 来计算得到其行索引。
- 红色段和蓝色段不是完整的段，需要进一步将它们的分段和与其他段相加。

目录

自适应参数调优 - 分块宽度 ω



$$\omega = \frac{\text{向量寄存器长度}}{sizeof(VT)}$$

我们为分块每一列分配一个核心上的 SIMD 通道，充分利用向量指令。因此，分块的宽度 ω 应该等于**每个核心上的 SIMD 通道的数量**。

对于龙芯 3A5000，其扩展向量寄存器为 256 位，所以对于单精度和双精度 SpMV，我们分别设置 $\omega = 8$ 和 $\omega = 4$；

单精度前缀和 SIMD 优化

```
// inclusive prefix-sum scan
inline __m256 hscan_lasx(__m256 in256)
{
    __m256 t0;
    //shift1 + add
    t0 = (__m256)__lasx_xvbsll_v(in256, 4);
    t0[4] = in256[3];
    in256 = __lasx_xvfadd_s(in256, t0);
    //shift2 + add
    t0 = (__m256)__lasx_xvbsll_v(in256, 8);
    t0[4] = in256[2], t0[5] = in256[3];
    in256 = __lasx_xvfadd_s(in256, t0);
    //shift3 + add
    t0 = (__m256)__lasx_xvpermi_q(in256, in256, 0b00101001);
    in256 = __lasx_xvfadd_s(in256, t0);
    return in256;
}
```

我们利用 `xvbsll.v`, `xvfadd.s`, `xvpermi.q` 等指令，实现了高效的单精度前缀和。

快速分段求和 SIMD 优化

Algorithm 2 利用 `seg_offset` 快速分段求和

```
1: function FAST_SEGMENTED_SUM(*in, *seg_offset)
2:     length ← SIZEOF(*in)
3:     MALLOC(*tmp, length)
4:     MEMCPY(*tmp, *in)
5:     INCLUSIVE_PREFIX_SUM_SCAN(*in)
6:     for i = 0 to length − 1 in parallel do
7:         in[i] ← in[i+seg_offset[i]] − in[i] + tmp[i]
8:     end for
9:     FREE(*tmp)
10: end function
```

$$\texttt{in}[i] \leftarrow \texttt{in}[i+\texttt{seg\_offset}[i]] - \texttt{in}[i] + \texttt{tmp}[i]$$

```
// fast segmented sum
tmp_sum256 = sum256;
sum256 = hscan_lasx(sum256);

scansum_offset256i = __lasx_xvadd_w(scansum_offset256i, (__m256i)(v8i32{0, 1, 2, 3, 4, 5, 6, 7}));

sum256 = __lasx_xvfsub_s((__m256)(__lasx_xvperm_w((__m256i)(sum256), scansum_offset256i)), sum256);
sum256 = __lasx_xvfadd_s(sum256, tmp_sum256);
```

$i+\texttt{seg\_offset}[i]$

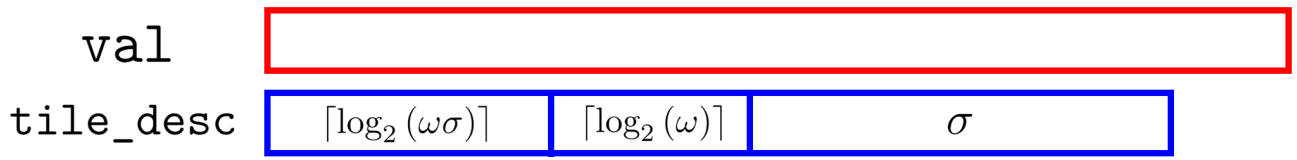
$-\texttt{in}[i]$

$+\texttt{tmp}[i]$

自适应参数调优 - 分块高度 σ

$$\begin{cases} \lceil \log_2(\omega\sigma) \rceil + \lceil \log_2(\omega) \rceil + \sigma \leq sizeof(VT) \\ 16 \cdot \omega\sigma \cdot sizeof(VT) + 8 \times 32\omega \leq 64K \end{cases}$$

其中VT 为矩阵元素值存储的数据类型。

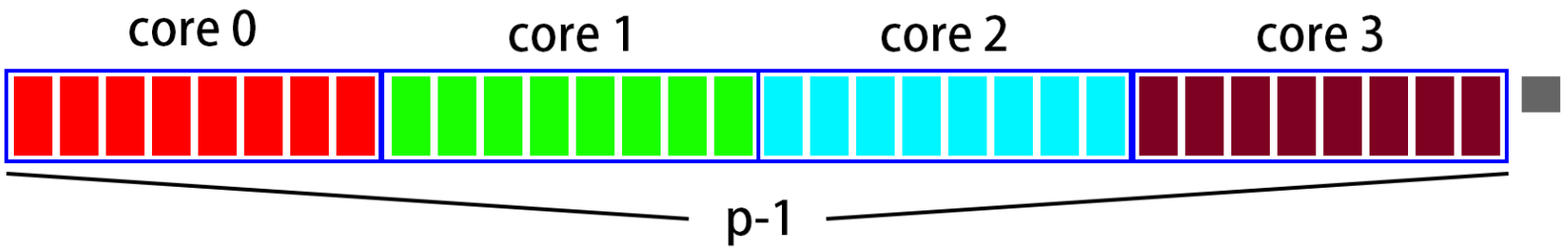


考虑两点：

- 保证**足够的位数**存储辅助信息 `tile_desc`
  - `y_offset`$[i] \leq \omega\sigma$，故需 $\lceil \log_2(\omega\sigma) \rceil$ 位；
  - `seg_offset`$[i] \leq \omega$，故需 $\lceil \log_2(\omega) \rceil$ 位；
  - `bit_flag` 为分块的每个列存储了 σ 个 1 位标志，故需 σ 位。
- 充分利用龙芯 3A5000 的**cache 容量**

所以在龙芯 3A5000 上，对于单精度和双精度 SpMV，我们分别设置 $\sigma = 21$ 和 $\sigma = 16$。

LaCSR5-SpMV 算法并行细节



- 总共划分成 $p = \lceil nnz/(\omega\sigma) \rceil$ 个分块；
- 利用 OpenMP 并行化每个分块的计算；
- 龙芯 3A5000 上有 4 个核心，每个核心 1 个线程，共 4 个线程，所有分块均匀分到 4 个线程上。

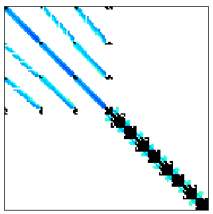
目录

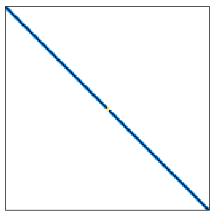
测试所用矩阵

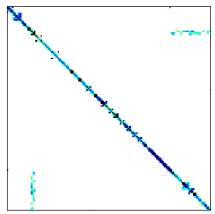
Id	Name	*Dimensions*	*nnz*	*nnz* per row (min, avg, max)
r1	Dense	2K×2K	4.0M	2K, 2K, 2K
r2	Uniform	10K×10K	10M	1K, 1K, 1K
r3	FEM/Spheres	83K×83K	6.0M	1, 72, 81
r4	FEM/Cantilever	62K×62K	4.0M	1, 64, 78
r5	Wind Tunnel	218K×218K	11.6M	2, 53, 180
r6	Protein	36K×36K	4.3M	18, 119, 204
r7	Epidemiology	526K×526K	2.1M	2, 3, 4
r8	FEM/Harbor	47K×47K	2.4M	4, 50, 145
r9	FEM/Ship	141K×141K	7.8M	24, 55, 102
r10	Economics	207K×207K	1.3M	1, 6, 44
r11	FEM/Accelerator	121K×121K	2.6M	0, 21, 81
r12	Circuit	171K×171K	959K	1, 5, 353
r13	Ga41As41H72	268K×268K	18.5M	18, 68, 702
r14	Si41Ge41H72	186K×186K	15.0M	13, 80, 662
i1	Webbase	1M×1M	3.1M	1, 3, 4.7K
i2	LP	4K×1.1M	11.3M	1, 2.6K, 56.2K
i3	Circuit5M	5.6M×5.6M	59.5M	1, 10, 1.29M
i4	eu-2005	863K×863K	19.2M	0, 22, 6.9K
i5	in-2004	1.4M×1.4M	16.9M	0, 12, 7.8K
i6	mip1	66K×66K	10.4M	4, 155, 66.4K
i7	ASIC_680k	683K×683K	3.9M	1, 6, 395K
i8	dc2	117K×117K	766K	1, 7, 114K
i9	FullChip	2.9M×2.9M	26.6M	1, 9, 2.3M
i10	ins2	309K×309K	2.8M	5, 9, 309K

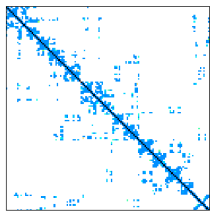
除了 r1 和 r2 矩阵，其他矩阵均来自 the University of Florida Sparse Matrix Collection [7]。

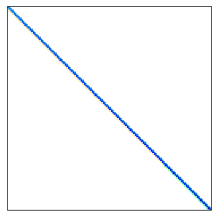
不同稀疏特征的测试矩阵

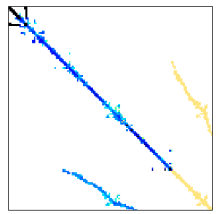

r3

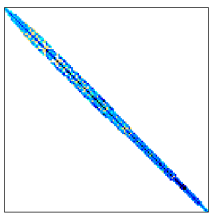

r4

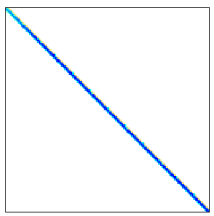

r5

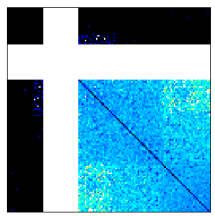

r6

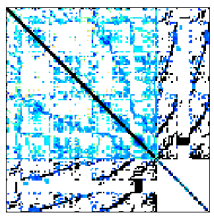

r7

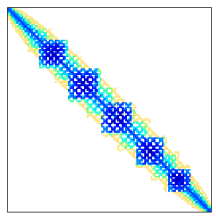

r8

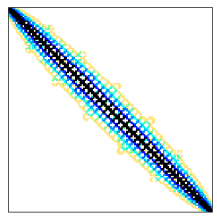

r9

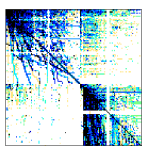

r10


r11

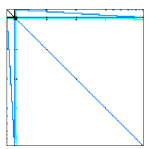

r12

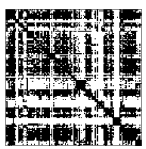

r13


r14

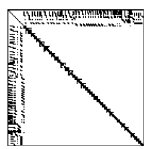

i1

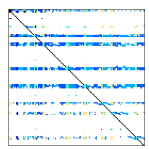
i2

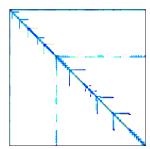

i3

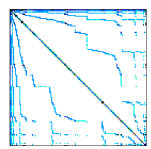

i4

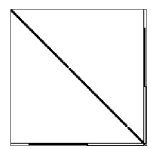

i5


i6

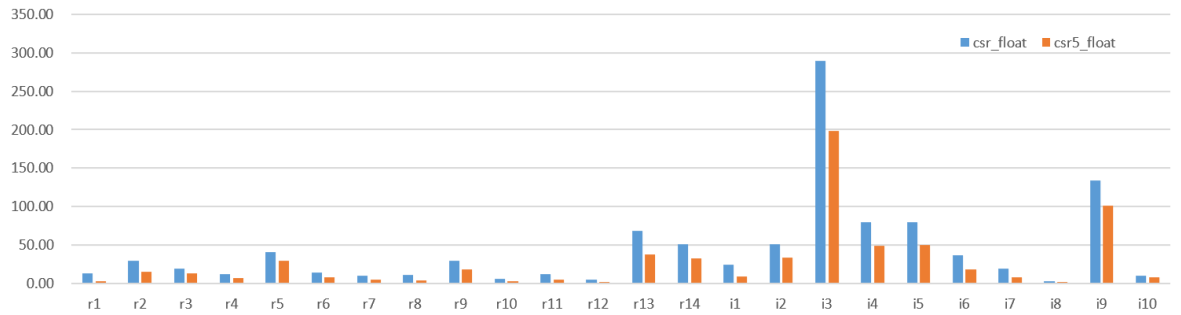

i7


i8

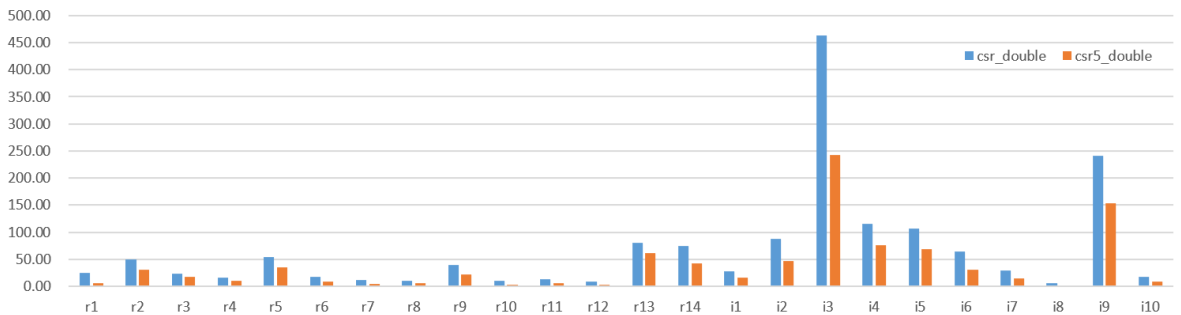

i9


i10

SpMV 测试结果



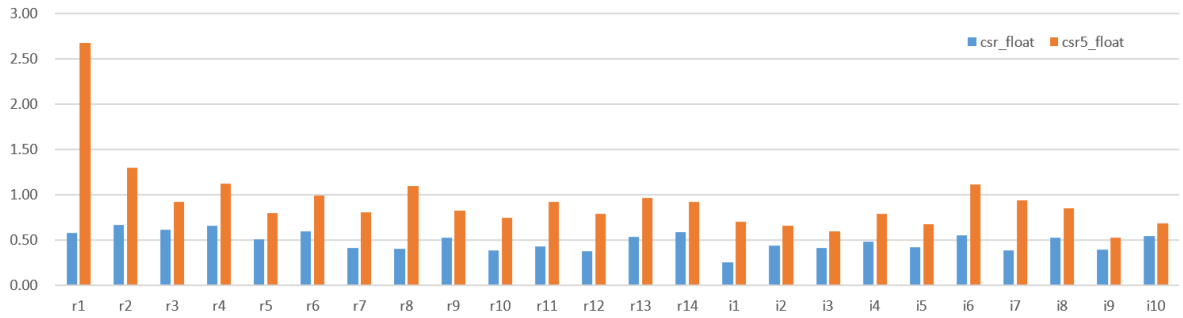
Time of float



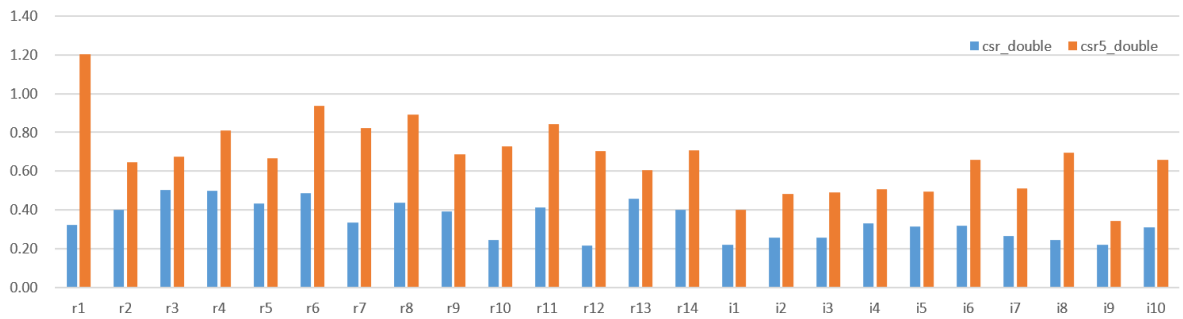
Time of double

对于除 r1（稠密矩阵）之外的 23 个稀疏矩阵，单精度和双精度 LaCSR5-SpMV 的加速比均值分别为**1.80x**和**1.95x**。

SpMV 测试结果



GFlops of float



GFlops of double

对于 13 个规则的稀疏矩阵，单精度和双精度 LaCSR5-SpMV 的加速比平均值分别为**1.84x**和**1.96x**。
对于 10 个不规则的稀疏矩阵，单精度和双精度 LaCSR5-SpMV 的加速比平均值分别为**1.75x**和**1.92x**。

格式转换时间的迭代分析

Precision	float			double		
Metrics	Preprocess ratio	Speedup of #iter=50	Speedup of #iter=500	Preprocess ratio	Speedup of #iter=50	Speedup of #iter=500
regular	3.37x	1.73x	1.83x	3.20x	1.84x	1.95x
irregular	3.09x	1.65x	1.74x	2.31x	1.83x	1.91x

Table: 格式转换时间的迭代加速比

$$\text{迭代加速比} = \frac{n \cdot T_{spmv}^{csr}}{T_{pre}^{new} + n \cdot T_{spmv}^{new}}$$

其中 T_{spmv}^{csr} 是一个单次 CSR-SpMV 操作的执行时间，T_{pre}^{new} 是预处理时间，T_{spmv}^{new} 是使用新格式的单次 SpMV 时间。
在 $n = 50$ 时，加速比已接近单独 SpMV 的加速比；
在 $n = 500$ 时，格式转换时间已经可以**忽略不计**。

消融实验：分块高度 σ 的影响

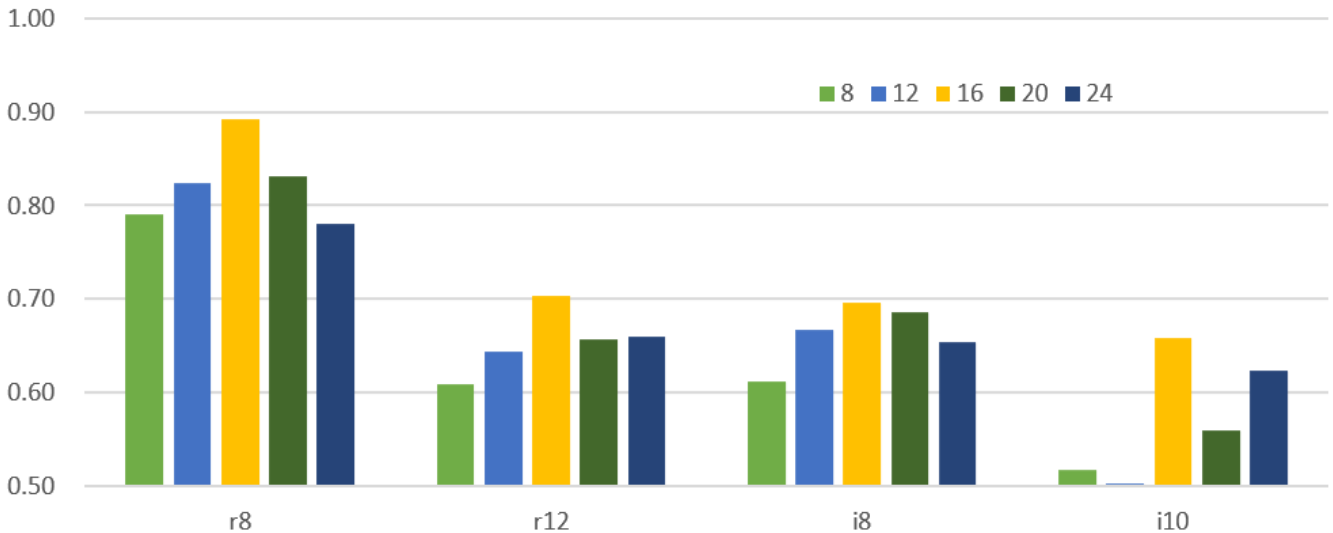


Figure: 控制 ω 不变，改变 σ 的值，采用双精度值，在 r8、r12、i8 和 i10 矩阵上分别进行 3 次实验，观察 LaCSR5-SpMV 的性能变化。柱状图每一簇从左到右分别代表 σ 的值为 8、12、16、20、24 时的 GFlops。

由图可见，$\sigma = 16$ 是最优的选择（黄色柱）。

- 当 σ 过小时，不能充分利用 cache 的容量；
- 当 σ 过大时，`tile_desc` 数组的长度会过长，导致额外的存储开销，且过大的分块会造成过多的片外访存。

LaCSR5：基于 CSR5 的龙芯 3A5000 上的高效 SpMV

感谢专家评委的聆听和指导!

汇报人 项强